The Dissertation Committee for Soheil Ghanbarzadeh
certifies that this is the approved version of the following dissertation:

Pore Fluid Percolation and Flow in Ductile Rocks

Committee:

Maša Prodanović, Supervisor

Marc A. Hesse, Co-Supervisor

Kamy Sepehrnoori

Daniel Ebrom

Steven L. Bryant

David DiCarlo

Pore Fluid Percolation and Flow in Ductile Rocks

by

Soheil Ghanbarzadeh, B.S., M.S.

DISSERTATION

Presented to the Faculty of the Graduate School of

The University of Texas at Austin

in Partial Fulfillment

of the Requirements

for the Degree of

DOCTOR OF PHILOSOPHY

THE UNIVERSITY OF TEXAS AT AUSTIN

August 2016

Dedicated to my dear and lovely wife, Parisa,

my hero, my son, Neekon

and my wonderful parents, Masoumeh and Hamid,

for their endless love, support and encouragement.

Acknowledgments

I would like to thank all those who inspired and helped me throughout this journey and made this dissertation possible.

I express my sincere gratitude to my supervisors, Dr. Maša Prodanović and Dr. Marc Andre Hesse for their continuous guidance, support, and encouragement throughout this project. They were supportive, encouraging and willing to share their fascinating ideas which were crucial to the success of this dissertation. I am particularly thankful for the collaborative environment and numerous opportunities they gave me to present my work in several conferences and for the endless hours we spent to publish several research papers. I am indebted to Dr. James E. Gardner for his endless help and fruitful discussions. I would like to thank Dr. Martin P.A. Jackson, Dr. Richard Ketcham and all the staff in UTCT. I also appreciate invaluable comments and feedback from my committee members, Dr. Kamy Sepehrnoori, Dr. Daniel Ebrom, Dr. Steven Bryant and Dr. David DiCarlo.

This research is entirely supported by the Statoil Fellows Program at The University of Texas at Austin. I gratefully acknowledge the financial support from Statoil during my school years, as well as two summer internships in Statoil that significantly helped me throughout this project. I specifically am thankful to Dr. Daniel Ebrom, Dr. Allison K. Thurmond, Mr. James Kalinec, Mr. Larry Adamson, Dr. Yaping Zhu, Mrs. Brit Ragnhildstveit, Dr. Robert Hunsdale, Dr. Tore M. Løseth in Statoil for their endless help and support during the course of this project. I would also like to acknowledge the staff of the Department of Petroleum and Geosystems Engineering, Roger Terzian, Tim Guinn, Frankie Hart, Amy D. Stewart, Sandy Taylor, Jin Lee and Leilani Swafford for their support to facilitate our academic life.

I am indebted to many of my friends and colleagues who shaped this dissertation. Specifically I would like to thank Javad Behseresht, Saeedeh Mohebinia, Mohsen Rezaveisi, Aboulghasem Kazeminia, Jake Jordan, Kiran Sethaye and Rahul Verma for their indispensable help and knowledge sharing about numerical modeling and working with level set method. I also enjoyed technical discussions with my colleagues Kiran Sethaye, Daria Akhbari, Maryam Mirabolghasemi, Mahmood Shakiba, Ali Abouie and Collin MaNeece. I would like to thank my awesome friends Amir Kianinejad, Amin Anvari, Ali Abouie, Mahmood Shakiba, Behzad Eftekhari, Mohammad Reza Shafiei, Mehdi Haddad, whose moral support has been critical along my graduate studies, as well as Siyavash Motealleh and many others whom I did not cite explicitly.

At the end, I would like to express my deepest appreciation to my beloved wife, Parisa, without whose support, I would never have found the courage to overcome all the difficulties during this research. She was always caring, supportive and patient with my long work hours. I also want to thank my hero, my son, Neekon, for just being here with me and giving meaning to my life. My heartfelt gratitude goes to my dear parents Masoumeh and Hamid for their abundant love and encouragements.

Pore Fluid Percolation and Flow in Ductile Rocks

Publication No. _____

Soheil Ghanbarzadeh, Ph.D.
The University of Texas at Austin, 2016

Supervisor: Maša Prodanović
Co-Supervisor: Marc A. Hesse

Ductile rocks have capacity to deform in response to large strains without macroscopic fracturing. Such behavior may occur in rocks that did not undergo diagenesis, in weak materials such as rock salt or at greater depths in all rock types where higher temperatures promote crystal plasticity and higher confining pressures suppress brittle fracture (partially molten rocks). The pore network topology and fluid distribution in ductile rocks are governed by textural equilibrium. Therefore, textural equilibrium controls the distribution of the liquid phase in many naturally occurring porous materials such as partially molten rocks and alloys, salt-brine and ice-water systems. In this dissertation, we present a level set method to compute an implicit representation of the liquid-solid interface in textural equilibrium with space-filling tessellations of multiple solid grains in three dimensions.

In ductile rocks, pore geometry evolves to minimize the solid-liquid interfacial energy while maintaining a constant dihedral angle, θ, at solid-liquid contact lines. Interfacial energy minimization with level set method is achieved by evolving the solid-liquid interface under surface diffusion to constant mean curvature surface. The liquid volume and dihedral angle constraints are added to the formulation using virtual convective and normal

velocity terms. This results in a initial value problem for a system of nonlinear coupled PDEs governing the evolution of the level sets for each grain. A domain decomposition scheme is devised to restrict the computational domain of each grain to few grid points around the grain. The coupling between the interfaces is achieved in a higher level on the original computational domain.

Our results show that the grain boundaries with the smallest area can be fully wetted by the pore fluid even for $\theta > 0$. This was previously not thought to be possible at textural equilibrium and reconciles the theory with experimental observations. Even small anisotropy in the fabric of the porous medium allows the wetting of these faces at very low porosities, $\phi < 3\%$. Percolation and orientation of the wetted faces relative to the anisotropy of the fabric are controlled by θ. We have studied the fluid percolation and percolation thresholds in regular and irregular media. The results show that the pore space is connected at any non-zero porosity when $\theta \leq 60^\circ$, and percolation threshold in an irregular media comprised of grains with different shapes and sizes is much higher than previously thought. Our results show that the pore network connectivity in ductile rocks is affected by the history of the systems and hysteresis determines the percolation when $\theta > 60^\circ$. We have also computed permeability of the pore networks in different porosities and dihedral angles for both regular and irregular media using Lattice Boltzmann method. Furthermore, we studied the effects of grain texture anisotropy on the permeability anisotropy.

Until recent years, rock salt has been considered to be impermeable as it seems to contains and keep gas inclusions for long time. Increasing energy demand and necessity of producing hydrocarbon reservoir enclosed or touched by salt deposit and also urgent need of safe repository sites for high-level nuclear waste have brought attention to research and study the porosity and permeability of natural rock salt. Rock salt in sedimentary basins has long been considered to be impermeable and provides a seal for hydrocarbon accumulations

in geological structures. The low permeability of static rock salt is due to a percolation threshold. However, deformation may be able to overcome this threshold and allow fluid flow. We confirm the percolation threshold in static experiments on synthetic salt samples with X-ray microtomography. We then analyze wells penetrating salt deposits in the Gulf of Mexico. The observed hydrocarbon distributions in rock salt require that percolation occurred at porosities considerably below the static threshold, due to deformation-assisted percolation. In general, static percolation thresholds may not always limit fluid flow in deforming environments.

Here we use pore-scale simulations of texturally equilibrated pore networks to study the possibility of core formation by porous flow in planetesimals. Rapid core formation in early planetary bodies is required by geochemical data from extinct radionuclides. The most obvious mechanism for metal-silicate differentiation is the segregation of dense core forming melts by porous flow. However, experimental observations show that the texturally equilibrated metallic melt resides in isolated pockets that prevent percolation towards the center. The proposed hypothesis in this dissertation is that the porosities can be large enough to exceed percolation threshold and allow metalic melt drainge to center. The melt network remains interconnected as drainage reduces the porosity below the percolation threshold and only 1-2% is trapped. X-ray microtomography of lodran-like meteorite NWA 2993 provides evidence that volume fraction of metallic phases can exceed this percolation threshold. Lattice Boltzmann simulations show that the permeability during drainage remains significant. A model for metal-silicate differentiation by porous flow in a viscously compacting planetesimal is also proposed and shows that the efficient core formation requires early accretion and is completed almost 2 Myr after the onset of melting.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Ductile Rocks

1.1.1 What Are Ductile Rocks?

Ductile rocks have capacity to deform in response to large strains without macroscopic fracturing. Such behavior may occur in rocks that did not undergo diagenesis, in weak materials such as rock salt or at greater depths in all rock types where higher temperatures promote crystal plasticity and higher confining pressures suppress brittle fracture (partially molten rocks). In general, rocks that can deform or even flow without macroscopic fracturing are considered ductile. Fig. 1.1 shows examples of the ductile rocks.

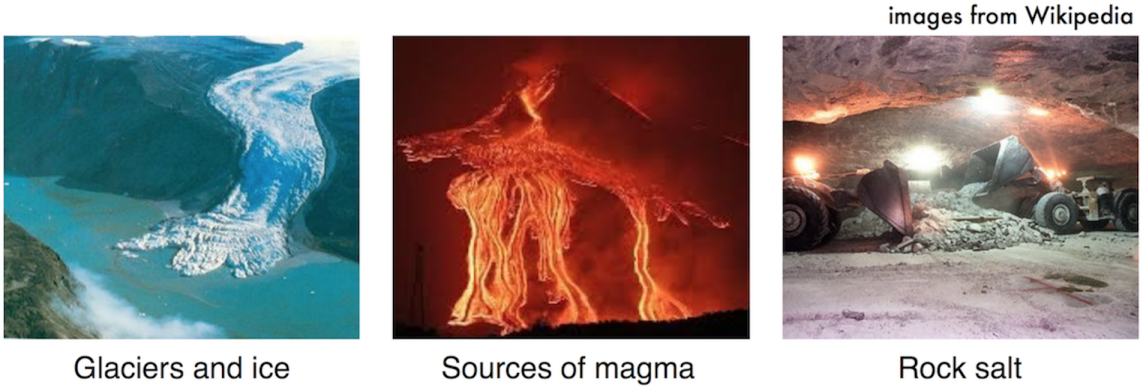


Figure 1.1: Polycrystalline ice at near melting temperature, partially molten rocks and rock salt are examples of the ductile rocks, i.e. the rocks that can flow.

One characteristic property that all ductile rocks have in common is that under certain conditions, there seems to be no transport limit in ductile rocks. In others word, under those conditions, there is no percolation threshold in ductile rocks and their pore

network percolates at any non-zero porosity (Fig. 1.2). Experimental data, presented in Fig. 1.2, shows permeability has a finite values at porosities below 1%. However, in clastic rocks there is a finite porosity that below it there is no percolation possible in the pore space. Experimental study on a large variety of the clastic rocks shows that there is no transport below 3% porosity (Fig. 1.3).

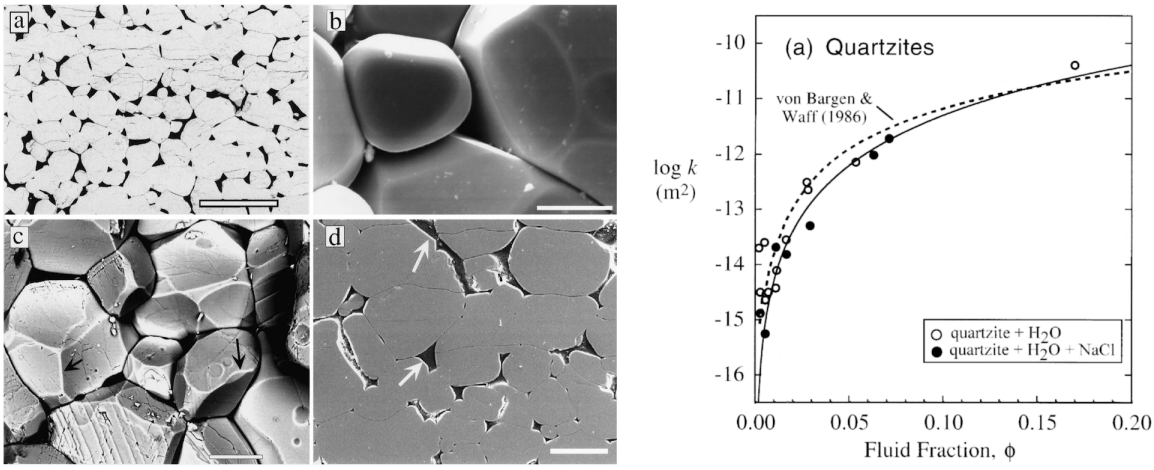


Figure 1.2: Percolation in ductile rocks. The connectivity of the pore fluid on the grain edges of the ductile rock in textural equilibrium allows the percolation in any non-zero porosity. Images from Wark and Watson (1998).

The pore network topology and fluid distribution in ductile rocks are governed by textural equilibrium. Textural equilibrium is the ultimate form of equilibrium in a two-phase aggregate. Two reactive materials usually reach thermal, then mechanical followed by chemical equilibrium. This state is called thermodynamic equilibrium. After reaching mechanical equilibrium, diffusive and textural equilibrium are achieved. In diffusive equilibrium, there is a balance of mass between phases and components and in textural equilibrium, interfaces minimize their interfacial energies. Therefore, textural equilibrium controls the ultimate distribution of the liquid phase in many naturally occurring porous materials such as partially molten rocks and alloys, salt-brine and ice-water systems. In these materials, pore geometry evolves to minimize the solid-liquid interfacial energy while

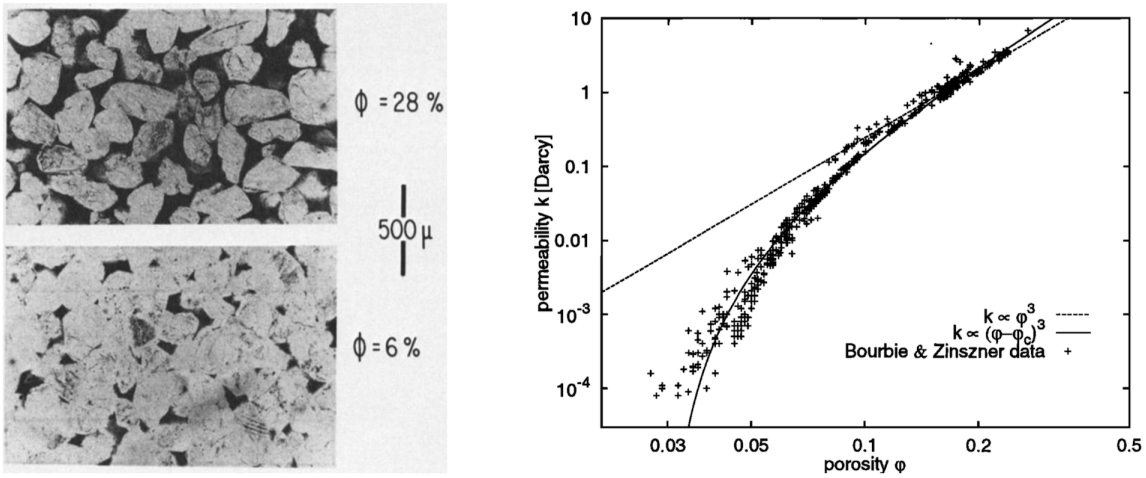


Figure 1.3: Percolation threshold in clastic rocks. The experimental data shows that there is no percolation at porosities below 3%. Figures from Bourbie and Zinszner (1985) and van der Marck (1999).

maintaining a constant dihedral angle, θ, at solid-liquid contact lines with a given volume (fraction) of liquid.

1.1.2 What Is Textural Equilibrium?

A porous medium is considered texturally equilibrated if the solid-liquid interface minimizes the surface energy density, γ_{sl}. For a system under hydrostatic pressure and with isotropic surface energies, textural equilibrium requires that the mean curvature, $\bar{\kappa}$, of the solid-liquid interface is constant (Smith, 1948; Beere, 1975). This interface divides space into two interpenetrating and nonintersecting subspaces, similar to bicontinuous cubic phases common in biological systems (Schwarz and Gompper, 2000; DiDonna and Kamien, 2002). In the porous media considered here, however, the solid phase is polycrystalline and crystallographically distinct grains introduce additional interfaces into the solid subspace (Fig. 1.4*a*). These solid-solid interfaces, i.e., the grain boundaries, are considered stationary on the timescale required to reach textural equilibrium, so that the solid-solid surface energy density, γ_{ss}, is not minimized. The pre-existing grain edge network of the

polycrystalline solid therefore imposes a structure on the pore space, which is commonly referred to as grain edge porosity (Tucker, 1979). The pre-existing grain boundaries also introduce contact lines along which solid-liquid and solid-solid interfaces meet at sharp angles (Fig. 1.4*b*). In a thermodynamically stable material at most three interfaces, separating two solid grains and the liquid, can meet at a contact line (Gibbs, 1957; Bulau et al., 1979). Mechanical equilibrium at such a contact line requires that, $\gamma_{ss} = 2\gamma_{sl}\cos(\theta/2)$, where θ is the dihedral angle.

An important property of texturally equilibrated porous media is that the pore network percolates at any porosity for $\theta \leq 60^\circ$ while a percolation threshold exists for $\theta > 60^\circ$ (Bulau et al., 1979). Assuming that all the interfacial energies are isotropic, the equilibrium value of the tip angle, ξ in Fig. 1.5, for fluid distributed in isolated pockets would be

$$\cos\xi = \left[\sqrt{3}\tan\frac{\theta}{2}\right]^{-1} \tag{1.1}$$

which is only a function of the dihedral angle value. The importance of Eq. 1.1 for liquid distribution low volume fractions is that the value of ξ increases from 0° for $\theta = 60^\circ$ to 90° for $\theta = 180^\circ$ and this equation does not have a real solution when $\theta < 60^\circ$ (von Bargen and Waff, 1986; Laporte and Provost, 2000) . This means that when dihedral angle is less than 60°, there is no tip angle and no isolated fluid pockets can form. Therefore the fluid is connected and pore space is percolating.

Textural equilibrium develops if the solid-liquid reaction kinetics are fast or time scales are long and is therefore common in geological systems and partially molten materials. The pore fluid in these systems is often a melt, which is studied after quenching it to a glass. In this case, the pore space is commonly referred to as the intergranular phase and considered part of the microstructure of the material (Clarke, 1989). The percolation of texturally equilibrated intergranular phases was initially studied in the context

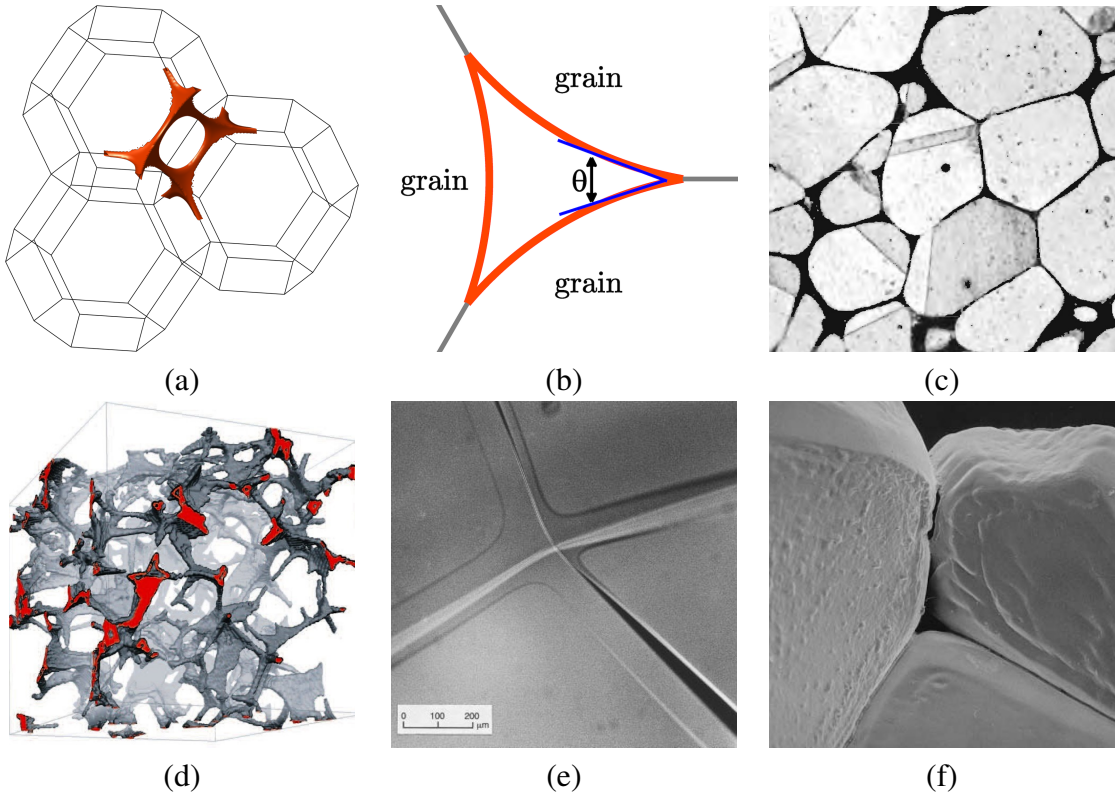


Figure 1.4: (a) Texturally equilibrated pore network with $\theta = 30°$ and $\phi = 1.5\%$ in a polycrystalline comprising truncated octahedral grains. (b) Definition of the dihedral angle, θ, in a cross-section of a channel along a three-grain corner. (c) Melt network with $\theta \approx 0°$ in a copper-silver alloy (Smith, 1948). (d) Melt network with $\phi = 5\%$ in an olivine-basalt aggregate (Zhu et al., 2011) used with permission from The American Association for the Advancement of Science, (e) Quadruple junction of a melt network between ice grains near 0°C (Rempel et al., 2001) used with permission from Nature Publishing Group, (f) Drained brine network in halite with $\theta \approx 45°$ at 1.5kbar and 395°C (Lewis and Holness, 1996), used with permission from the Geological Society of America

of two-phase alloys where it controls electrochemical properties (Fig. 1.4*c*) (Smith, 1948). In liquid phase sintering textural equilibrium controls the degree of densification (German et al., 2009). In nuclear engineering the release of fission gases from polycrystalline uranium dioxide is controlled by the formation of texturally equilibrated grain edge porosity (Tucker, 1979).

Geological applications of textural equilibrium include partial melting and melt

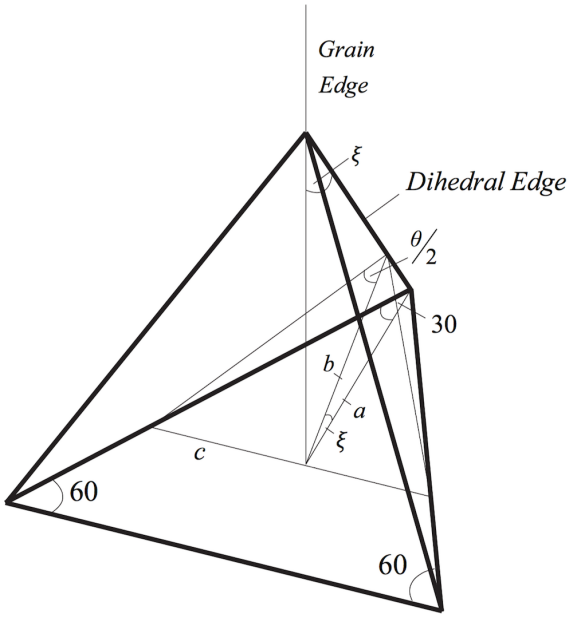


Figure 1.5: Textural equilibrium at a vertex with similar grains and isotropic interfacial energies. Tip angle can be represented as a function of the dihedral angle. Each edge in this figure represents a crystal-crystal edge, such as shown in Fig. 1.4*a*

segregation which control the chemical differentiation of terrestrial planets. Observations suggest both very low porosities yet efficient extraction of partial melt (Spiegelman and Elliott, 1993; von Bargen and Waff, 1986; Cheadle, 1989). This can be explained by the low θ of melt-rock systems that allows the percolation of texturally equilibrated pore networks at very low porosities (Fig. 1.4*d*).

Polycrystalline ice also develops a texturally equilibrated water network near its melting point (Fig. 1.4*e*) (Nye, 1989; Mader, 1992), and even at sub-zero temperatures in polar ice-sheets impurities depress the melting point and give rise to melt networks (Wolff and Paren, 1984; Dash et al., 1995). These water networks provide fast diffusion path that may displace climate signals recorded in ice-cores (Rempel et al., 2001) and provide a habitat for microbial communities (Price, 2000; Mader et al., 2006).

The fast reaction kinetics of salt dissolution and precipitation also allow the for-

mation of texturally equilibrated pore networks in rock salt (Fig. 1.4*f*). Subsurface salt deposits are generally considered impermeable and hydrocarbon accumulations are often associated with them (Downey, 1984). The extremely low permeabilities of salt are consequence of the large salt-brine θ that prevents percolation at low porosities. At the higher pressures and temperatures, however, the salt-brine θ decreases below 60° and allows the formation of a percolating pore network (Lewis and Holness, 1996; Holness and Lewis, 1997). This offers an elegant explanation for field observations of oil impregnated salt (Schoenherr et al., 2007) and implies that highly radioactive waste stored in rock salt may come into contact with groundwater if the decay heat increases the temperature sufficiently to allow brine percolation (Lewis and Holness, 1996).

1.2 Motivation and Problem Description

Evaporation of salt-rich waters can result in the deposition of thick salt layers. Over time, denser sediments cover the layer of salt and bury it under large overburden. Density difference between the salt and ambient sediments causes vertical flow of salt, and the formation of salt pillows or domes. The hydraulic properties of the evaporites change during the burial process. The permeability of uncompacted salt is in the order of $10^{-7}m^2$, but after 500m burial, it can be as low as $10^{-21}m^2$ (Ingebritsen et al., 2006) due to the plastic deformation of the salt and its tendency to recrystallize at higher pressure and temperature. Thus the rock salt found in salt domes is almost impermeable and this low permeability allows the rock salt to seal large hydrocarbon columns and fluid pressure cells. The presence of such seals is an essential element of a petroleum system in sedimentary basins. For example, two-thirds of the deep water Gulf of Mexico is covered with salt, and a lot of hydrocarbon reservoirs are associated to them (Fig. 1.6).

There is, however, considerable and diverse evidence that salt may act as a fluid

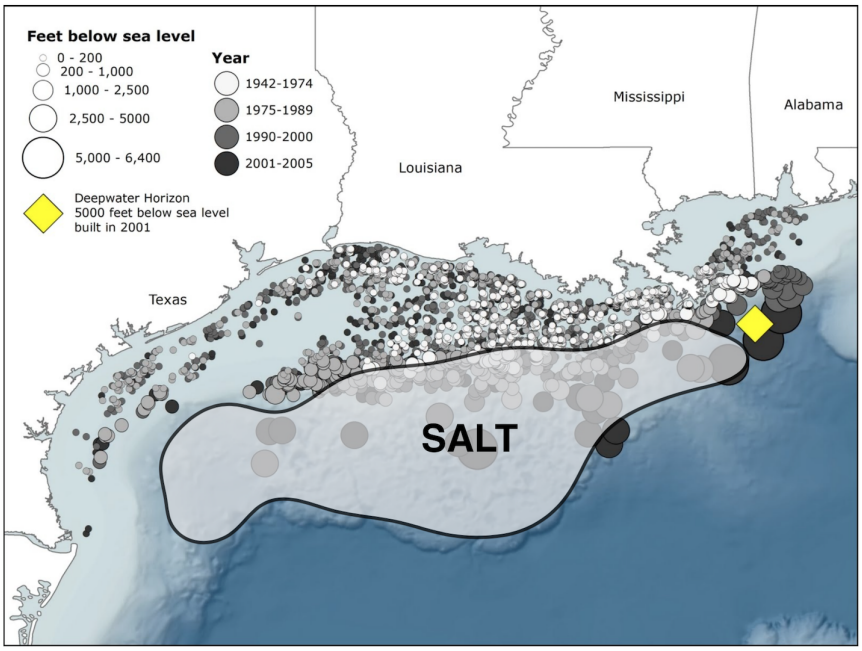


Figure 1.6: Two-thirds of deep water Gulf of Mexico is covered with salt.

conduit, in particular at greater depth in sedimentary basins. Two processes are known to increase permeability in rock salts. The first is dilation and micro-cracking during dynamic recrystallization. The second process is the formation of topologically connected pores and channels on grain edges due to changes in interfacial tension between brine and rock salt with increasing p-T. The change in the ratio of salt-salt and salt-brine interfacial energies changes the water-halite dihedral angle (θ). When $\theta \leq 60^{\circ}$, textural equilibrium enforces the brine to be imbibed in salt and wet the crystal edges, and create an interconnected thermodynamically stable network of channels at grain-boundary triple junctions. The possibility that salt deposits act as fluid conduits at greater depths in sedimentary basins has a significant effect on fluid circulation and hydrocarbon migration in sedimentary basins. This also suggests that radioactive waste stored in rock salt may become connected to the ambient hydrological system if temperature increases sufficiently. In this study we focus on percolation and fluid flow in ductile rocks, including rock salt, and perform experiments

and field analysis to constrain sealing capacity of the rock salt.

In addition to salt-brine systems, textural equilibrium determines the solid-liquid topology in many naturally occurring materials such as partially molten rocks, ice-water and alloy-melt systems (Fig. 1.4*c*-1.4*e*). In these materials the connectivity and geometry of pore network is also controlled by the ratio of surface energies of the mineral grains and the pore fluid. One of important application and implication of such phenomenon is the melt segregation which controls the chemical differentiation of terrestrial planets. Rapid core formation in early planetary bodies is required by geochemical data from extinct radionuclides and the most obvious mechanism for metal-silicate differentiation is the segregation of dense core forming melts by porous flow. However, experimental observations show that the texturally equilibrated metallic melt resides in isolated pockets ($\theta \approx 90^\circ$) that prevent percolation towards the center. Here we propose the novel idea of the hystresis in pore network connectivity that allows the melt to remain interconnected as drainage reduces the porosity below the percolation threshold and only 1-2% is trapped.

1.3 Research Objectives

All the objectives listed below can address several open research questions regarding percolation, fluid flow, properties and dynamic evolution of the ductile rocks.

1.3.1 A Level Set Method for Materials with Texturally Equilibrated Pores

In this step, a mathematical model will be developed to model the textural equilibrium in polycrystalline materials. Current analytical/computational methods are limited by symmetry and isotropy assumptions and are only valid in two-dimensions otherwise they do not link-up to a three-dimensional network (Fig. 1.7). The objective in this step is to solve the textural equilibrium problem in realistic crystal textures and investigate fluid distribution

and percolation path in materials with texturally equilibrated pores. We formulate a theoretical model for interfacial topology, which minimizes the solid-liquid interfacial surface area, which eventually yields a three-dimensional equilibrium solid-liquid interfacial surface. To our knowledge, we are the first to compute the two-dimensional fluid distribution in textural equilibrium with any crystal lattice.

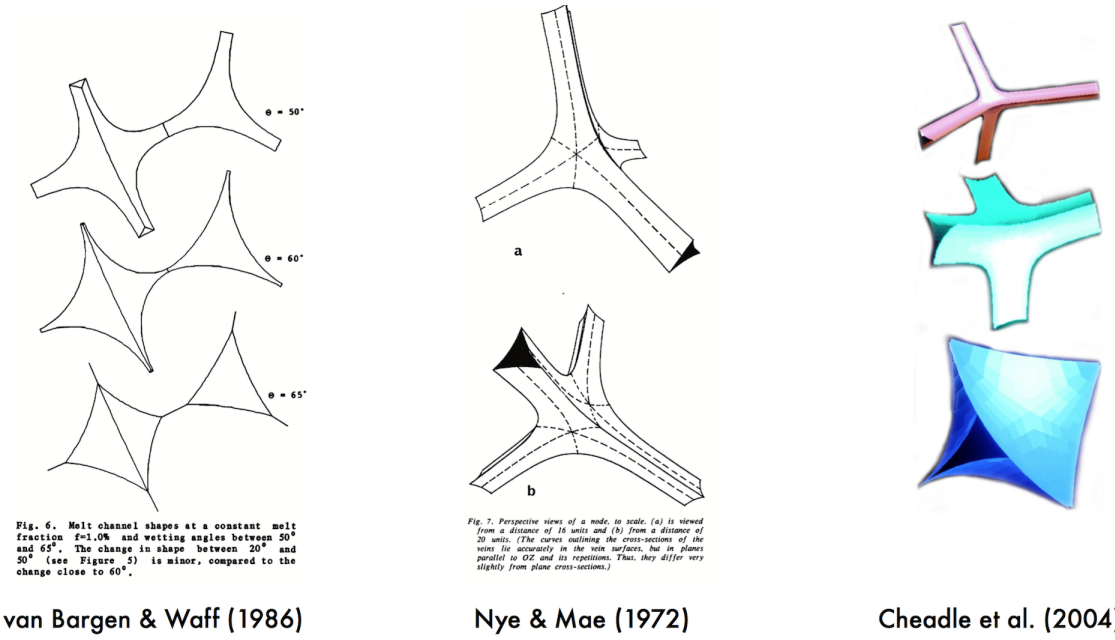


Figure 1.7: Current models of textural equilibrium only consider a piece of the pore network and extrapolate results using an assumption of symmetry. In reality, this assumption is not valid and the symmetric pieces do not link to a three-dimensional network. Images from von Bargen and Waff (1986); Nye (1989) and Cheadle et al. (2004).

1.3.2 Percolation and Physical Properties of Ductile Rocks

In texturally equilibrated material, θ has first order effects on matrix physical properties via its underlying control on fluid topology. The pore configuration determines a wide range of physical properties such as percolation, permeability, capillary pressure curve, formation factor, cementation exponent (m), elastic moduli and acoustic velocities (V_p and V_s). In this work, we will quantify the effect of θ, porosity (ϕ), anisotropy and irregularity in

grain geometries on important physical properties of materials with texturally equilibrated pores using direct simulation of fluid flow and electrical conductivity. To study the percolation, we run a series of simulations to obtain percolation threshold at different values for porosity and dihedral angle. For computing permeability, we simulate the single-phase in the computationally obtained pore networks using Lattice Boltzmann method (LBM). The electrical conductivity problem is also solved with the idea that a variational principle exists for the linear electrical conductivity problem.

1.3.3 Pore-Scale Experimental Study of Rock Salt

Understanding the process in which salt may lose its sealing capacity requires additional knowledge about salt pore structure in different pressures, temperatures and porosities. The aim of this part of project is to reconstruct the three-dimensional pore space of salt at different porosities and P-T conditions using X-ray micro-tomography and available digital image processing techniques. The obtained pore geometries can also be used to determine physical properties listed above. To our knowledge, this work is the first to quantify textures and extract the main characteristic of salt-brine pore space using X-ray micro-tomography and porous media image processing techniques.

1.3.4 Filed Study of Fluid Percolation in Ductile Rock Salt in Gulf of Mexico

Until recent years, rock salt has been considered to be impermeable as it seems to contains and keep gas inclusions for long time. Increasing energy demand and necessity of producing hydrocarbon reservoir enclosed or touched by salt deposit have brought attention to research and study the porosity and permeability of natural rock salt. This commercial interest in the large hydrocarbon accumulations below extensive bodies of allochthonous salt in the deep water Gulf of Mexico provides an opportunity to test the static pore-scale theory in slowly moving natural rock salt. In order to do so, we analyze 48 wells penetrating

salt deposits in the Gulf of Mexico. The observed hydrocarbon distributions in rock salt require that percolation occurred at porosities considerably below the static threshold. The goal is to constrain the brine and hydrocarbon connectivity in rock salt.

1.3.5 Dynamic Compaction in Partially Molten Ductile Rocks

In order to conclude the study, we connect the pore-scale configurations and properties to a larger scale and study the dynamics of texturally equilibrated materials. In this step, we aim to provide a consistent mathematical model for governing physical processes that determines flow of molten iron in a deforming silicate matrix. The foundations of model are conservation of mass, momentum and energy, considering both solid and fluid as incompressible material. The proposed model couples the computations of the micro-structure that controls the permeability with evolution of the gravity field, macro-scale pressure, temperature and melt generation with radiogenic heat decay. We use the hypothesis of the hysteresis in pore network topology to address the planetary core formation with porous flow.

1.4 Dissertation Outline

The organization of this dissertation is straightforward. The chapters exactly follow the research objectives that are outlined in Section 1.3. In Chapter 2, we discuss the details of developed level set method for materials with texturally equilibrated pores. In Chapter 3, properties of porous ductile rocks are throughly investigated. Chapter 4 covers the pore-scale static experiments on salt-brine system to establish criteria for sealing capacity of rock salt. Chapter 5 addresses the need for an extensive field study on fluid entry and percolation in dynamically deforming rock salt by studying data from 48 subsalt wells in Gulf of Mexico. In Chapter 6 we propose a hypothesis for planetary core formation by

porous flow with hysteresis in pore network connectivity.

Every chapter starts with a short background information, problem description and literature review. This is followed by the methodology to conduct the research and achieve the research objective. The last section in each chapter focuses on the results, discussion and analysis of the results. The last chapter, Chapter 7, contains a comprehensive conclusion of this dissertation followed by recommendations for future research in this field. Three appendices present the developed code for the level set method for materials with texturally equilibrated pores, dynamic compaction model and automatization of dihedral angel measurement from X-ray microtomography images. Two other appendices cover two-dimensional semi-analytical solutions for materials with texturally equilibrated pores as well as computed permeabilities from Lattice Boltzmann simulations.

Chapter 2

A Level Set Method for Ductile Materials with Texturally Equilibrated Pores

2.1 Background and Literature Review

Textural equilibrium determines the solid-liquid topology in many natural materials, such as partially molten rocks (von Bargen and Waff, 1986), ice-water systems (Rempel et al., 2001), salt-brine systems (Lewis and Holness, 1996) and alloys (Smith, 1948), see (Ghanbarzadeh et al., 2014) for a recent review. Textural equilibrium is the state of thermodynamic equilibrium where the interfacial area has evolved to minimize the solid-liquid surface energy density, γ_{sl} (Holness, 2010), and hence to constant mean curvature, κ, if the pressure is hysdrostatic and the grains are isotropic. In these materials the topology and geometry of the pore network is controlled by the dihedral angle, θ, which is a function of the surface energies of the mineral grains and the pore fluid (Ghanbarzadeh et al., 2014). The basic theory of textural equilibrium in two-phase materials have been introduced by (Smith, 1948, 1964) in the context of partially molten alloys. Texturally equilibrated pores are common in porous materials with fast solid-liquid kinetics or in cases where long equilibration time scales are available, therefore they are common in geological systems. In most cases, solid-solid interfaces can be considered stationary on the timescale required to reach textural equilibrium of the pore network (von Bargen and Waff, 1986; Ghanbarzadeh et al., 2014), so that the solid-solid surface energy density, γ_{ss}, is not minimized. Fig. 2.1 illustrates how these pre-existing grain-grain boundaries impose a lattice on the pore space and introduces contact lines along which solid-liquid and solid-solid interfaces meet at

sharp angles (Fig. 2.1). In a two phase material with isotropic surface energies, mechanical equilibrium at the contact line requires that

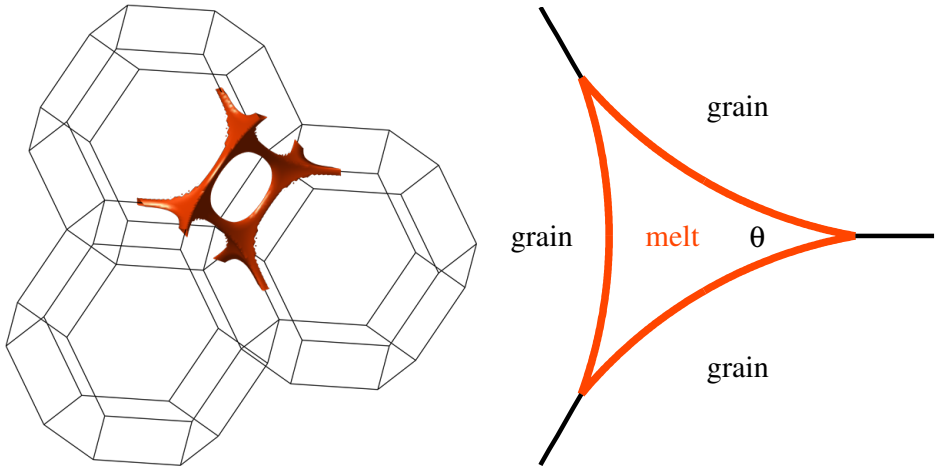


Figure 2.1: (a) Wireframe of three truncated octahedron grains with a texturally equilibrated grain edge porosity of 1%. (b) Cross section of a grain-edge channel illustrating the definition of dihedral angle, θ. Images from (Ghanbarzadeh et al., 2014) used with permission from the American Physical Society.

$$\gamma_{ss} = 2\gamma_{sl} \cos(\theta/2) \tag{2.1}$$

where θ is the dihedral angle, γ_{ss} and γ_{sl} are solid-solid and solid-liquid surface energy density, respectively (Holness, 2010).

The physical principles of textural equilibrium are similar to standard wetting problems (de Gennes, 1985). The most important difference between textural equilibrium and standard wetting problems is the role of the solid. In typical wetting problems the solid geometry is given and does not evolve. In the problem considered here the solid has a dual role. The geometry of pre-existing solid-solid grain boundaries does not evolve and provide a static lattice for fluid percolation. The solid-liquid grain boundaries, however, do evolve and can eliminate the solid-solid boundaries as the solid-solid-liquid triple lines migrate.

A characteristic of texturally equilibrated porous media is that the pore network percolates at any porosity for $\theta \leq 60^\circ$ while a percolation threshold exists for $\theta > 60^\circ$ (Bulau et al., 1979). This property is specially important in comparison with the percolation theories in granular porous media where a porosity of 3% is required for connectivity of the pore space (van der Marck, 1999). This ability of texturally equilibrated pore networks to percolate at very low porosities provides an elegant explanation for several geological observations (Zhu et al., 2011; Schoenherr et al., 2007; Rempel et al., 2001). For example, the small dihedral angle between basaltic melt and olivine explains the near instantaneous extraction of partial melts beneath mid-ocean ridges that was inferred from a number of indirect observations (Bulau et al., 1979; von Bargen and Waff, 1986; Sobolev and Shimizu, 1993; Lundstrom et al., 1995; Spiegelman and Elliott, 1993). The decrease of the dihedral angle between rock salt and brine with increasing pressure and temperature (Lewis and Holness, 1996; Holness and Lewis, 1997) can explain how rock salt that is generally impermeable at shallow depth (Downey, 1984) can become become permeable and stained by oil at greater depth (Schoenherr et al., 2007).

The first models which calculated the three-dimensional shape of pore networks in textural equilibrium were developed by (Beere, 1975) and (von Bargen and Waff, 1986). The former was based on interfacial surface energy minimization and the latter was developed based on the idea that at equilibrium, chemical potential of components in different phases is constant. Both models eventually reach to same *essential condition* for texturally equilibrated pores

$$\kappa = \text{const} \tag{2.2}$$

where κ is mean curvature of the solid-liquid interface for a two-phase system under hydrostatic pressure and with isotropic surface energies. Later, the model developed in

(von Bargen and Waff, 1986) was reproduced to study the seismic wave velocities of partially molten rocks (Takei, 2002) and their electrical properties (Pervukhina and Kuwahara, 2008). Recently, (Wimert and Hier-Majumder, 2012) developed a three-dimensional micro-geodynamic model to solve for grain-melt geometry in an isotropic unit cell comprised of rhombic dodecahedral grains balancing pressure, surface tension, and viscous deformation forces.

A closed surface which minimizes the area subject to a fixed enclosed volume must have constant mean curvature, κ (Oprea, 2000). Therefore, in textural equilibrium, solid-liquid interface is a minimal surface subject to dihedral angle condition at boundaries. Considering a solid-liquid interface given by $z = f(x, y)$, mean curvature can be defined as

$$\kappa = \frac{(1 + f_y^2) f_{xx} + (1 + f_x^2) f_{yy} - 2 f_x f_y f_{xy}}{2(1 + f_x^2 + f_y^2)^{\frac{3}{2}}} = \text{const} \tag{2.3}$$

which should be constant at textural equilibrium problem and simultaneously, solid-liquid interfaces need to satisfy dihedral angle condition (Eq. 2.1) at boundaries. In order to close the problem with additional unknown of $\kappa = \text{const}$, Eq. 2.3 should be solved with volume constraint

$$V_f = \int_\Omega f(x, y)\, dx\, dy. \tag{2.4}$$

The system of equations presented in Eqs. 2.3 and 2.4 subject to Neumann boundary conditions is a free boundary problem in Cartesian coordinate system. This equation should be solved coupled with systems representing other disconnected pieces of solid-liquid interfacial surfaces in medium. This results in very complex system which requires topology tracking and changing computational domains. Also free boundary nature of the problem

makes it more complicated to be solved in systems with complex grain configurations. Current three-dimensional computational models (Beere, 1975; von Bargen and Waff, 1986; Nye, 1989; Cheadle, 1989) are therefore limited by symmetry assumptions in grain geometry resulting in unrealistic pore shapes which do not link up to a network in three-dimensions.

Despite extensive studies and the common ocurrence of materials with texturally equilibrated pores, first order questions have not been resolved. Of particular importance, is the presence of percolation path and wetted grain faces and their effects on the physical properties of these porous media (Hirth and Kohlstedt, 1995; Takei, 2002; Endres et al., 2009) in both isotropic and anisotropic crystal lattices (Ghanbarzadeh et al., 2014). To address this problems we propose a novel level set model to determine an implicit representation of liquid distribution in textural equilibrium with realistic and complicated polycrystalline solids. This model is verified comparing the two-dimensional results with solutions to minimal surface problem. Two-dimensional representation of Eq. 2.3 in cylindrical coordinate system, $r = f(\Theta)$, is given by

$$\kappa = \frac{r^2 + 2r'^2 - rr''}{\left(r^2 + r'^2\right)^{\frac{3}{2}}} = \text{const} \tag{2.5}$$

which removes the free boundary nature of problem. Here r' and r'' are first and second derivatives with respect to Θ.

The level set method (Osher and Sethian, 1988; Osher and Fedkiw, 2002) is a numerical technique that tracks a surface by representing it as a zero level set of a hypersurface, φ. This allows propagating the interface by solving an initial value problem for the evolution of φ governed by a Hamilton-Jacobi equation. In this setting, curvatures and normals may be evaluated easily and topological changes occur in a natural manner (Sethian, 1999) and thus it is a good choice for modeling the behavior of complex surfaces. As the

method uses a fixed Cartesian grid, extension to any number of dimensions is straightforward. The level set function, φ, separates the interior and exterior region of an interface by its sign and evolves by general equation of the form

$$\varphi_t + \mathbf{v} \cdot \nabla\varphi = 0, \tag{2.6}$$

where $\mathbf{v}$ is the interface velocity and includes the physics of problem discussed in Section 2.2. While a level set function undergoes the evolution with appropriate velocities, it is important to keep the magnitude of its gradient ($|\nabla\varphi|$) bounded to ensure the numerical stability, convergence and accuracy. To do so, the level set function is replaced by a signed distance function in a process called reinitialization that does not move the interface position. To reinitialize φ, we solve

$$\varphi_t + \text{sign}(\varphi)\ [|\nabla\varphi| - 1] = 0 \tag{2.7}$$

every few time step (Peng et al., 1999). This process diffuses the interface in sub-resolution scales and corrections to the method exist in (Smereka, 2003). The gradient magnitude of the steady-state answer to Eq. 2.7 is one in computation domain.

The incorporation of solid-liquid contact angle is an important application in multiphase flow simulations (Spelt, 2005; Liu et al., 2005; Li et al., 2010). Most of these studies consider a flow of single droplet on flat surfaces and the contact angle condition is applied by local reconstruction of level set function during reinitialization. (Jettestuen et al., 2013) extended the level set formulation of the critical displacements of fluids during drainage and imbibition in porous media (Prodanovic and Bryant, 2006) to include the solid-liquid contact angle. They combined the fluid equation of motion in the pore space (Prodanovic

and Bryant, 2006) and fluid contact angle term at the solid phase (Lee et al., 2010) into a single evolution equation over the entire computational domain.

In this work we follow a similar procedure to include the dihedral angle on solid-liquid contact lines. But here we need two level set functions for each grain, i, one representing solid grain, ψ_i, and one representing liquid phase around the same grain, φ_i. Surface diffusion happens in many film growth problems and the steady-state solution to interface motion driven by surface diffusion is a constant mean curvature surface (Chopp and Sethian, 1999; Smereka, 2003). Therefore, in order to obtain constant mean curvature surface, the solid-liquid interface evolves with surface Laplacian of curvature as interface speed. This evolution is volume conservative so the desired volume of liquid (porosity) is achieved by adding a normal velocity term to the equation that inflates or deflates φ. The dihedral angle condition is added to the formulation in form of normal and convective velocity terms, which adjust the interface on solid-liquid contact lines. The solid-liquid contact line forms in places where two adjacent solid grains and their associated liquid level sets meet.

The resulting system of nonlinear PDEs is solved explicit in time with an implicit representation of grains as initial guess. The simulation performance is optimized by computational domain decomposition for each grain (Ω_i) and evaluating coupling terms on the original computational grid (Ω). The domain decomposition is used to limit the computational domain size of each grain to the region outside and close to the interface. The method is tested by demonstrating a narrow distribution of mean curvature on solid-liquid interface and comparing the distribution of steady state dihedral angles with the prescribed angle. Unlike previous methods (von Bargen and Waff, 1986; Beere, 1975) the developed method is not limited by grain geometry constraints. The geometric flexibility of the method is demonstrated by presenting pore geometry in isotropic and anisotropic textures with equal

and unequal grains as simulation examples.

2.2 Level Set Formulation

Assuming hydrostatic pressure and isotropic surface energy densities, the solid-liquid interface must satisfy both conditions stated in Eq. 2.1 and 2.2 simultaneously (von Bargen and Waff, 1986; Bulau et al., 1979). Surfaces of constant mean curvature are given by the steady-state solutions to interface motion driven by surface diffusion (Chopp and Sethian, 1999; Smereka, 2003). The velocity of an interface moving with surface diffusion is the surface Laplacian of curvature, $\Delta_s \kappa$. Then the evolution equation takes the form

$$\varphi_t + \Delta_s \kappa \left| \nabla \varphi \right| = 0 \tag{2.8}$$

in which φ is a level set function and the interface is given by its zero level-set. The interface given by steady-state answer of Eq. 2.8 is a constant mean curvature which preserves the volume (Smereka, 2003). The surface Laplacian of curvature is given by

$$\Delta_s \kappa = \nabla_s \cdot \nabla_s \kappa \tag{2.9}$$

where

$$\nabla_s = \left(\nabla - \vec{n} \left(\vec{n} \cdot \nabla \right) \right), \tag{2.10}$$

and $\vec{n}$ is the outward normal and κ is the mean curvature

$$\kappa = \nabla \cdot \vec{n} = \nabla \cdot \left(\frac{\nabla \varphi}{\left| \nabla \varphi \right|} \right). \tag{2.11}$$

Function φ separates the domain (Ω) to interior (Ω^-) and exterior (Ω^+) regions. Mathematically, we represent this by smoothed Heaviside function

$$H(\varphi) = \begin{cases} 0 & \varphi < -\epsilon \\ \frac{1}{2} + \frac{\varphi}{2\epsilon} + \frac{1}{2\pi}\sin\left(\frac{\pi\varphi}{\epsilon}\right) & -\epsilon \leq \varphi \leq \epsilon \\ 1 & \varphi > \epsilon \end{cases} \tag{2.12}$$

where $1.5\Delta x \leq \epsilon \leq 3\Delta x$ is a tuning parameter. The interface can be identified by $\delta(\varphi)$ function which is the derivative of $H(\varphi)$.

In the method presented here, each grain, i, is represented by two level sets, one representing the solid-solid interfaces of the grain, ψ_i, and another representing the solid-liquid interfaces of the grain, φ_i. The level-set functions for the soild-solid grain boundaries does not evolve in time and provides a reference frame for the solid-liquid contact lines. We should note that the physical solid-solid interface evolves with time, changes in the porosity and the dihedral angle. To compute the solid-liquid interface with constant mean curvature, the φ_i level set evolves by surface diffusion. Due to computational difficulties and instability from unbounded values of $\Delta_s\kappa$ (Chopp and Sethian, 1999), we only consider surface diffusion close to interface, in the region where $\delta(\varphi)$ is nonzero. Then, in a polycrystalline material with N grains, the liquid level set of the i-th grain undergoes evolution with

$$(\varphi_i)_t + \left[\hat{\delta}(\varphi_i)H(-\psi_i)\right]\ \Delta_s\kappa_i\left|\nabla\varphi_i\right| = 0 \tag{2.13}$$

The term $\hat{\delta}(\varphi_i) = \Delta x\delta(\varphi_i)$, where Δx is the grid size, avoids large values of the delta function ($\delta \propto 1/\Delta x$). The term $H(-\psi_i)$ ensures that the diffusive motion is occurring inside and close to interface of each individual grain.

The other constraints of the problem, the desired porosity and the dihedral angle on solid-liquid contact lines, must be added to formulation as additional terms to Eq. 2.13.

In each time step, the global implicit function representing the solid-liquid interface as its zero level set, Φ, on the original computational domain (Ω) is computed as union of φ_i's by

$$\Phi = \min\left(\varphi_1, \varphi_2, \ldots, \varphi_N\right), \tag{2.14}$$

because φ_i is positive outside the i-th grain. The liquid volume in each time step is then calculated using

$$V_f = \int_\Omega H(-\Phi)\, dV \tag{2.15}$$

which results in O(Δx) volume accuracy regardless of the integration method used (Osher and Fedkiw, 2002). The relative error between current and desired volume, $\hat{v}$, can then easily be evaluated. In order to adjust the porosity, a normal motion is added to the evolution equation of the level set representing the solid-liquid interfaces, φ_i. This motion causes the liquid phase to expand or shrink by moving the level set function in a normal direction. The velocity of normal motion is proportional to $\hat{v}$ and is scaled by factor of $\Delta x^{-2} e^{\hat{v}}$, by trial and error, in order to be comparable to other terms in Eq. 2.13. This velocity term vanishes as porosity of medium converges to desired value. Eq. 2.13 then can be rewritten as follow

$$(\varphi_i)_t + \left[\hat{\delta}(\varphi_i) H(-\psi_i)\right] \Delta_s \kappa_i \left|\nabla \varphi_i\right| + \left[\frac{\hat{v}}{\Delta x^2} e^{\hat{v}}\right] \left|\nabla \varphi_i\right| = 0 \tag{2.16}$$

The other constraint of the problem which must be satisfied is the mechanical equilibrium condition along the liquid-solid-solid contact lines. The liquid region is defined by the intersection of the outside of all liquid level sets associated to each solid grain (φ), mathematically represented by Eq. 2.14. As can be seen in Fig. 2.2, two liquid level sets (φ_i and φ_j) intersect with dihedral angle θ on corresponding grain faces and the normals

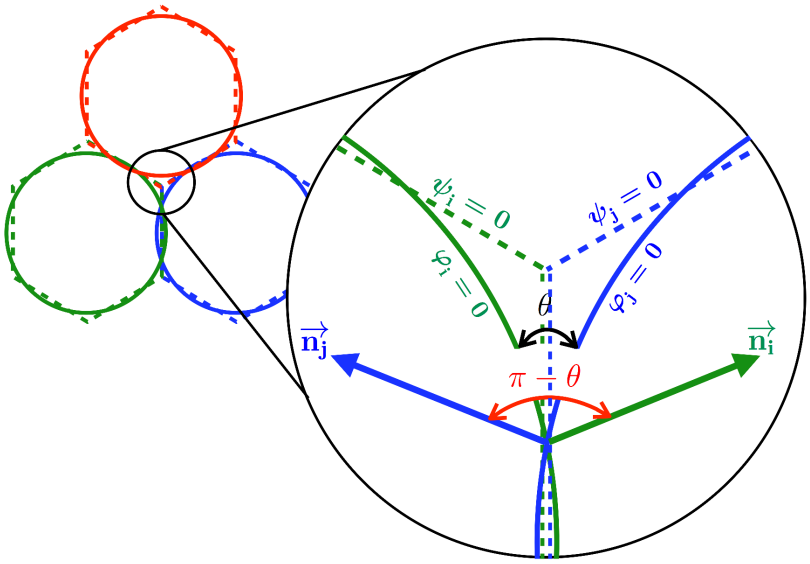


Figure 2.2: Two-Dimensional description of the dihedral angle, dihedral edge, liquid and solid level sets. While the two corresponding liquid level sets, φ_i and φ_j, meet with the angle θ, their normals make the angle $\pi - \theta$ with each other.

to interface make the angle $\pi - \theta$ with each other (Jettestuen et al., 2013). Satisfying the dihedral angle condition

$$[\vec{n}_i \cdot \vec{n}_j - \cos(\pi - \theta)] \left[\hat{\delta}(\varphi_i)\,\hat{\delta}(\varphi_j)\right] \left[\hat{\delta}(\psi_i)\,\hat{\delta}(\psi_j)\right] = 0. \tag{2.17}$$

The term $\hat{\delta}(\varphi_i)\,\hat{\delta}(\varphi_j)$ ensures that the dihedral angle condition is only active where two level set meet and term $\hat{\delta}(\psi_i)\,\hat{\delta}(\psi_j)$ keeps the intersection point on grain-grain contact between grains i and j. Substituting the definition of the interface normal, $\vec{n} = \nabla\varphi / |\nabla\varphi|$, Eq. 2.17 can be expressed as

$$\begin{aligned}\left[\nabla\varphi_i \cdot \nabla\varphi_j - |\nabla\varphi_i||\nabla\varphi_j| \cos(\pi - \theta)\right] \\ \left[\hat{\delta}(\varphi_i)\,\hat{\delta}(\varphi_j)\right] \left[\hat{\delta}(\psi_i)\,\hat{\delta}(\psi_j)\right] [S(\varphi_i)\,S(\varphi_j)] = 0\end{aligned} \tag{2.18}$$

in which $S(\varphi)$ is the sign function suggested by (Peng et al., 1999) and $S(\varphi_i)S(\varphi_j)$ term enforces the dihedral angle to spread away from adjacent level sets and secures the numeri-

cal stability (Jettestuen et al., 2013). Eq. 2.18 contains two velocity terms for the motion of i-th level set, one normal and one convective, that move the interface at liquid-solid-solid edges to satisfy dihedral angle condition. The final equation for evolution of the i-th level set is obtained by combining Eq. 2.16 and 2.18, so that

$$\begin{aligned}
(\varphi_i)_t &+ \left[\hat{\delta}(\varphi_i)H(-\psi_i)\right]\ \Delta_s\kappa_i\,|\nabla\varphi_i| \\
&+ \left[\frac{\hat{v}}{\Delta x^2}e^{\hat{v}} - \cos(\pi-\theta)\,\hat{\delta}(\varphi_i)\,\hat{\delta}(\psi_i)\,S(\varphi_i)\right. \\
&\quad \left.\sum_{j=1:n}^{j\neq i}\left\{\hat{\delta}(\varphi_j)\,\hat{\delta}(\psi_j)\,S(\varphi_j)\,|\nabla\varphi_j|\right\}\right]|\nabla\varphi_i| \\
&+ \left[\hat{\delta}(\varphi_i)\,\hat{\delta}(\psi_i)\,S(\varphi_i)\right. \\
&\quad \left.\sum_{j=1:n}^{j\neq i}\left\{\hat{\delta}(\varphi_j)\,\hat{\delta}(\psi_j)\,S(\varphi_j)\nabla\varphi_j\right\}\right]\cdot\nabla\varphi_i = 0 \\
&\qquad\qquad\qquad\qquad\qquad\qquad \text{for} \quad i = 1\ldots N
\end{aligned} \tag{2.19}$$

where N is the number of grains and Eq. 2.19 needs to be solved for each grain, thus it represents a system of N coupled non-linear PDEs. Textural equilibrium is achieved once the steady-state solution to Eq. 2.19 is obtained. In steady-state, the constant mean curvature of the solid-liquid interface will result in zero values for $\Delta_s\kappa\,\hat{\delta}(\phi_i)$, the term $\hat{v}$ will be zero and satisfaction of Eq. 2.17 returns normal and convective velocities which cancel out each other. In order to keep the level set a signed distance function and preserve numerical stability, one needs to solve Eq. 2.7 every few time steps. As the intersection of level sets creates the actual liquid distribution, final level set representing texturally equilibrated pore network with dihedral angle, θ, and porosity, ϕ, is then given by Eq. 2.14.

2.3 Implementation

The numerical discretization of Eq. 2.7 and Eq. 2.19 is implemented in MATLAB® utilizing the Level Set Toolbox developed by (Mitchell, 2008). The toolbox is modified to handle any number of level sets with structure data type, generate grid optimally and evaluate the required velocity terms. Surface Laplacian of curvature (κ_{ss}) is approximated with forth order (O(Δx^4)) central differences while convective and normal terms are discretized using fifth order (O(Δx^5)) Hamilton-Jacobi Weighted Essentially Non-Oscillatory (WENO) scheme (Osher and Fedkiw, 2002). Explicit time integration requires a very restrictive CFL condition, $\Delta t < C\Delta x^4$, due to the fourth order spatial derivatives in Eq. 2.19. The constant C is determined in a real-time manner. High-order discretization schemes enable us to use large grid sizes while reducing truncation error. In addition to decreasing memory usage, larger grid size results in larger time step required for stability ($\Delta t \propto \Delta x^4$).

All the boundary conditions are set to bi-linear extrapolation, and numerical discretization on boundaries is done by adding one stencil to computational domains, Ω and Ω_i, in each direction. Time variable in Eq. 2.19 does not have a physical dimension, θ and $\hat{v}$ are dimensionless, and delta, Heaviside and sign functions return non-dimensional values. As all the parameters in Eq. 2.19 are dimensionless, the grid size, Δx, does not have a physical meaning and it should be compared to grain size. Therefore, Δx in simulations is set in a way that a certain number of grid points (N_{grid}) span characteristic length of the grain (l_c). All level set functions are also signed distance functions and their values are relative to the grain size.

For each liquid level set function (φ_i), simulation starts with implicit representation of the corresponding grain (details in section 2.3.1) and solution is advanced explicitly in time. All the derivatives and coupled velocity terms are calculated from last time step, then all φ_i's undergo evolution with corresponding updates. The simulation continues until

the required conditions for textural equilibrium, Eqs. 2.1 and 2.2, are satisfied and final porosity is equal to the desired porosity. As the value of mean curvature in final answer is unknown, the error in the curvature is measured by standard deviation in values of curvature while the porosity and dihedral angle are the problem's input and the error in each time step is evaluated as an absolute error. We investigate the validity of these conditions once the steady state answer is reached. An array representing the mean curvature of solid-liquid interface corresponding to grain i is given by

$$\boldsymbol{\kappa}_i = \frac{\kappa_{\varphi_i}\hat{\delta}(\varphi_i)H(-\psi_i)\,|\nabla\varphi_i|}{\hat{\delta}(\varphi_i)H(-\psi_i)\,|\nabla\varphi_i|} \quad \text{where} \quad \hat{\delta}(\varphi_i)H(-\psi_i) \neq 0 \tag{2.20}$$

At the end of each iteration, uniformity of mean curvature can be examined by the value of standard deviation (σ) in distribution of $\boldsymbol{\kappa}_i$ for all grains satisfying

$$\sigma(\boldsymbol{\kappa}_i) \leq \epsilon_\kappa \qquad \text{for} \qquad i = 1 \ldots N \tag{2.21}$$

in which ϵ_κ is tolerance for variation in value of mean curvature in nodes close to solid-liquid interface. The second condition that needs to be satisfied is mechanical equilibrium at the dihedral edges given by Eq. 2.1. The error in dihedral angle on solid-liquid contact lines is calculated via

$$\mathbf{e}_{\theta_i} = \frac{\sum_{j=1:N}^{j\neq i} A_{ij}\hat{\delta}(\varphi_i)\hat{\delta}(\varphi_j)\;|\nabla\varphi_i|}{\sum_{j=1:N}^{j\neq i} \hat{\delta}(\varphi_i)\hat{\delta}(\varphi_j)\;|\nabla\varphi_i|} \;\text{ where }\; \sum_{j=1:N}^{j\neq i} \hat{\delta}(\varphi_i)\hat{\delta}(\varphi_j) \neq 0 \tag{2.22}$$

where

$$A_{ij} = \frac{\nabla\varphi_i}{|\nabla\varphi_i|} \cdot \frac{\nabla\varphi_j}{|\nabla\varphi_j|} - \cos(\pi - \theta) \tag{2.23}$$

where $\mathbf{e}_{\theta_i}$ is an array containing error in dihedral angle. We assume Eq. 2.1 is satisfied on solid-liquid contact lines once

$$\left|\left|\mathbf{e}_{\theta_i}\right|\right|_\infty \leq \epsilon_\theta \qquad \text{for} \qquad i = 1 \ldots N \tag{2.24}$$

is valid. Here $\left|\left| \ldots \right|\right|_\infty$ denotes infinity norm and ϵ_θ is the acceptable tolerance for error in dihedral angle.

2.3.1 Initialization

The numerical method requires level set representation for each grain (ψ_i) as input. Solid grains can have arbitrary shape and size, but grain-grain contact is necessary to establish solid-liquid contact lines to apply the dihedral angle constraint. Initial grain representation can optimally be constructed from pore-scale micro-tomographic images of synthesized or natural samples. To do this, a watershed algorithm should be applied to segmented images to separate the grains. Then signed distance function can be calculated to establish implicit representation of grains. Because the grain separation of the different solid grains is not a trivial process, in this work we have built level set representation of space-filling tessellations in three-dimensional space using signed distance of grid points from grain faces.

In order to create a level set representation for liquid phase corresponding to each grain (φ_i), we initialize φ_i with ψ_i, then evolve the liquid phase level set (φ) with curvature and normal motion terms to round the initial flat liquid interface, establish solid-liquid contact lines instead of contact plains, and to initiate porosity in domain. Therefore, the following evolution equation should be solved establish initial guess for φ_i

$$(\varphi_i)_t + ((v_n)_i - \kappa_i) \left|\nabla \varphi_i\right| = 0 \quad i = 1 \ldots N \tag{2.25}$$

which needs to be reinitialized (Eq. 2.7) every few time steps. In this work, the reinitialization processes are done every 5 time steps. The final answer also undergoes reinitialization to establish initial condition of Eq. 2.19 as a signed distance function. Normal velocity (v_n) and final time to stop evolution in Eq. 2.25 are trivial and κ is curvature given by Eq. 2.11. If a series of simulations with same grain network configuration is being done for a set of ϕ or θ values, this initialization from the grain geometry is only necessary for the first case. Computations can continue using results of previous simulations as initial condition.

2.3.2 Domain Decomposition

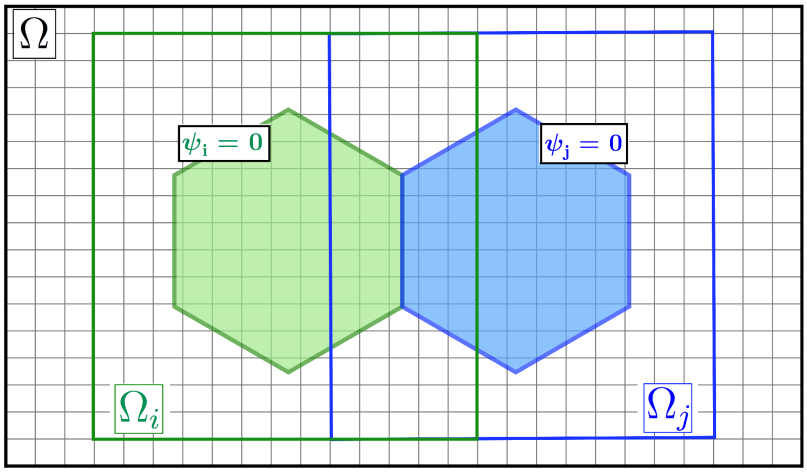


Figure 2.3: Two-dimensional schematic of domain decomposition. Computational domain of each grain, Ω_i, is a subset of main computational domain, Ω. Coupling terms between PDEs, which initiate from dihedral angle constraint, are calculated on Ω and then mapped on Ω_i.

The system of non-linear PDEs given by Eq. 2.19 is coupled though the normal and convective velocities arising from the dihedral angle constraints. The desired porosity is set by a normal velocity which is calculated on entire domain by Eq. 2.15 and the term which initiate from mean curvature constraints is evaluated independently for each grain. As we are interested in behavior of solid-liquid interface (zero level set), the solution domain can

be decomposed for each grain to just include the grain, solid-liquid interface and few grid points away from interface. The coupling between grains can be established on the original computational domain, Ω. To do so, implicit representation of each grains can be separated to inside and outside with logical operations then we can extend the computational domain of each grain to a number of grid points around the grain faces. In this study, we extended the domain from grain faces by $5\Delta x$ in all directions. Fig. 2.3 shows how initial representation of each grain on Ω, is reduced to smaller sub-domains, Ω_i. This method can be applied to any signed distance function representing the grain interface as its zero level set.

At each time step, the coupling velocity terms, originating from the dihedral angle constraint, are re-evaluated on Ω and then mapped yo Ω_i. Localization of computations and domain decomposition reduces the computational cost by orders of magnitude and enables us to solve large complex system presented in Eq. 2.19 very efficiently, even with a desktop or notebook.

2.3.3 Mesh Refinement for Visualization

Using large grid size with high-order discretization schemes to decrease the numerical error results in low quality visualization of the results. The visualization problem is more severe in three-dimensional simulations due to surface triangulation in the marching cube algorithm (Lorensen and Cline, 1987). In order to overcome this problem, a mesh refinement scheme is applied to steady-state answer of Eq. 2.19 and the final level set, representing the liquid-solid interface given by Eq. 2.14, is interpolated onto a refined grid using a spline gridded interpolation function for each liquid level set, (φ).

Fig 2.4 shows the effect of mesh refinement on visualization of the final results in a network of truncated octahedron grains for two dihedral angles θ = 30° and 90°. Fig. 2.4a and 2.4d represents the zero-isosurface of the final level set, Φ, with unrefined grid,

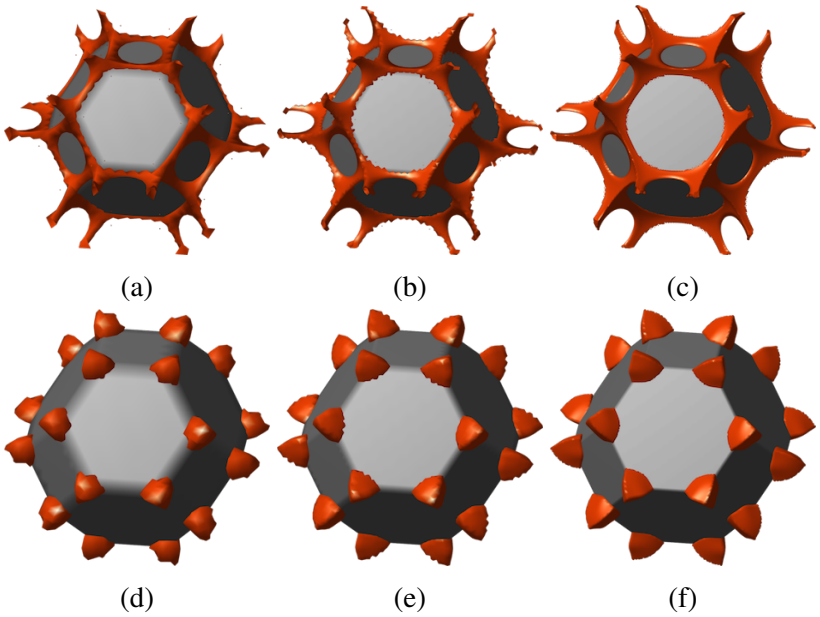


Figure 2.4: Comparison of visualization of final level set results with different mesh refinement levels for a case of truncated octahedron grain with ϕ = 2%. Original simulation is done with $N_{grid}/l_c = 20$. (a-c) $\theta = 30^\circ$ and $N_{grid}/l_c = 20$, 40 and 100 from left to right, (d-f) $\theta = 90^\circ$ and $N_{grid}/l_c = 20$, 40 and 100 from left to right

$N_{grid}/l_c = 20$. On the coarse computational grid the triangulation of the interface generated by the marching cube algorithm is not smooth. Fig. 2.4b and 2.4e present the zero level set of refined Φ with two-times refinement ($N_{grid}/l_c = 40$) and Fig. 2.4c and 2.4f are generated with five-times refinement ($N_{grid}/l_c = 100$). Mesh refinement can also improve the accuracy of volume calculation (Eq. 2.15). Table 2.1 summarizes the obtained porosity at the end of simulation for visualized cases in Fig. 2.4. In all simulations, relative error between current and target liquid volume ($\hat{v}$, used in Eq. 2.19) is calculated using refined Φ with $N_{grid}/l_c = 100$.

2.4 Model Verification

The accuracy of the proposed level-set method for textually equilibrated liquid-solid interface is demonstrated by comparison with analytic solutions in two-dimensions.

θ	30°			90°		
N_{grid}/l_c	20	40	100	20	40	100
$\phi(\%)$	2.45	2.12	1.99	2.31	2.25	2.14
error (%)	22.5	5.93	0.7	15.3	12.5	6.93

Table 2.1: Final porosity and error between target and obtained porosity with original grid size, $N_{grid}/l_c = 20$ and mesh refinement with $N_{grid}/l_c = 40$ and 100. Final results are plotted in Fig. 2.4. The target volume in all the cases is 2%.

In three dimensions no analytic solutions are available, instead we demonstrate that the numerical solution satisfies the constraints on curvature, dihedral angle, and porosity.

2.4.1 Two-Dimensional Simulation

In two dimensions the problems simplifies greatly, because all constant curvature surfaces are segments of either circles or lines. The results of proposed method can therefore be compared to the solution of Eq. 2.5. This equation is essentially a nonlinear ODE and is solved on one-dimensional grid with high-order Newton-Raphson method. Figs. 2.5b-2.5c show the equilibrium geometry of a single two-dimensional pore at a symmetric triple-junction at constant ϕ for increasing θ obtained from the proposed level set method and interfacial area minimization. Matching of results verifies the validity of constraints in steady-state answer of the developed numerical method. Fig. 2.5d compares the mean curvature of the solid-liquid interface obtained from both method. The level set method simulations are done for four dihedral angle values and gray shaded area shows the standard deviation in values of mean curvature near the interface. Comparison between the computed shapes in Figs. 2.5b-2.5c and the standard deviation of the curvature in Fig. 2.5d shows that the standard deviation does not reflect the accuracy of the interface. The distribution in curvatures is mostly due to sampling a curvature in a finite region around the interface.

Fig. 2.5 confirms that the developed level set model converges to semi-analytical

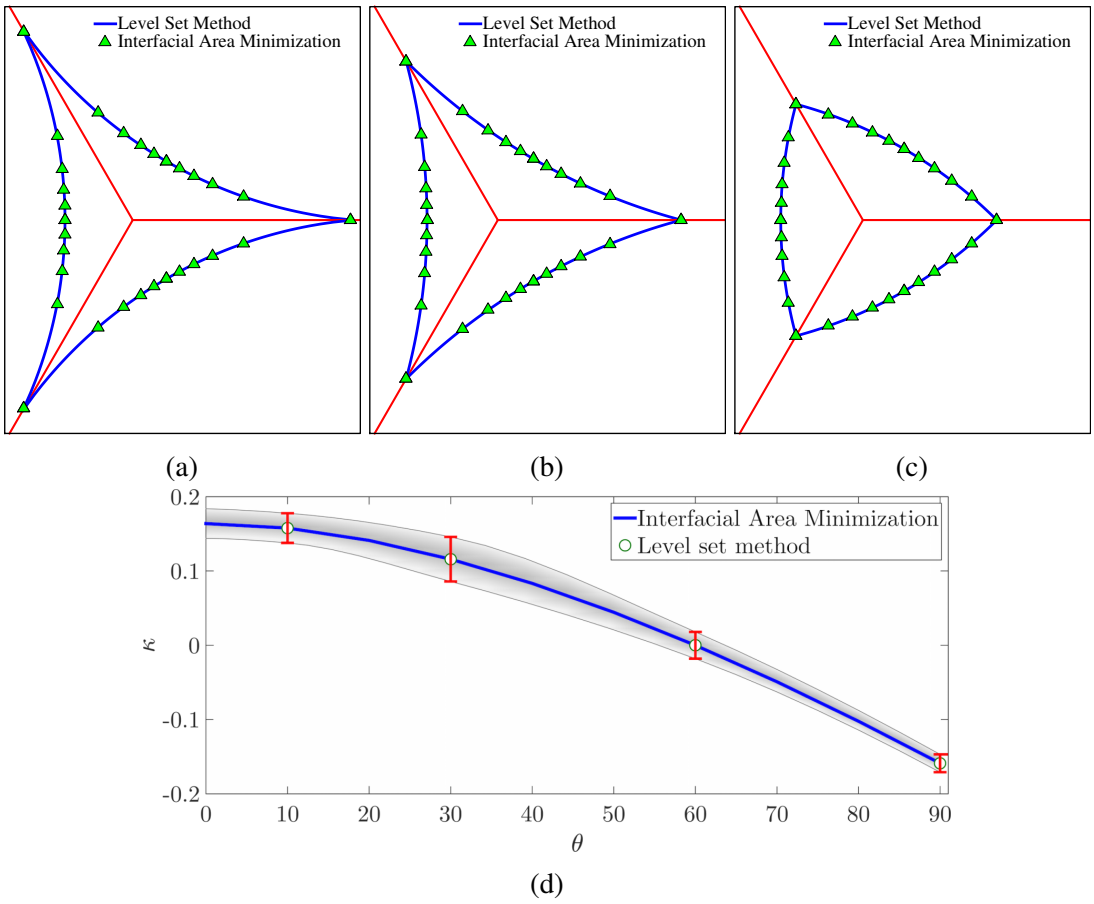


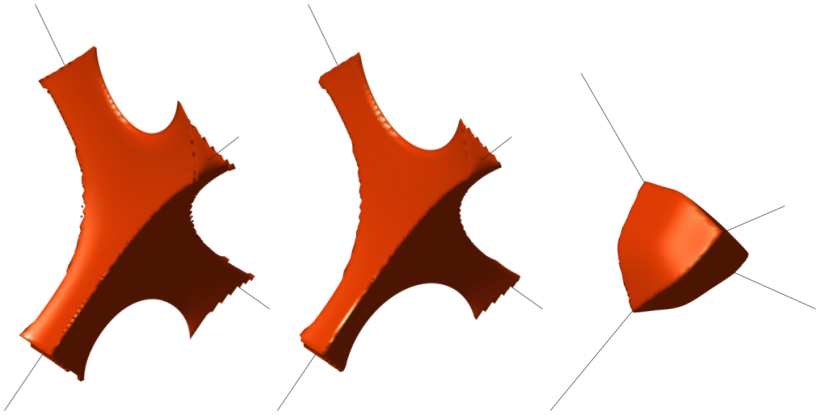
Figure 2.5: Effect of θ on the equilibrated geometry of a two-dimensional single pore. Calculations are done using level set method and interfacial area minimization. (a) $\theta = 10°$, (b) $\theta = 30°$, (c) $\theta = 90°$. (d) Comparison of mean curvature of the solid-liquid interface obtained from level set method and interfacial area minimization. The porosity is 10% in all simulations.

solution in two-dimensions. The error in mean curvature value and dihedral angle is very small in two-dimensions and the statement is true even with less number of grid points per grain. In this work all the simulations are done with $N_{grid}/l_c = 20$, where l_c in all the cases is kept 2 and $\Delta x = 0.1$.

2.4.2 Three-Dimensional Simulation

No analytic solution for texturally equilibrated pores is available in three dimensions. Similarly the comparison with previous work is difficult because the source codes are not available. The validation of the three dimensional results is therefore limited to demonstrating that the computed interface has constant curvature and that the constraints on the dihedral angle and porosity are satisfied.

Fig. 2.6 presents the equilibrated pore geometry in least symmetric sub-volume element in a texturally equilibrated material comprised of truncated octahedron grains. Close inspection of Fig. 2.6 shows that the 4-grain junctions formed by truncated octahedron does not have tetrahedral symmetry that is assumed to simplify the computations in previous works (von Bargen and Waff, 1986; Beere, 1975; Takei, 2002). For isotropic and near isotopic network of grains, change of dihedral angle from 0° to 60° has moderate effect on shape of pores. Furtherer change in dihedral angle ($> 70^\circ$) flips the sign of curvature on solid-liquid interface and changes the geometry of the equilibrated pore space dramatically.



(a) (b) (c)

Figure 2.6: Effect of θ on the equilibrium pore geometry at a junction formed at the intersection of four truncated octahedral grains. The visualized pore space is cut from a network of 6×6×6 grains, with 3% porosity (a) $\theta = 10^\circ$, (b) $\theta = 30^\circ$, (c) $\theta = 90^\circ$.

Normalized distribution of mean curvature near solid-liquid interface for five con-

ditions with same porosity ($\phi = 3\%$) and different dihedral angles are plotted in Fig. 2.7a. The data is collected from a network of 6×6×6 isotropic truncated octahedron grains. As the total number of grid points near interface is different in each case, the distribution is normalized by the total number of data points and a Gaussian fit is plotted along with data. The distribution of the curvatures shows strong maximum but a heavier tails than a Gaussian function. The situation is very similar to curvature measurements from experimentally detected liquid-liquid interface data (Armstrong et al., 2012). Fig. 2.7a also shows the distribution of curvature near solid-liquid interface for an anisotropic network of grains, with $\phi = 3\%$, $\theta = 10^\circ$ and $f = 1.5$. Here f is the elongation factor, by which the grains are stretched in z-direction in order to construct anisotropic grains from initial truncated octahedron grains.

Fig. 2.7b shows the distribution of the dihedral angle for the same simulations. In all cases the mean of the dihedral angle distribution is within 2° of the prescribed angle. The spread in curvature and dihedral angles is due to the finite distance of the grid points from solid-liquid interface. Figs. 2.7c-2.7e show the interpolated curvature on solid-liquid interface which is essentially uniform. Figs. 2.7f and 2.7g present the effect of grid size on distribution of curvature and dihedral angle for a case study with $\phi = 2\%$ and $\theta = 30^\circ$. Refinement reduces the distribution of the values but does not affect the mean values. This suggests that even computations with the coarsest grid give the correct interface. Error bars in Figs. 2.7f and 2.7g are derived from standard deviation of fitted Gaussian curves to normalized distribution of curvature and dihedral angle data. As time step required for stability of numerical simulation reduces with using smaller grid size ($\Delta t \propto \Delta x^4$), simulations for studying the grid size were limited to four grains.

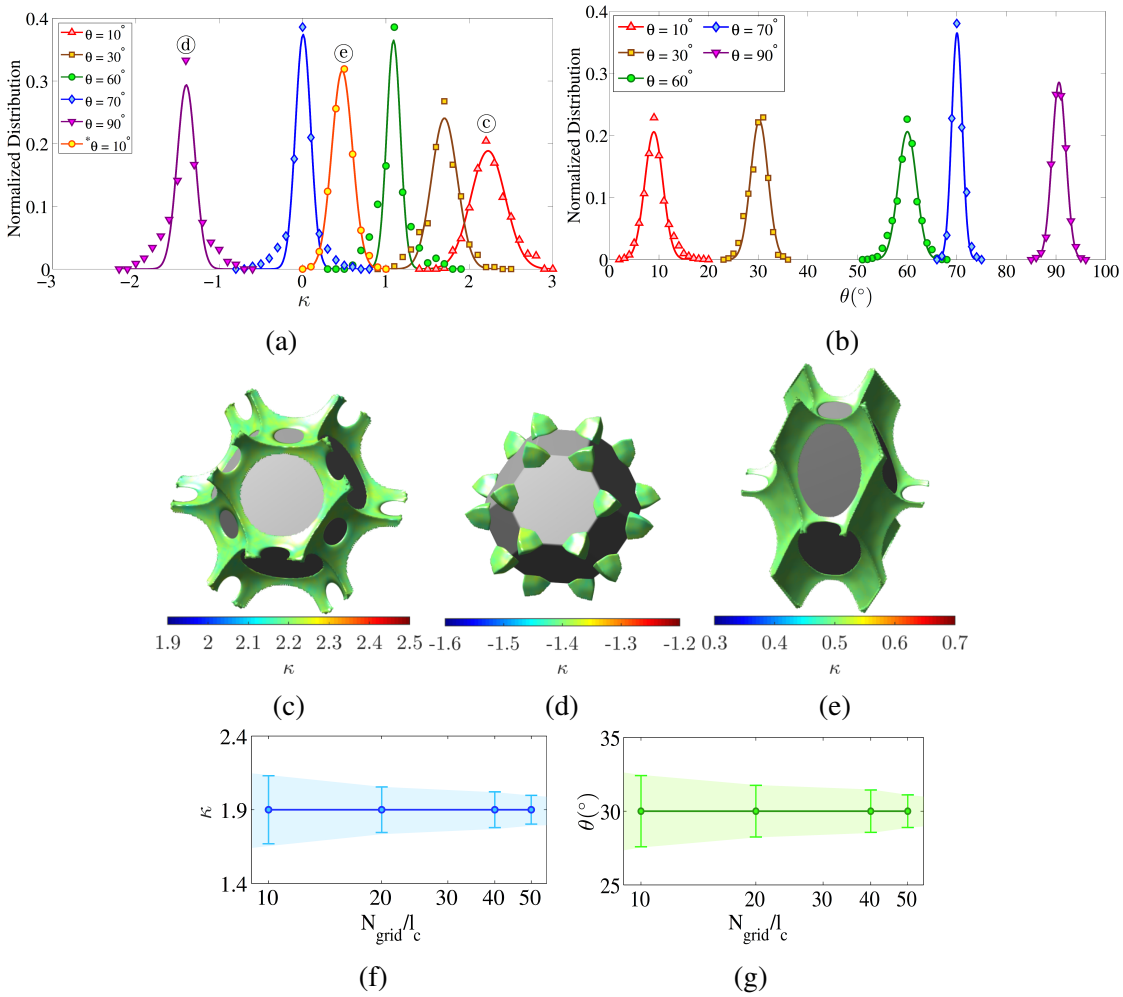
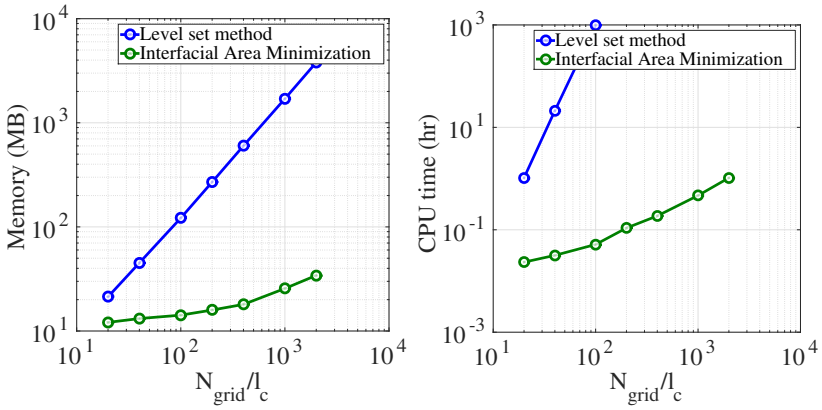


Figure 2.7: (a-b) Distribution of mean curvature (κ) and dihedral angle (θ) around solid-liquid interface in three-dimensional simulations. A normal distribution function (Gaussian) is fitted to each data set for statistical analysis. In simulations, the porosity is kept 3% and $N_{grid}/l_c = 20$. Solid network is comprised of 6×6×6 truncated octahedron grains. (c-e) Visualization of solid liquid interface colored with mean curvature for cases marked in (a). The interface color and the color bar show that the curvature is almost constant in all the cases. (f-g) Effect of grid size on mean value and error of κ and θ. Due to time intensity of simulations, only four grains are considered in simulations. Finer grid size makes the standard deviation of data smaller but doesn't change mean value.

2.5 Simulation Performance

Fig. 2.8a shows the memory usage for two-dimensional simulation comparing memory intensity of interfacial area minimization and the level set method presented here.

The computation domain consist of a network of 10×10 grains. Fig. 2.8b presents the CPU time for same simulations with different grid sizes. Simulation times are orders of magnitude larger for level set method. CFL condition for stability of explicit integration requires $\Delta t < C\Delta x^4$ and naturally simulation time grows with slope at least 4 on log-log scale. We were not able to finish simulation with $N_{grid}/l_c > 100$.



(a) (b)

Figure 2.8: Effect of grid size on (a) memory usage and (b) CPU time for both level set method and intefacial area minimization problem with identical two-dimensional network of grains. As shown, level set model, by orders of magnitudes, is more computationally expensive.

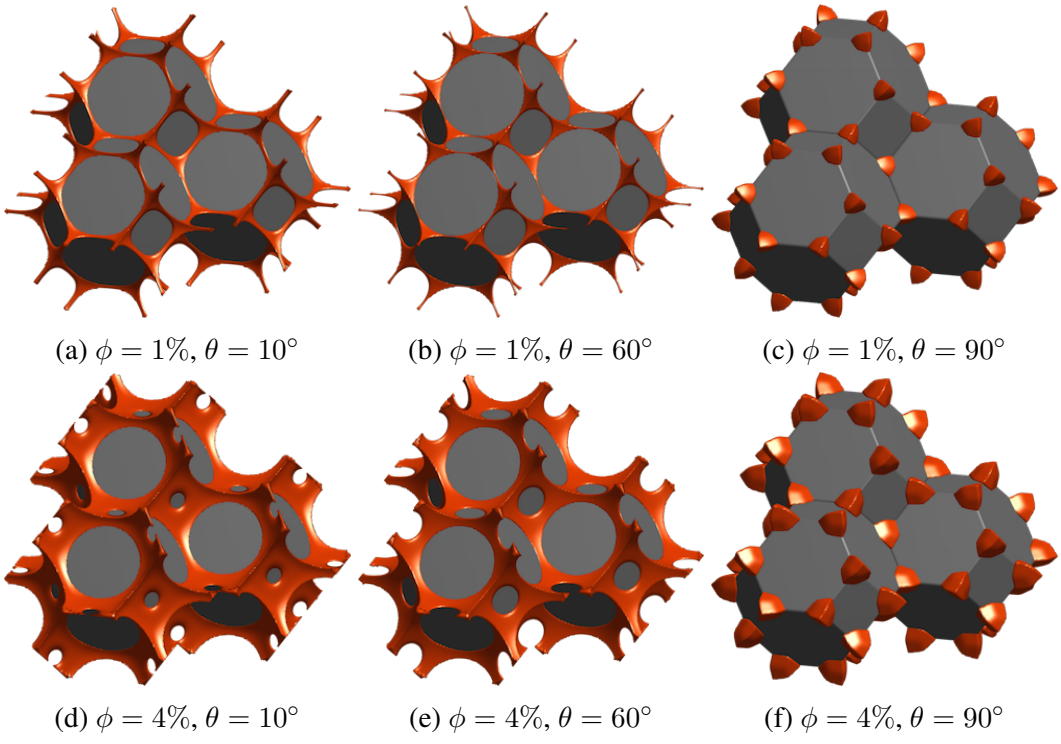
A comparison between Eq. 2.5 and Eq. 2.19 reveals that in two-dimensional simulations, Eq. 2.5 is an ODE but in the level set method, we solve a nonlinear system of PDE in two-dimensions in order to get one-dimensional interfacial curve (zero-level set). This makes the level set method computationally expensive. Although application of Eqs. 2.3 and 2.5 to minimize the interfacial area subject to dihedral angle boundary condition reduces computational cost, tracking topological changes, specially in 3-D, is not trivial. The level set method takes advantages of an implicit form, allowing explicit time integration of highly non-linear terms on a fixed Cartesian grid. In this work, all computations are performed on a single processor (Intel(R) Xeon(R) CPU E3-1270 3.50 GHz and with 32

GB RAM).

2.6 Simulation Examples and Discussion

2.6.1 Regular Media Comprised of Truncated Octahedron Grains

In this section we present three-dimensional simulation results in polycrystalline materials comprised of uniform and non-uniform grains. First set of simulations are done on a network of 6×6×6 symmetric truncated octahedron grains. Fig 2.9 shows the distribution of liquid on grain boundaries for different dihedral angles and porosities. As can be seen, liquid phase is connected along the grain edges when $\theta < 60^\circ$. Increasing dihedral angle to values above 60° will result in negative solid-liquid interface curvature and liquid resides in disconnected pockets on grain corners.



(a) $\phi = 1\%$, $\theta = 10^\circ$ (b) $\phi = 1\%$, $\theta = 60^\circ$ (c) $\phi = 1\%$, $\theta = 90^\circ$

(d) $\phi = 4\%$, $\theta = 10^\circ$ (e) $\phi = 4\%$, $\theta = 60^\circ$ (f) $\phi = 4\%$, $\theta = 90^\circ$

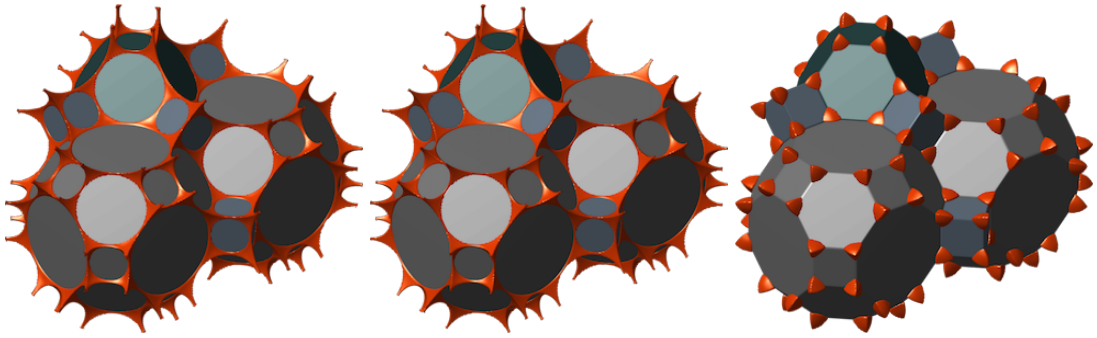
Figure 2.9: Texturally equilibrated pore networks in a polycrystalline solid with an isotropic fabric. Solid network is comprised of 6×6×6 uniform truncated octahedron grains.

2.6.2 Regular Media Comprised of Grains with Different Shapes

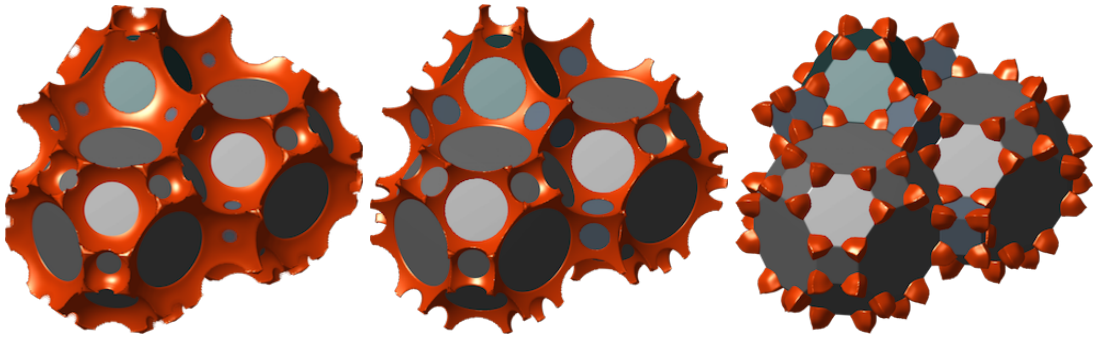
In the second set of simulations, we present the distribution of liquid phase in textural equilibrium with a medium comprised of unequal grains. The chosen polycrystalline lattice is a cantitruncated cubic honeycomb which is a uniform space-filling tessellation in three-dimensions comprised of truncated cuboctahedra, truncated octahedra, and cubes in a ratio of 1:1:3. The results presented in Fig 2.10 show the complexity and richness of texturally equilibrated pore networks and confirm that the pore network is percolating in all directions even for very small porosities when $\theta < 60^\circ$. In the case of $\theta = 10^\circ$ and $\phi = 4\%$ (Fig. 2.10d) the liquid spreads wider on smaller grain boundaries (square faces) and grain boundaries are wetted at smaller porosities in comparison to polycrystalline materials composed of equal grains. Consequently, grain boundary wetting is likely to be more common in polycrystalline materials with disordered or unequal grains. An increase in the dihedral angle at constant porosity reduces the wetting tendency of the liquid and texturally equilibrated pore networks form channels along the grain edges in case of $\theta = 60^\circ$ (Figs. 2.10b and 2.10e). Further increase in dihedral angle results in negative curvature of the solid-liquid interface and the liquid resides in disconnected pockets on grain corners (Figs. 2.10c and 2.10f). Simulation results for a cantitruncated cubic honeycomb lattice shows that the level set method is able to handle the simulation on a polycrystalline solid comprised of grains with arbitrary geometry, as long as the input geometry is space filling and there are two-dimensional solid-solid contact between grains.

2.6.3 Irregular Media Comprised of Distinctive Grains

In this section we are present the computed pore network in a polycrystalline solid comprised of grains with arbitrary shapes and sizes. The initial grain shapes are reconstructed by X-ray diffraction contrast tomography (DCT), which is a technique for the



(a) $\phi = 1\%, \theta = 10^\circ$ (b) $\phi = 1\%, \theta = 60^\circ$ (c) $\phi = 1\%, \theta = 90^\circ$

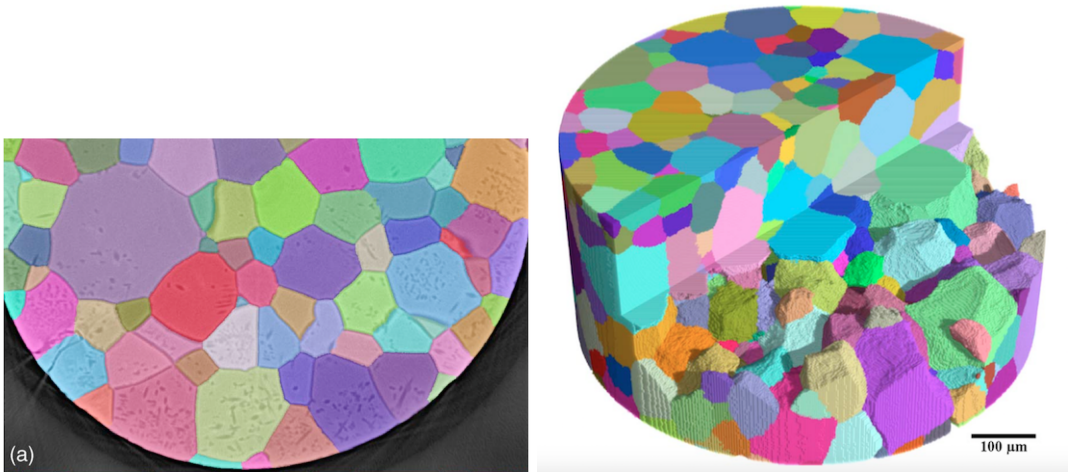


(d) $\phi = 4\%, \theta = 10^\circ$ (e) $\phi = 4\%, \theta = 60^\circ$ (f) $\phi = 4\%, \theta = 90^\circ$

Figure 2.10: Texturally equilibrated pore networks in a polycrystalline solid with unequal grains. Grain configuration is a cantitruncated cubic honeycomb lattice. Results show that the developed model can deal with arbitrary geometries. Displayed images are selected from a domain of 900 grains.

mapping of grain shape and crystal orientation in polycrystalline materials in a nondestructive way (Ludwig et al., 2009). DCT can be regarded as a variant of the techniques generally known as three-dimensional x-ray microscopy (3DXRD). This data set has been made available to us by Ludwig et al. (2009). A visualization of the scanned and reconstructed beta-Ti sample with 1008 grains is shown in Fig. 2.11.

Fig. 2.12 presents the texturally equilibrated fluid distributions in different porosity and dihedral angle values in the irregular media constructed by DCT. As expected, pore network is connected in all porosities when $\theta < 60^\circ$. In the case of $\theta = 10^\circ$, the fluid is initially distributed only on the grain edges in small porosities, and increasing the porosity



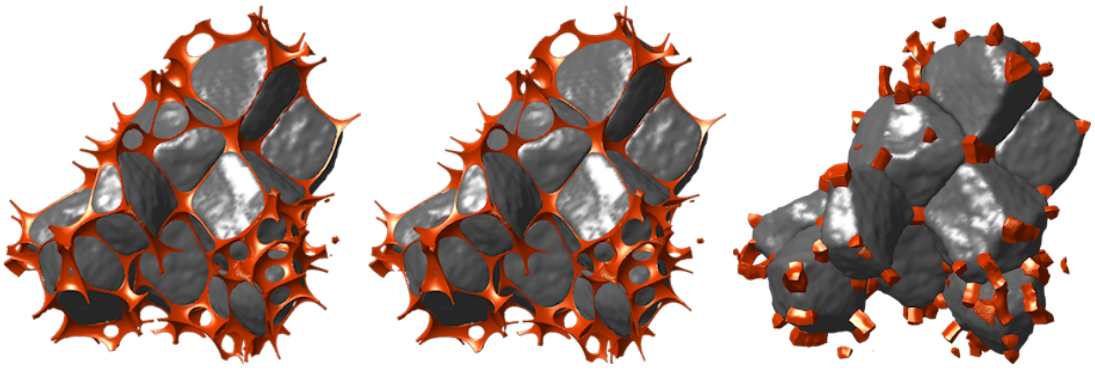
(a) (b)

Figure 2.11: (a) Two-dimensional cross section of the segmented grains with DCT. (b) Three-dimensional visualization of the scanned cylindrical specimen (Ludwig et al., 2009). The images are used with permission from American Institute of Physics.

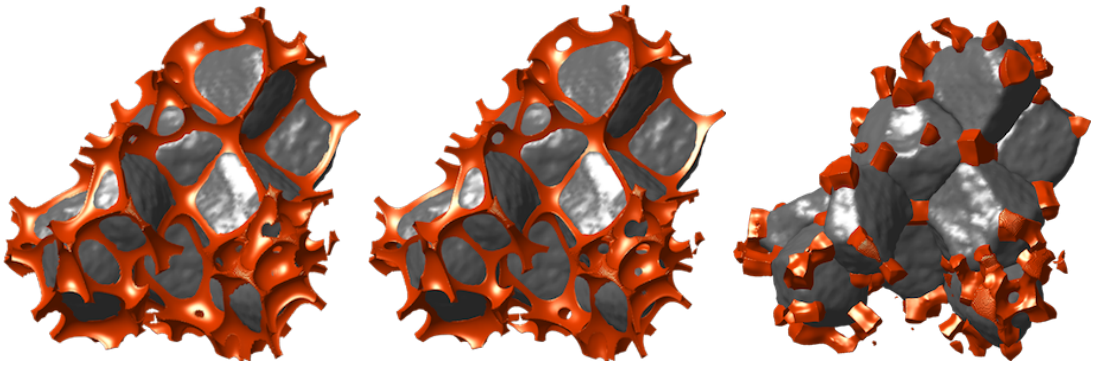
to $\phi = 5\%$ spreads the fluid on some small grain faces. When $\theta = 60°$, the fluid has a neutral tendency to wet the grain faces and stays on channels on the grain edges. In the case of $\theta = 90°$, the fluid is resides in isolated pockets in small porosities. As the porosity increases, some of the stranded fluid pockets become connected and form cuboid of different sizes. The shrinkage of the fluid pockets at the time they connect, makes the establishment of a connected path through the sample possible in much higher porosities. The presented three dimensional simulation results (Figs. 2.9, 2.10 and 2.12) show the level set approach developed here is applicable to polycrystalline materials comprised of grains with arbitrary shapes.

2.7 Effects of Anisotropy in Grain Fabric

Textural equilibrium provides a powerful model to understand the first-order properties of complex polycrystalline two-phase materials based only on knowledge of the sur-



(a) $\phi = 2\%, \theta = 10°$ (b) $\phi = 2\%, \theta = 60°$ (c) $\phi = 2\%, \theta = 90°$



(d) $\phi = 5\%, \theta = 10°$ (e) $\phi = 5\%, \theta = 60°$ (f) $\phi = 5\%, \theta = 90°$

Figure 2.12: Texturally equilibrated pore networks in a polycrystalline solid comprised of grains with arbitrary shapes and sizes. Grains are reconstructed with X-ray diffraction contrast tomography and the chosen material is a β-titanium alloy Timet 21S sample (Ti-15Mo-3Nb-3Al wt%) (Ludwig et al., 2009). The scanned section of the sample includes 1008 grains, the resolution of scan is 0.56 μm with average grain size of 55 μm.

face energies or equivalently the dihedral angle. Although Fig 2.1 illustrates qualitative similarity between the model and observations, there is considerable debate if real systems approach equilibrium sufficiently for the model to be predictive. Computations of texturally equilibrated pores at isolated tetrahedral grain corners show accumulation of the melt along grain edges (Fig. 2.1*a*) and no wetting of entire grain boundaries (Beere, 1975; von Bargen and Waff, 1986; Cheadle, 1989; Nye, 1989). If this theoretical prediction is true it places important bounds on various physical properties (Hirth and Kohlstedt, 1995; Takei, 2002; Endres et al., 2009), which are generally hard to determine experimentally.

On the other hand, it is generally accepted that the observation of wetted grain boundaries would provide strong evidence that textural equilibrium has not been attained. Proving the existence of wetted grain boundaries conclusively, however, has been challenging. Initial attempts were not conclusive due to the limitations of manual serial sectioning and the difficulty of reconstructing three dimensional geometries from two-dimensional images (Waff and Faul, 1992; Faul, 1997; Faul et al., 1994; Cmiral et al., 1998). Later microtomography provided three dimensional images of the melt distribution (Fig. 2.1a), but the resolution of the images is not sufficient to conclusively image melt films along grain boundaries at low porosities (Zhu et al., 2011). Most recently, advances in high-resolution serial sectioning have provided evidence for wetted grain boundaries (Garapic et al., 2013). Here we address the fundamental assumption underlying this debate and investigate if grain boundary wetting is possible in low-porosity texturally equilibrated networks.

Ductile deformation of polycrystalline materials often generates a fabric of elongated and aligned grains that induces anisotropy in the physical properties of the porous medium. Strong anisotropic fabrics occur even in rock salt despite the cubic symmetry of the individual salt grains (Schoenherr et al., 2007). To explore the effect of anisotropic grain fabric on texturally equilibrated pore networks, we consider geometries where the solid grains have been stretched in the z-direction by a factor, f. The grains are oriented such that the stretching is normal to one of the square faces.

Fig. 2.13 illustrates that θ controls the geometry of the pore network that develops in a material with an anisotropic fabric. For $\theta < 60^\circ$ the pores percolate for all ϕ investigated and the the grain boundaries parallel to the direction of stretching are wetted once ϕ exceeds a threshold. Unlike isotropic media, where grain boundaries are only wetted at small θ, anisotropic fabrics also allow the wetting of grain boundaries at $\theta > 60^\circ$. In this case, grain boundary wetting occurs at $\phi \approx 1.5\%$ for all $\theta > 60^\circ$. The wetted boundaries

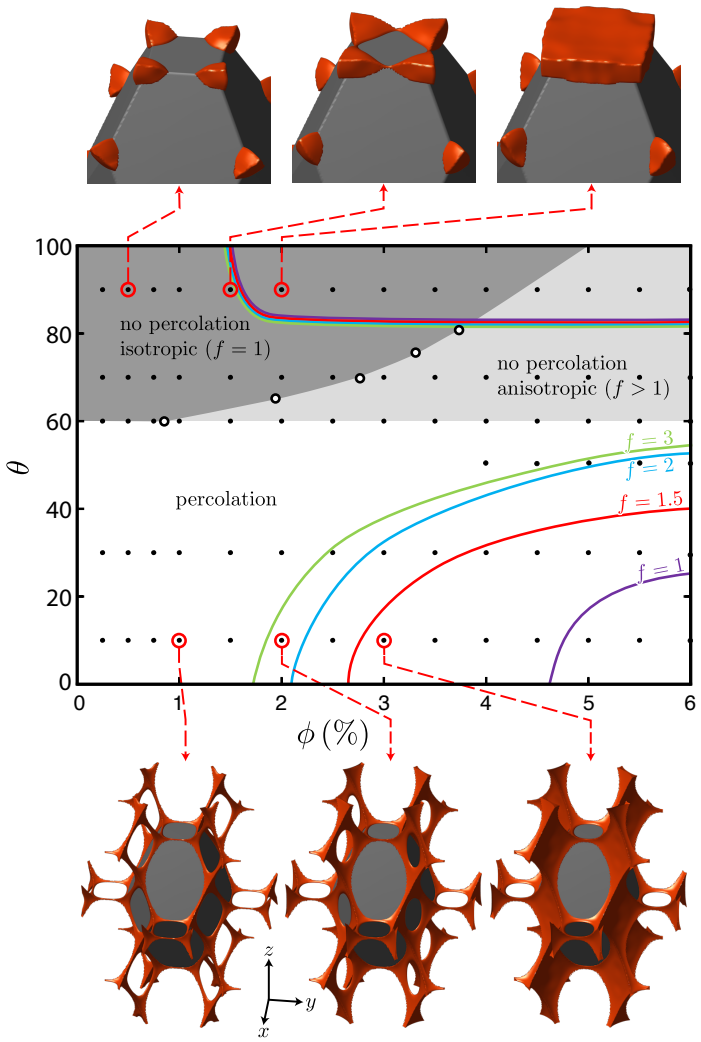


Figure 2.13: The center shows a regime diagram for percolation and grain boundary wetting in $\phi\theta$-space. The dark gray region corresponds to conditions where isotropic media do not percolate. The black circles indicate the previously determined percolation boundary (von Bargen and Waff, 1986). The expanded no percolation region for anisotropic media is shown in light gray. The colored contours indicate the boundaries where grain boundary wetting occurs for different anisotropies. The top row illustrates the formation of wetted grain boundaries for $\theta = 90°$ and $f = 1.5$ as ϕ increases from 0.5% to 2%. The bottom row illustrates the formation of wetted grain boundaries for $\theta = 10°$ and $f = 1.5$ as ϕ increases from 1% to 3%.

are perpendicular to the stretching of the fabric but the pores do not percolate for the investigated range of ϕ. In anisotropic fabrics, grain boundary wetting is minimized for $\theta \approx 60°$.

The percolation of the pore network and the occurrence of wetted grain boundaries is summarized in the regime diagram shown in Fig. 2.13. For $\theta < 60°$ grain boundary wetting occurs at smaller ϕ as the anisotropy of the fabric increases. Given the common occurrence of anisotropic fabrics in natural systems, wetting of the smaller grain boundaries should be common in texturally equilibrated pore networks.

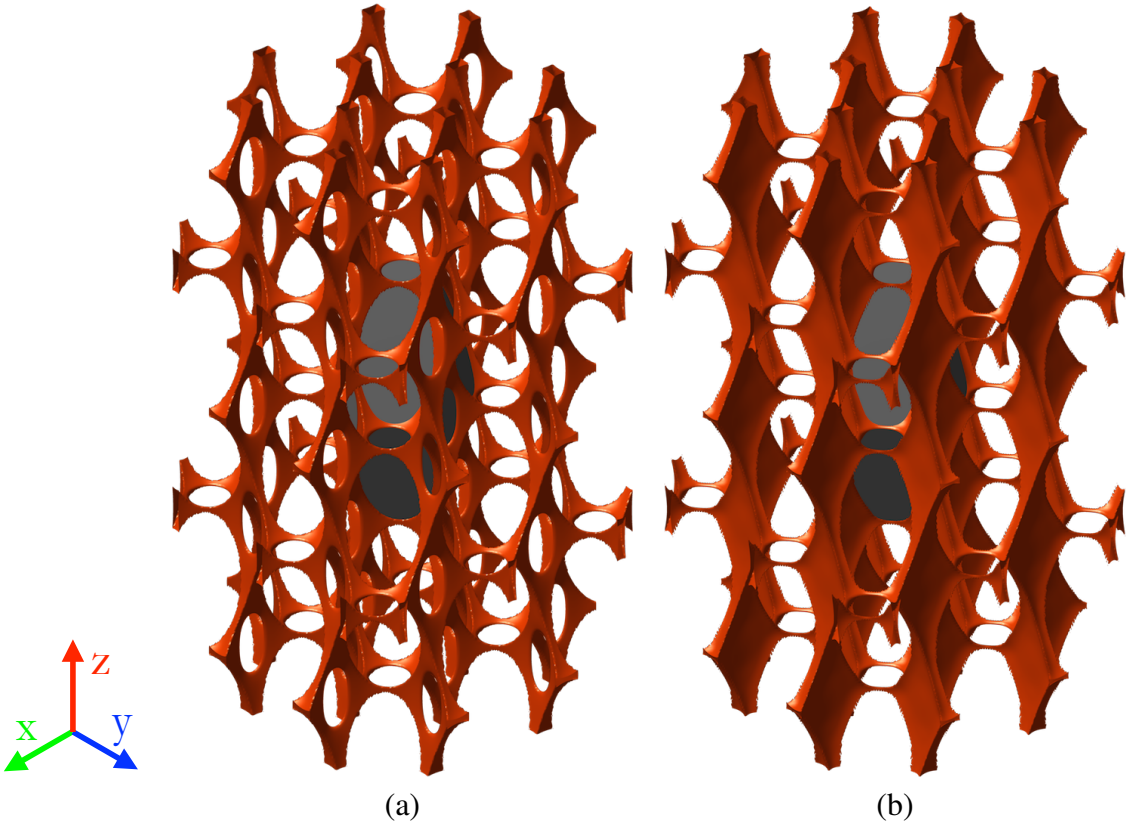


Figure 2.14: Fluid distribution between grains in textural equilibrium with $f = 2$, porosity 3%. (a) $\theta = 10°$, (b) $60°$. The polycrystalline material is stretched in the z-direction.

Even very small stretching factors effectively prevent percolation of pores in materials with $\theta > 60°$, due to contraction of isolated pores illustrated in the top row of Fig. 2.13. In the isotropic fabric, all edges have the same length and the isolated pores connect simultaneously along all edges, once the percolation threshold is overcome. In the anisotropic fabric the pores first connect along the shortest edge. This change in the topology leads to a

contraction of the pores along the other edges, so that subsequent percolation requires very large ϕ. Given the observed small ϕ of texturally equilibrated media, percolation at $\theta > 60^\circ$ in solids with anisotropic fabric is unlikely. The basic connect-contract mechanism is the same in disordered media with variable grain size so that the percolation behavior will be similar.

In contrast to the isotropic case, both channels and tabular pores coexist in anisotropic pore networks at small θ. Since most of the liquid resides in the tabular pores parallel to the direction of stretching, the channels perpendicular to the stretching are very thin at small ϕ (Fig. 2.14*a*). This suggests that strong anisotropies may exist in the physical properties of these pore networks. To test this hypothesis, the permeability of the texturally equilibrated pore networks has been calculated using lattice Boltzmann simulations (Huber et al., 2013). Representative pore networks used in the computations are shown in Fig. 2.14. This hypothesis, anisotropy in physical properties due to anisotropy in grain fabric, is investigated throughly and presented in sections 3.4.2 and 3.5.3.

Chapter 3

Properties of Ductile Rocks with Texturally Equilibrated Pores

3.1 Background and Literature Review

In ductile rocks with texturally equilibrated pores, the dihedral angle, θ, has primary and first order control on rock physical properties via its underlying effects on liquid topology. As discussed in the previous chapters, the dihedral angle, determined by the ratio of the solid-liquid and solid-solid interfacial energies controls the percolation and distribution of the the liquid phase in ductile rocks. Several authors have done numerical and experimental study to determine the connectivity and percolation of these rocks (von Bargen and Waff, 1986; Cheadle et al., 2004; Pervukhina and Kuwahara, 2008; Takei and Hier-Majumder, 2009; Yoshino et al., 2003). However, all the computational studies are limited to simplifying assumptions that result in fluid pieces that do not link to a three dimensional network. Moreover, experimental determination of percolation is generally hard and imposes new challenges for validation (Ghanbarzadeh et al., 2014; Yoshino et al., 2003).

Calculation and measurement of permeability in the texturally equilibrated pore networks is vital in study of fluid flow in ductile rocks. Permeability has been measured experimentally in few studies (Wark and Watson, 1998; Liang et al., 2001; Roberts et al., 2007; Watson and Roberts, 2011; Miller et al., 2014). McKenzie (1984) has fitted a Blake-Kozeny-Carman type equation ($k = d^2\phi^n/b$) to available data in order to represent permeability as a function of porosity. This widely accepted correlation considers the Kozeny

constant (b) to be 1000 and n to be 3. This equation does not consider the effect of θ and anisotropy in grain textures. A wide range of values for $1 < n < 3$ and $200 < b < 10^4$ have been suggested (McKenzie, 1989; Riley Jr. and Kohlstedt, 1991; Minarik and Watson, 1995; Wark and Watson, 1998; Parsons et al., 2008). von Bargen and Waff (1986) and Cheadle et al. (2004) have calculated the permeability from equilibrium geometry and introduced a correction factor to include the effects of porosity and dihedral angle.

Fluid distribution determines the electrical properties of equilibrated material. Electrical resistivity data of mantle provides essential information about connectivity, distribution, water content (in the form of dissolved hydrogen), strain-induced anisotropy of grains and differences in spreading rate between plates (Evans et al., 1999; Yoshino et al., 2009; Bagdassarov et al., 2009). Also conductivity of polar ice sheets could be due to the presence of mixture of acids with water layers at the grain boundaries Wolff and Paren (1984). There is no doubt that electrical properties of the rock salt would also be affected by connectivity or disconnectivity of brine. Given extensive applications of electrical conductivity, to our knowledge, there is only one study by Pervukhina and Kuwahara (2008) that has calculated effective medium properties using textual equilibrium geometry of von Bargen and Waff (1986). As mentioned before, the fluid channels in the work by von Bargen and Waff (1986) does not link to a three dimensional network.

3.2 Percolation and Percolation Threshold

In this section, the focus is on determining the condition at which the texturally equilibrated pore networks percolate. In order to this, we successively change the porosity and dihedral angle in the simulations with the level set method. The connectivity of pore space in the resulting network is then determined using a grass fire algorithm. A percolation map can be established using this dataset to characterize "percolation" versus "no

percolation" regions.

3.2.1 Regular Media

Here we compute texturally equilibrated pore networks in a regular polycrystalline solid comprised of truncated octahedra (Fig. 1.4*a*). Fig. 3.1 shows computed pore networks around a single grain for a range of dihedral angles and porosities. These confirm that the liquid is connected via channels along grain edges and forms a percolating pore network for all investigated ϕ when $\theta < 60^\circ$. For $\theta > 60^\circ$, the liquid shrinks along grain edges and forms isolated pores on corners where four grains meet. The percolation threshold for these isolated pores increases with the dihedral angle (von Bargen and Waff, 1986). The grain edge channels that form above the percolation threshold are increasingly unstable due to a Rayleigh instability as θ increases (Carter and Glaeser, 1987).

The percolation map for a polycrystalline solid comprised of equal truncated octahedron grains is shown Fig. 3.2. The dots indicate the parameter combinations (ϕ and θ) that have been investigated in this study. The gray area shows the region in $\phi - \theta$ space that the pore network do not percolate and the white area shows the percolating region. The percolation threshold is plotted with solid black line and is obtained by averaging the porosities that fall between the "percolation" and "no percolation" regions. As expected, pore networks percolate at all porosities when the dihedral angle is below 60°. Percolation threshold increases from 2% for $\theta = 70^\circ$ to 11% for $\theta = 120^\circ$.

3.2.2 Irregular Media

We follow a similar procedure to establish a percolation map for an irregular polycrystalline material comprised of grains with different shapes and sizes. The details of the solid structure is provided in section 2.6.3. We should note that the results in this sec-

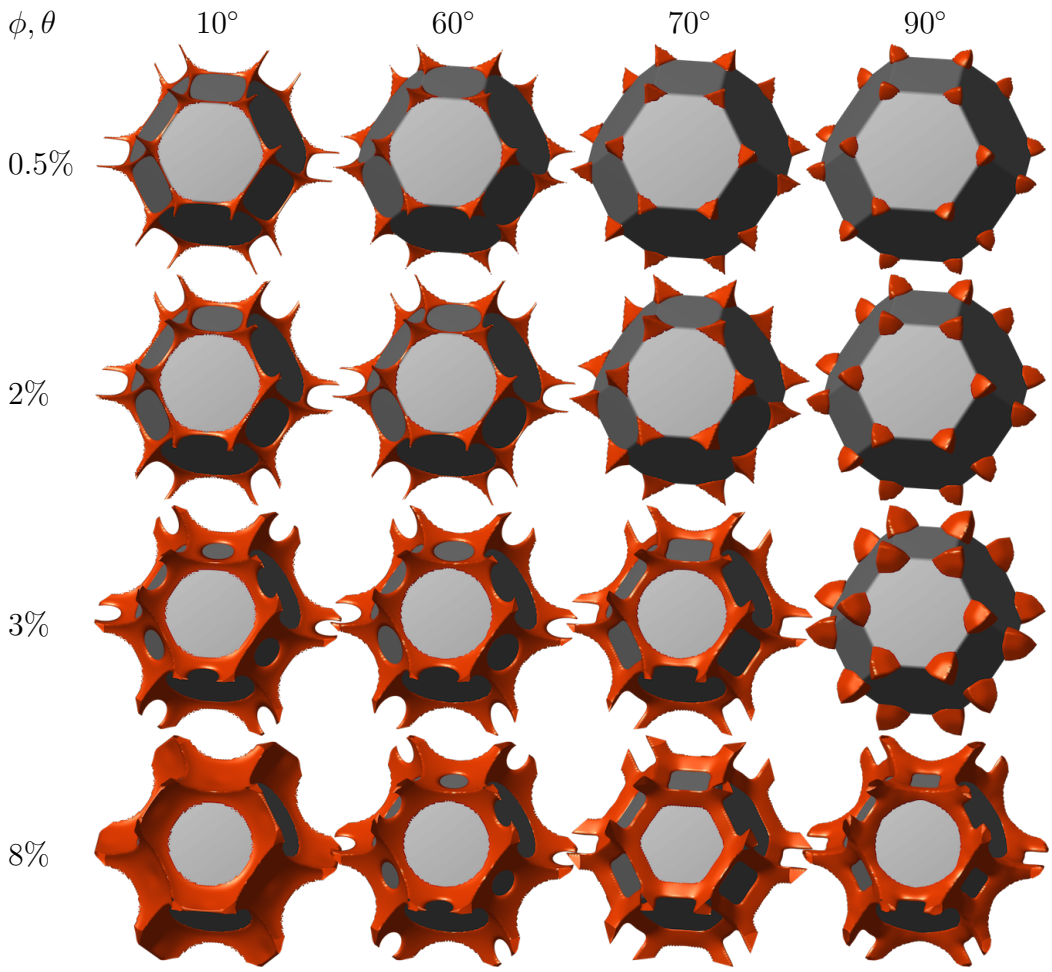


Figure 3.1: Texturally equilibrated pore networks in a polycrystalline solid with an isotropic fabric. Only a single grain extracted from a network of 200 grains is shown.

tion is only valid for this irregular material, but it may be approximately extended to other materials with a variation in grain shape and size.

In this part, the value of the porosity is incrementally increased from 0.5% to 20% in a series of the simulations with fixed dihedral angle. After each successive simulation, dihedral angle is changed (range between 10° to 120°). As the grains are different, visualization of only one grain may not be representative of the entire pore space. Therefore, Fig. 3.3 represent the fluid distribution in a three dimensional network. For better visual-

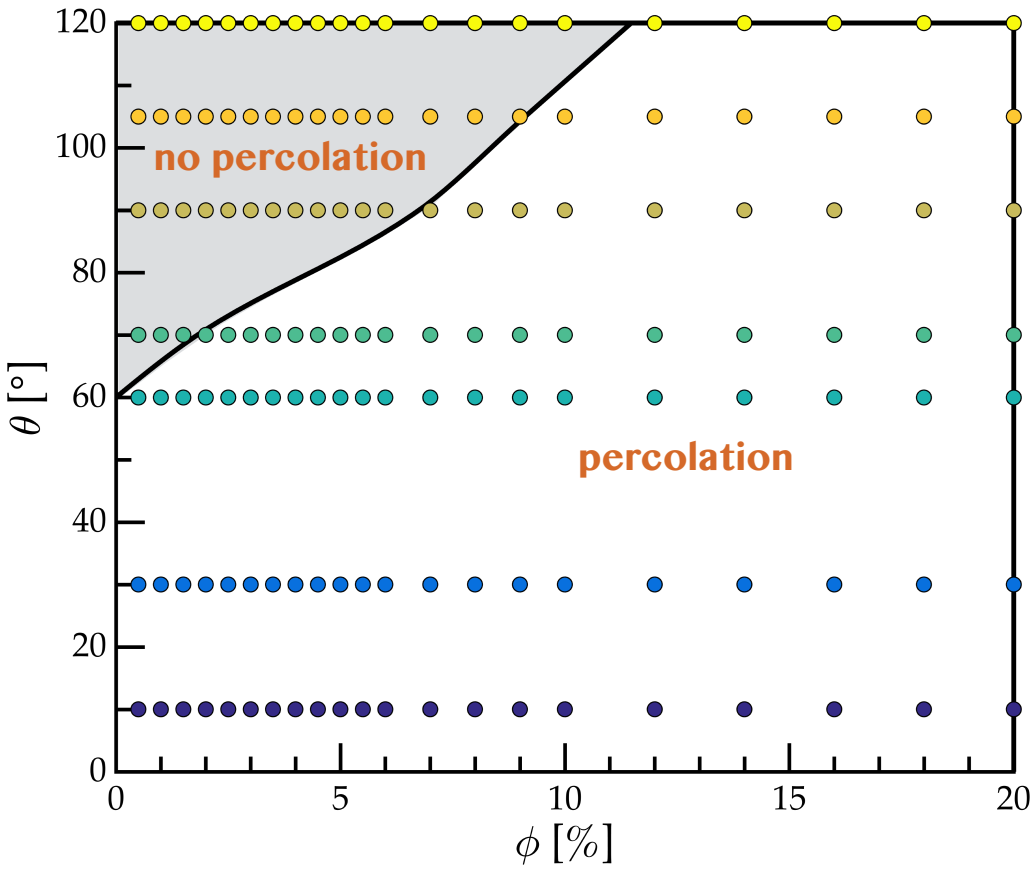


Figure 3.2: Percolation map for a regular polycrystalline solid comprised of truncated octahedra.

ization, only on octant of the entire pore network is plotted.

Fig. 3.4 presents the percolation map for the texturally equilibrated pore networks in an irregular polycrystalline solid. As expected, the percolation in the region with $\theta \leq 60^\circ$ is not affected by the solid structure and the variety in the grain geometry. On the other hand, the percolation threshold is dramatically increased (almost increases by 100%) in comparison to regular solid (compare with Fig. 3.2). This highlights the complexity and richness of the pore networks in this sample and emphasizes the importance of developing the level set method that can handle arbitrary geometries. As the grains have different sizes and shapes, we have a distribution of the grain edge length in the sample. When dihedral

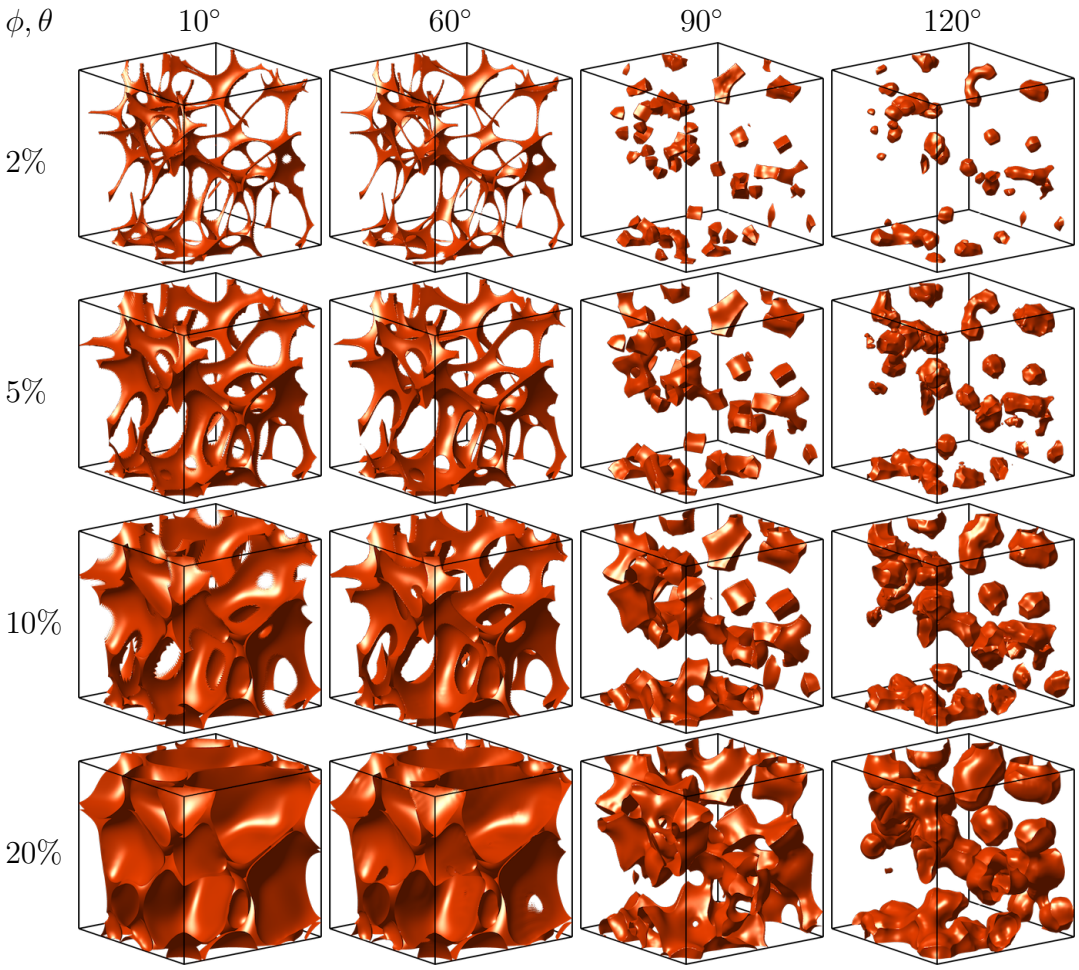


Figure 3.3: Texturally equilibrated pore networks in a polycrystalline solid with different grain shapes and sizes. Only 1/8 of the entire pore network is plotted for better visualization.

angle is larger than 60°, with incrementally increasing porosity from small values, some of disconnected fluid pockets become connected on the shorter grain edges and shrink. Then the required porosity to connect the next fluid pockets, and ultimately connecting enough isolated pockets to create a percolating path in the sample, increases due to this shrinkage. Therefore, we have a larger gray or the "no percolation" zone in this case. The large percolation threshold in the realistic systems might be the reason researchers do not consider to have a permeable sample or percolating pore space once the dihedral angle is

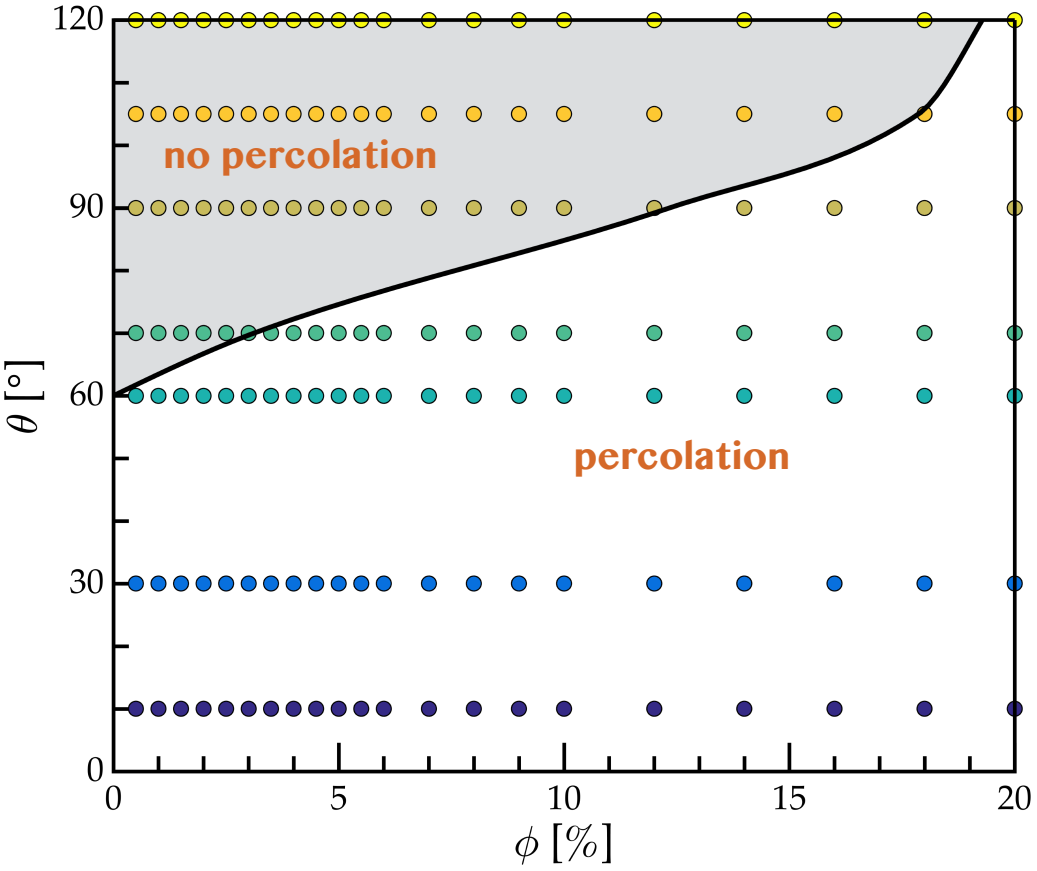


Figure 3.4: Percolation map for an irregular polycrystalline solid comprised of grains with different shapes and sizes.

above 60°.

3.3 Hysteresis in Pore Network Connectivity

In this part we demonstrate the existence of non-unique answers for complex non-linear system presented by Eq. 2.19 with two- and three-dimensional examples. In all the simulations, except the initial porosity, parameters are initialized with the equilibrium interface from the last porosity (As discussed in previous chapter, for the first value of porosity, all the parameters are initialized by solid grain representations). A sequence of simulations

with increasing or decreasing porosity can therefore be interpreted as successive stages in a melting or freezing process. This also resembles the condition in natural systems. For example, in partially molten rocks, the melting, drainage of melt and also freezing happen successively. The imbibition of brine in the rock salt, and the subsequent compaction and the drainage is also a step-by-step process and happens in a successive way. All this introduces an effect of material history in the sense that the next simulation will evolve towards a new equilibrium interface that is topologically similar to the previous one. Here we use the level set method and compute the equilibrium pore network topology with increasing the porosity to a finite value and then decrease the porosity in an incremental way. This reveals that the topology is a function of the history of the system and we introduce the idea of hysteresis in pore network connectivity.

3.3.1 Bifurcation of the Pore Network Topology in 2D

Fig. 3.5 shows the mean curvature of the solid-liquid interface as a function of porosity for two-dimensional elongated honeycomb grains with $f = 1.05$ and a dihedral angle of 70°. As the porosity is increased, starting from zero, disconnected liquid pockets with negatively curved interfaces appear on triple junction shown in Fig. 3.5*a*-3.5*c*. As the porosity is increase the isolated pockets touch at $\phi = 28\%$ along the shorter interfaces and form a connected horizontal pore network. As the pockets become connected in horizontal direction, the curvature changes sign and liquid recedes along the vertical grain boundaries (Fig. 3.5*d*). In this set of simulations the porosity is increased to 40% before it is decreased again. As the porosity is decreased below 28% the interface does not disconnect along the shorter grain edges (Figs 3.5*d*-3.5*f*). The porosity must be decreased below 10% to disconnect the interface, so that, the liquid has two possible geometries with different topologies which satisfy all physical requirements for textural equilibrium. Figs. 3.5*b* and 3.5*f* and also Figs. 3.5*c* and 3.5*e* are showing the fluid distributions for same porosity, dihedral angle

and grain configurations and the only difference is in the history which polycrystalline solid has experienced. Arrows in Fig. 3.5 show the path of increasing and decreasing porosity and filled and empty data points show connected versus disconnected porosity, respectively.

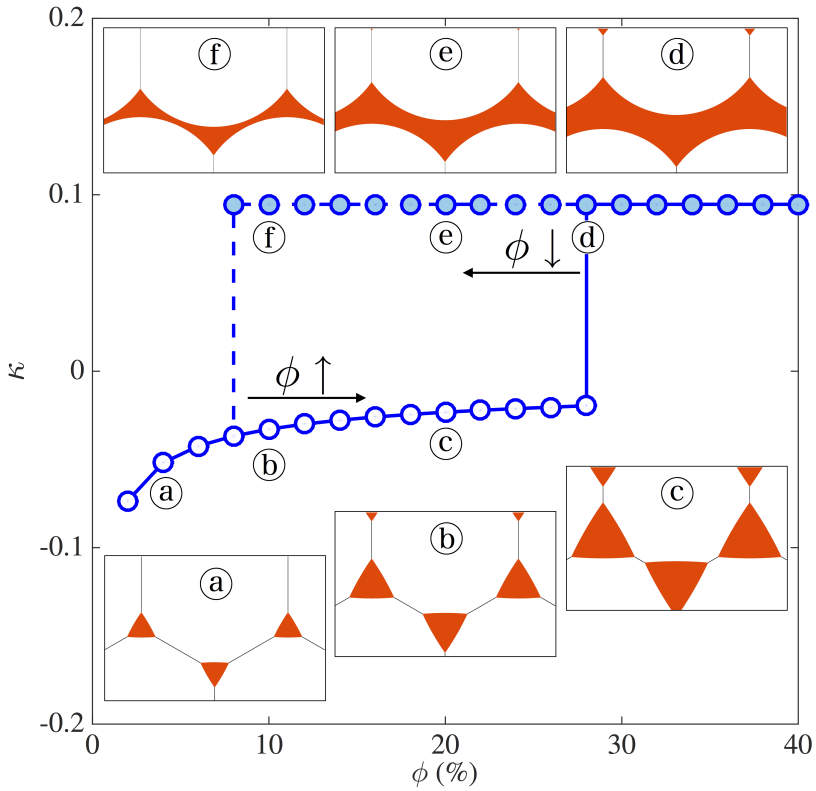


Figure 3.5: Mean curvature of solid-liquid interface as a function of porosity in network of hexagonal grains with $f = 1.05$ and $\theta = 70°$. Different topologies result from different curvature values of solid-liquid interface subject to identical constraints which originates from tendency of the system to keep its previous connectivity state. This results in history dependency of topology. Filled markers denote to the percolating pore networks.

3.3.2 Regular Media

The bifurcation of the pore network topology, resulting from the history of the material also exists in three-dimensional solutions. Fig. 3.6 represents the simulation results for a network of truncated octahedron grains with $\theta = 90°$. Simulation starts at zero porosity and it increases to 6% with 0.5% increments. Individual liquid pockets on grain edges

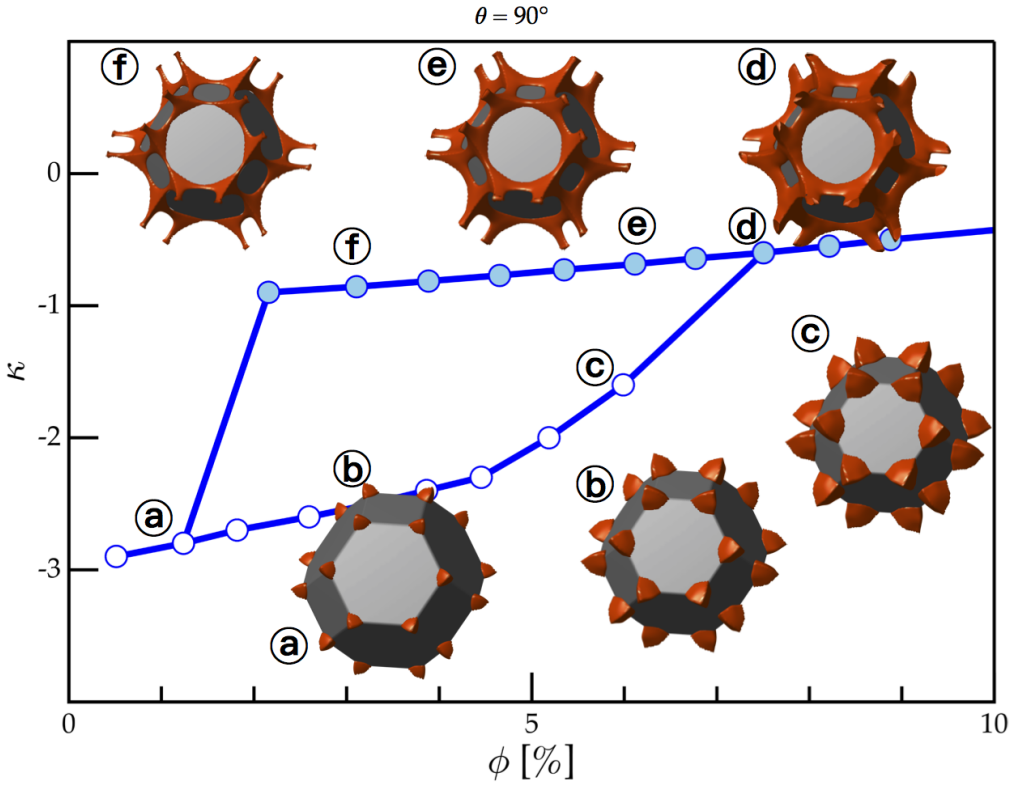


Figure 3.6: Mean curvature of solid-liquid interface as a function of porosity in a regular medium with truncated octahedron grains and $\theta = 90°$. Filled markers denote to the percolating pore networks. Texturally equilibrated liquid distribution is visualized in some porosities. History of the system is determining the connectivity of the pore network.

become connected when $\phi = 5\%$. When the final porosity is obtained, the porosity decreases with the same increment. The pores remain connected on grain edges even at small porosity of $\phi = 1\%$. Figs. 3.6*b* and 3.6*f* are representing liquid distribution for same porosity, dihedral angle and grain configurations and the only differentiating characteristic is the history of system. The same conditions apply to Figs. 3.6*c* and 3.6*e*.

This shows that pore network topology is affected by history of the system and hysteresis is playing a role in final pore geometry. As simulations are performed in the series of increasing or decreasing ϕ, using the last solution as the first guess, the initial condition determines which solutions is found. This has extensive effects on behavior

of materials with texturally equilibrated pores and the different configurations would be expected to be realized in melting and freezing processes.

3.3.3 Irregular Media

The history dependency of the pore network connectivity is also tested and verified for the irregular media presented in section 2.6.3. In this series of simulations (Fig. 3.7), the dihedral angle is fixed and considered to be 90°. The liquid level set is initialized with solid grain level set functions at zero porosity. The porosity increases successively until a percolation path is established in the sample (in this case at 13%) porosity. Porosity is further increases to 20% and then incrementally decreased. Once the percolation threshold is passed on the way decreasing the porosity, the pore network remains connected. The connectivity of the pore space is maintained until very small liquid volume fraction is left in the sample. This introduces the existence of a new threshold at which the percolating liquid network becomes disconnected. We call this threshold the "trapping threshold" (Fig. 3.7*f*).

3.3.4 Revised Percolation Map

As mentioned in the last section, hysteresis in pore network topology introduces a region where the texturally equilibrated pore networks can be either connected or disconnected based on the history of the system. This region can be added to the percolation maps presented in Figs. 3.2 and 3.4. Fig. 3.8 illustrates the percolation and trapping threshold with solid black lines for both regular and irregular media. The filled dots correspond to the pore networks that are percolating independent of the history of the system, empty dots correspond to disconnected pore networks and the half-filled dots represents the tested (ϕ, θ) pair that the connectivity of the pore space in a function of history of the system. This region is highlighted by shaded area in both plots in Fig. 3.8.

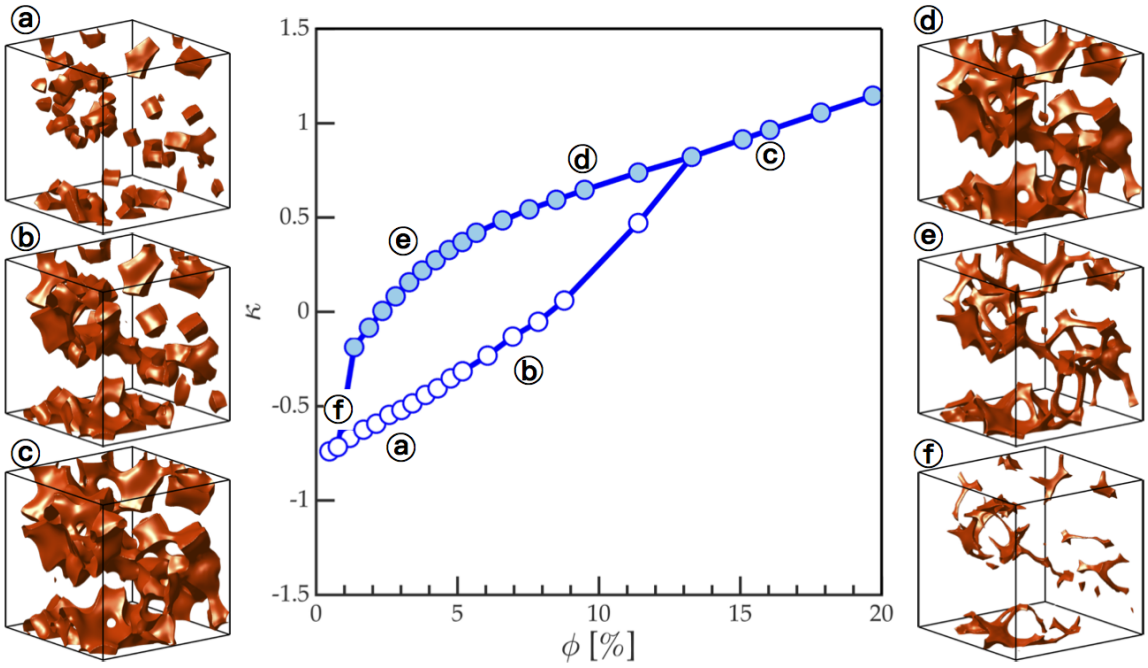


Figure 3.7: Mean curvature of solid-liquid interface as a function of porosity in an irregular medium with different grain shapes and sizes. The dihedral angle, θ, is considered to be fixed and 90°. Texturally equilibrated liquid distribution is visualized in some porosities. Filled markers denote to the percolating pore networks. History of the system is determining the connectivity of the pore network.

3.4 Permeability

The permeability of computed pore networks is calculated using Palabos (www.palabos.org). Palabos in an open source CFD package based on Lattice Boltzmann method implemented in parallel and installed on TACC (Texas Advanced Computing Center). Palabos is written based on Lattice Bhatnagar-Gross-Krook (LBGK) model where momentum of colliding particles will redistribute at some constant rate toward an equilibrium distribution. Permeability is calculated by imposing a constant pressure at the inlet, and a lower pressure at the outlet. The flux is compared with Darcy's law and the resulting constant is permeability. Here we use the computed pore networks in regular media to calculate the permeability of the pore space. We also consider the hysteresis, therefore values of permeability for the area

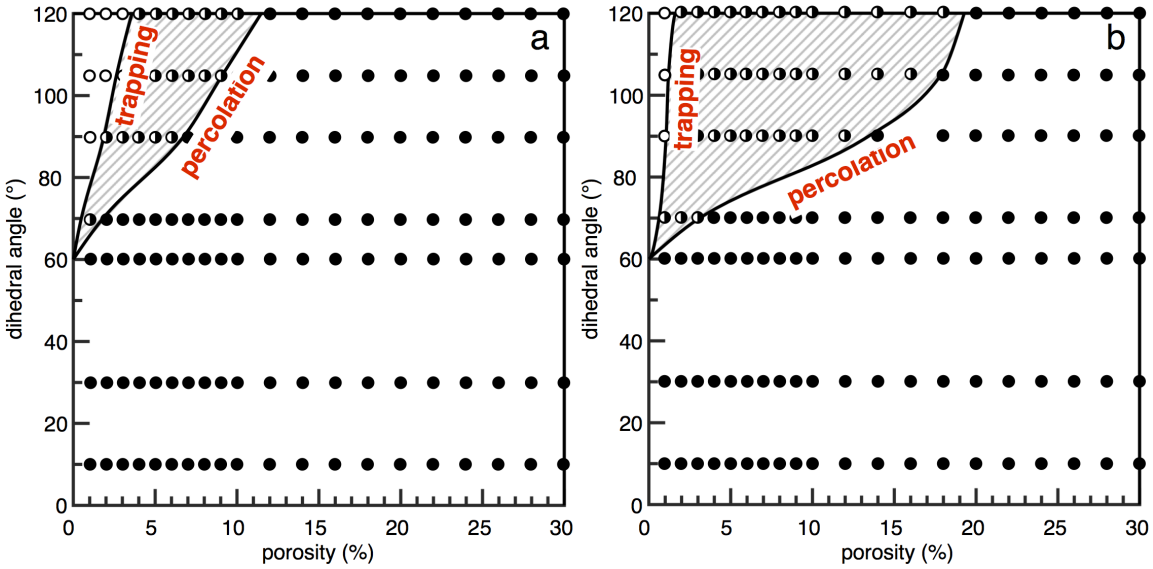


Figure 3.8: Revised percolation map in $\phi\theta$-space for (a) regular media comprised of truncated octahedron grains and (b) irregular media with different grain shapes and sizes. The connectivity of pore space in hatched areas between the trapping and percolation thresholds depends on the history of the system.

between the trapping and percolation thresholds are also computed. Fig. 3.9*a* presents the velocity field (magnitude of velocity) in texturally equilibrated pore network at $\phi = 10\%$ and $\theta = 90^\circ$ (red square in Fig. 3.9*b*). As expected, the fluid channels that are aligned to xy-plain do not contribute to flow in z-direction. Although the channels are connected in x and y directions, this provides an explanation for visual disconnectivity of the pore network presented by velocity field.

Fig. 3.9*b* presents the computed values of permeability versus porosity of the medium in both increasing and decreasing porosity paths for the case of $\theta = 90^\circ$. The permeability values are converted from lattice units to SI [m^2] units by scaling the average grain size to 1 mm. As expected, the permeability of the pore network is initially zero until the porosity exceeds the percolation threshold. After this, the permeability follows a smooth path. Decreasing the porosity to values below percolation threshold does not disconnect the pore

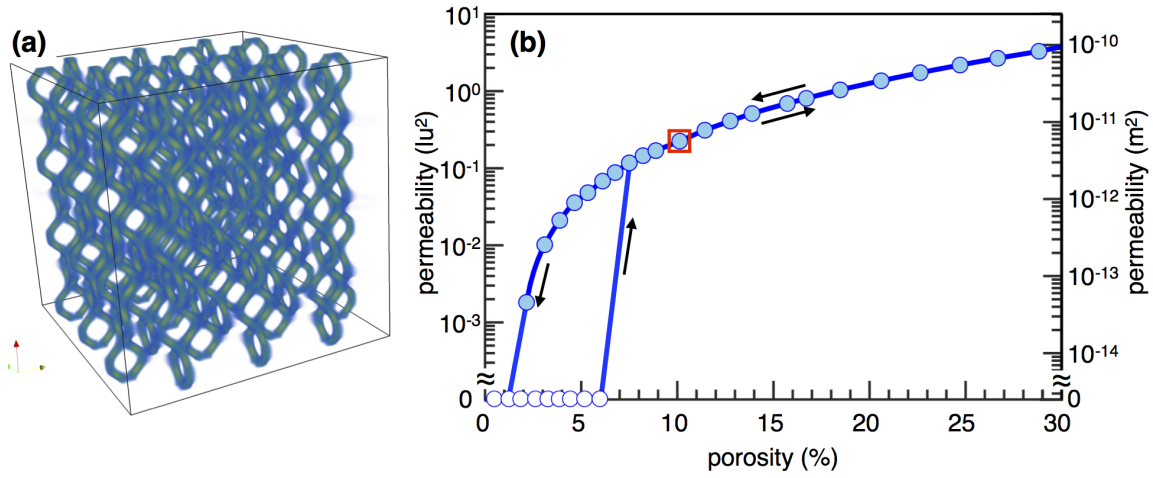


Figure 3.9: (a) Visualization of the velocity field due to pressure gradient in z-direction (red axis) in a regular medium with $\theta = 90°$. (b) Permeability of the corresponding porous material to Fig. 3.6 obtained from lattice Boltzmann simulations is shown in lattice units. Dimensional permeability is shown on right axis assuming average grain size of 1 mm. Hysteresis in pore network connectivity introduces a loop in permeability plot which corresponds to connected versus disconnected regions.

network and permeability does not vanish until trapping threshold is reached. The y-axis on this plot is in log scale and there is not a zero value on a log-scale axis. Therefore, the axis is broken in the bottom.

A similar procedure is repeated for the irregular media comprised of different grain shapes and sizes. Fig. 3.10*a* represents the velocity field in corresponding pore network to $\phi = 10\%$ and $\theta = 90°$ (red square in Fig. 3.10*b*). The flow is established in z-direction (red axis). The permeability values are computed in all the tested values for ϕ and θ (Fig. 3.8) and plotted versus porosity for the case of $\theta = 90°$. The loop in the permeability plot is due to hysteresis in pore network connectivity. The permeability values are converted from lattice units to SI [m^2] units by scaling the average grain size to 1 mm. The y-axis on this plot is in log scale and there is not a zero value on a log-scale axis. Therefore, the axis is broken in the bottom.

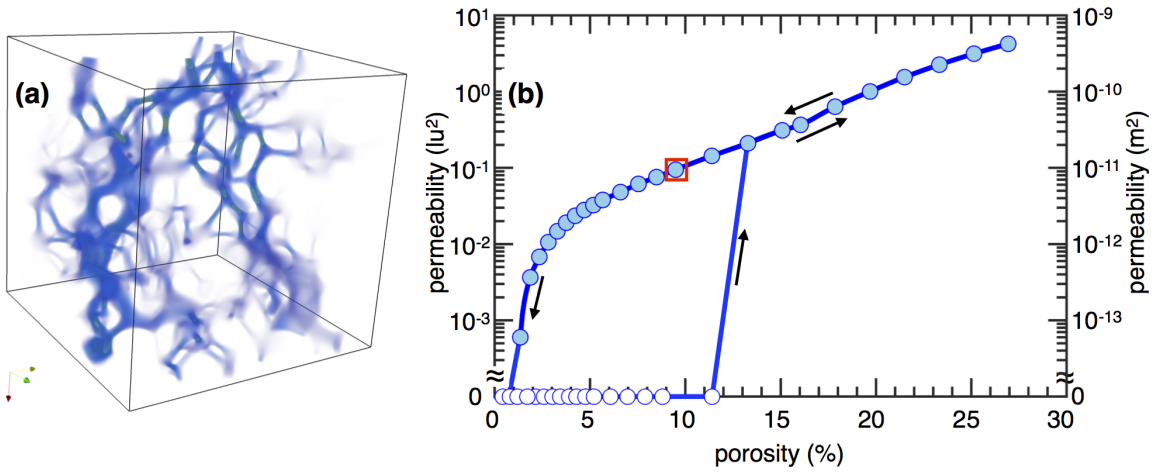


Figure 3.10: (a) Visualization of the velocity field due to pressure gradient in z-direction (red axis) in an irregular medium with $\theta = 90°$. (b) Permeability of the corresponding porous material to Fig. 3.7 obtained from lattice Boltzmann simulations is shown in lattice unites. Dimensional permeability is shown on right axis assuming average grain size of 1 mm. Hysteresis in pore network connectivity introduces a loop in permeability plot which corresponds to connected versus disconnected regions.

3.4.1 Permeability for Regular and Irregular Media

In this section, all the computed permeability data is plotted in $\phi\theta$ space for regular and irregular media. Fig. 3.11a shows the computed permeability in regular media and Fig. 3.11b represents the permeability in irregular media. For better visualization, $\log_1 0$ of the permeability is plotted as background of percolation map. Dots show the (ϕ, θ) pair at which we ran the simulations. The white are shows "no percolation" zone and hatched area represents the porosity and dihedral angle range at which we expect the connectivity be a function of the history of the system. As can be seen, the permeability is mainly a function of porosity and the dependence of permeability to dihedral angle is very weak. This implies that although dihedral angle has a first order control on connectivity of the pore network, once the pore space is connected, the important properties affecting the permeability, i.e. tortuosity, are not strong function of dihedral angle. As soon as the pore space is connected, effect of dihedral angle would be limited to the surface areas (walls for fluid flow), which

are smaller for larger dihedral angles.

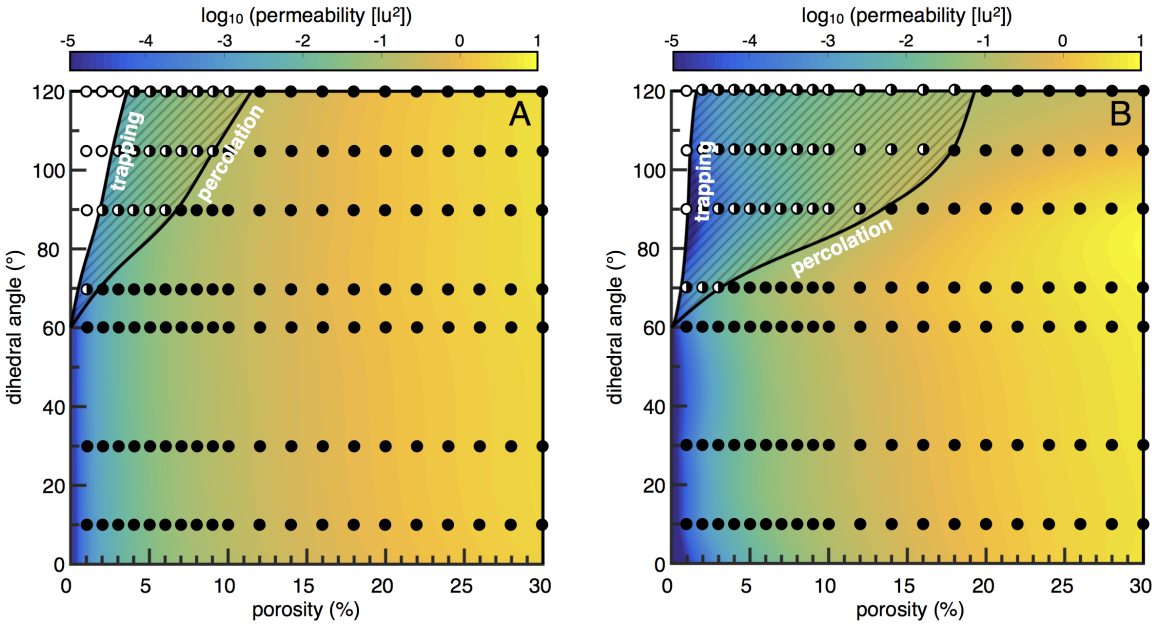


Figure 3.11: Revised percolation map in $\phi\theta$-space with permeability values as background. The calculated permeability of the texturally equilibrated melt network, in lattice units, is superimposed. Hatched area between trapping and percolation thresholds indicates the region where percolation via porous flow is possible due to hysteresis in melt connectivity once percolation threshold is reached. (a) polycrystalline material comprised of truncated octahedron grains, and (b) beta-titanium alloy comprised of realistic irregular grains.

Fig. 3.12 plots the computed permeability values with direct experimental measurements or LBM effective permeability of synthetic texturally equilibrated rocks. Fig. 3.12*a* shows the permeability values versus porosity on a log-log scale plot for different dihedral angles (see color bar) for the irregular medium. The permeability approximately shows a power law behavior. Fig. 3.12*b* collects all the permeability data for different dihedral angles for both regular and irregular media and a single power law curve is fittted to the data on a semi-log scale. Computed permeability values, which are obtained from computed pore networks, fall between data presented in literature. This provides a basic observation for the validity of the computed pore networks.

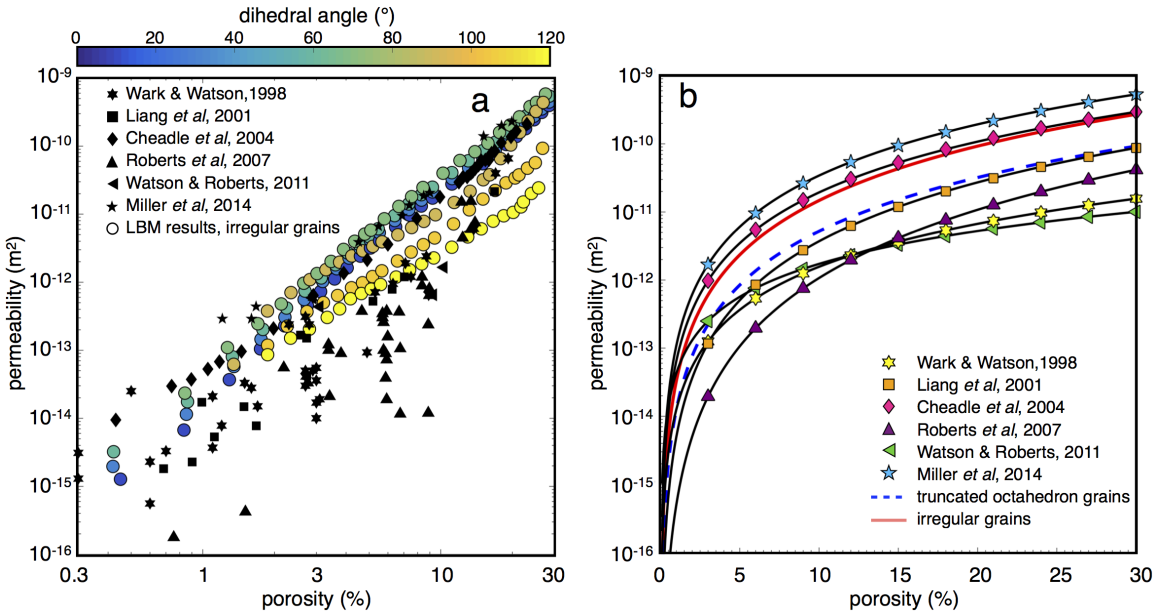


Figure 3.12: Comparison of computed permeability with experimentally measured and numerically calculated values. (a) permeability-porosity data on a log-log plot. The circle markers correspond to LBM results of computed pore networks using level set method in a polycrystalline material comprised of irregular grains. All other data are direct measurement or LBM computed permeability of synthetic texturally equilibrated rocks. (b) Power law fit of permeability shown in (a) on a semi-log plot. Permeability data are scaled to match the average grain size of 1 mm.

3.4.2 Permeability Anisotropy Due to Fabric Anisotropy

As mentioned in previous chapter, both channels and tabular pores coexist in anisotropic pore networks at small θ. Since most of the liquid resides in the tabular pores parallel to the direction of stretching, the channels perpendicular to the stretching are very thin at small ϕ (Fig. 2.14*a*). This suggests that strong anisotropies may exist in the physical properties of these pore networks. To test this hypothesis, the permeability of the texturally equilibrated pore networks has been calculated. Representative pore networks used in the computations are shown in Fig. 2.14. Both the vertical permeability, k_z, and the horizontal permeability, k_x, have been determined. Fig. 3.13 only reports results for $\theta \leq 60^\circ$ due to the lack of percolation at larger θ in anisotropic media.

Our simulations confirm that small amounts of anisotropy in the fabric of the porous media, $f \leq 3$, can induce dramatic permeability anisotropy, k_z/k_x. This anisotropy increases with f and is largest for small θ where the anisotropy in the fabric is amplified by two orders of magnitude (Fig. 3.13*b*-3.13*c*). Although both k_x and k_z increase monotonically with ϕ, the permeability anisotropy reaches a pronounced maximum near $\phi = 2\%$ for $\theta < 60^\circ$. This behavior can be understood in terms of the changes in the geometry of the pore network shown in the bottom row of Fig. 2.13. At low ϕ the connectivity in both directions is limited and the moderate permeability anisotropy is due to different length of the channels along the grain edges. As ϕ increases the liquid begins to wet the grain boundaries parallel to the direction of stretching. The accumulation of liquid on the grain boundaries drains the channels providing horizontal connectivity and the permeability anisotropy reaches a maximum. Further increase in ϕ will increase the diameter of the channels and increase horizontal permeability, which leads to decrease in k_z/k_x at larger ϕ. For $\theta \leq 60^\circ$, the absolute permeability in z-direction increases with ϕ, f, and θ.

3.5 Electrical Conductivity

Both computation of pore structure in texturally equilibrated materials and microtomographic images of porous rocks result in collection of discrete cubic grids in which each node represents a phase. In this part of project, a model has been developed based on works presented in Garboczi and Douglas (1996), Garboczi (1998) and Meille and Garboczi (2001) to solve the electrical conductivity problem in porous media. The theory behind this, is essentially the idea that a variational principle exists for the linear electrical conductivity problems (Milton et al., 2009). For a given porous media, subject to applied fields and certain boundary conditions, the voltage distribution is such that the total energy dissipated in the system is minimized (Gibiansky et al., 1999; Gibiansky and Mil-

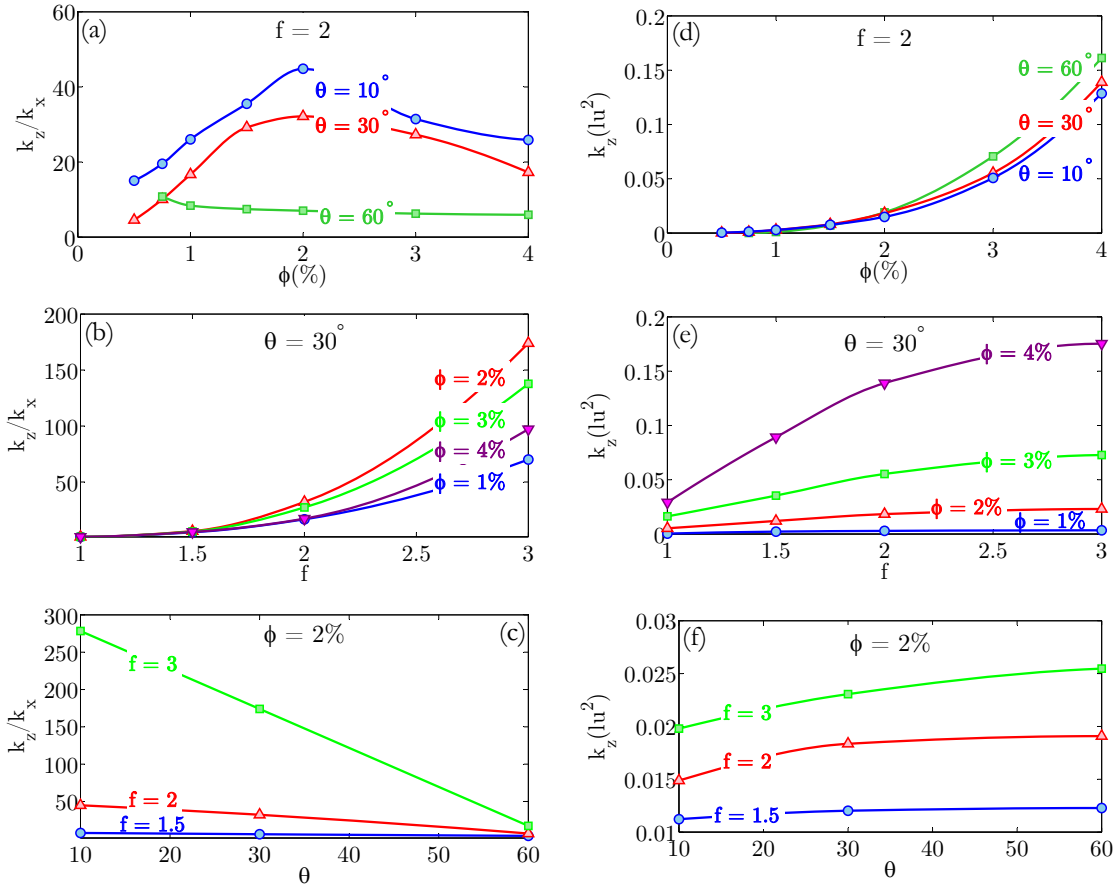


Figure 3.13: Development of permeability anisotropy of texturally equilibrated pore networks as function of ϕ, θ, and f. The top row shows the anisotropy, k_z/k_x. The bottom row gives the absolute value of the vertical permeability, k_z, in Lattice units $k[\text{lu}^2]$. In all cases, the polycrystalline material is stretched in the z-direction.

ton, 1993). In other words, the gradient of energy (U) with respect to the variables of the problem (voltage v) should be zero.

3.5.1 Methodology

In the developed model, each node attached to a corner of a grid (8 in 3-D, 4 in 2-D) has its own voltage. Total energy dissipated in system is sum of the energy in each grid, which in turn is a function of its nodal voltages as

$$U = \frac{1}{2} v_r D_{rs} v_s \tag{3.1}$$

where D_{rs} is stiffness matrix, a term originating in finite element treatments of linear elasticity problems. Assumption here is that each grid has equal lengths in all directions ($\Delta x = \Delta y = \Delta z$). Stiffness matrix of each grid can be represented as $D_{rs} = \left[n_{pr}^T \sigma_{pq} n_{qs} \right]$ in which n_{pr} is the derivative matrix linking nodal voltages to electric field and σ is conductivity matrix, which is symmetric and can be anisotropic. To apply periodic boundary condition, we can generally write $v_r = V_r + \delta_r$ where V_r is an 8-vector of the voltages at nodes, and δ_r is an 8-vector that corrects them to what they should be at boundaries. Substituting this general notation for voltages, energy for a given grid would be

$$U = \frac{1}{2} \left[V_r D_{rs} V_s + 2\delta_r D_{rs} V_s + \delta_r D_{rs} \delta_s \right] \tag{3.2}$$

which gives a term quadratic in the nodal voltages, a term linear in the nodal voltages, and a term constant with respect to the nodal voltages. This can be rewritten as

$$U = \frac{1}{2} V_r D_{rs} V_s + b_r V_r + C \quad \text{where} \quad b_s = \delta_r D_{rs}, \quad C = \delta_r D_{rs} \delta_s \tag{3.3}$$

Array b and scalar C are globally constant, and the only contributions to these come from grid points having nodes at boundaries and non-zero stiffness matrix. Once the energy equation is set up, all that remains is to find the set of voltages that minimize the electrical energy dissipated in system. Using above terminology, gradient of energy at each grid (m), which should be set to zero, can be represented as

$$\frac{\partial U}{\partial V_m} = A_{mn} v_n + b_m = 0 \tag{3.4}$$

The matrix A is of course built up from the individual D_{rs} matrices of the eight nodes that touch the grid labeled m. The matrix A is in principle large, but sparse. Coefficient matrix (A) and constant matrix (b) can easily be evaluated using Eq. 3.3 and the system of linear algebraic equation $A\mathbf{v} = -b$ can be set up to solve for extremum of energy. This system of equation is solved using Biconjugate gradient stabilized method. Once the voltages in each pixel are obtained, the average current density $\langle j_p \rangle$ can be calculated and from that we can get effective conductivity of medium

$$\langle j_p \rangle = \langle \sigma_{pq} e_q \rangle \equiv \sigma_{pq}^{\text{eff}} E_q \quad \text{where} \quad E_p = \langle e_p(\vec{r}) \rangle = \frac{1}{V} \int d^3 r e_p(\vec{r}) \tag{3.5}$$

3.5.2 Regular Media

Here we present the effective conductivity of texturally equilibrated pore networks in a regular media comprised of truncated octahedron grains. After obtaining the effective conductivity, formation factor is calculated using a special form of Archie's law for single phase fluid in the pore space

$$F = \frac{\sigma_w}{\sigma_{\text{eff}}} = \frac{a}{\phi^m} \tag{3.6}$$

and then is plotted versus the porosity on a log-log scale. Fig. 3.14 presents the value of Formation factors for different dihedral angles. As expected, the cementation factor increases with dihedral angle due to increase in tortuosity of the electrical current path. A linear function is then fitted on the log-log plot and the values of cementation factor (m) and tortuosity factor (a) are obtained. Table 3.1 summarizes the values of the cementation and tortuosity factors.

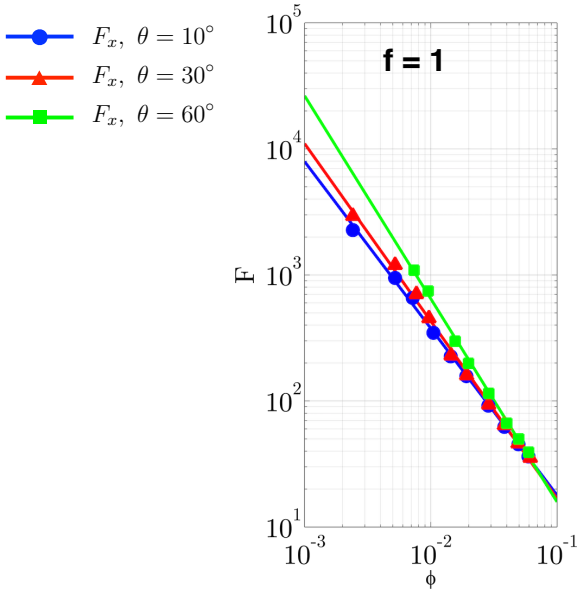


Figure 3.14: Formation factor versus porosity and dihedral angle for regular media with isotropic grains.

3.5.3 Effect of Anisotropy

Fig. 3.15 represent the formation factor values versus porosity of medium, for different dihedral angles and elongation (anisotropy) factors. Table 3.1 collects the cementation exponent for above cases. As can be seen, in all cases, increase of dihedral angle will result in increase of cementation exponent. The larger dihedral angle, means the fluid tends less to wet the grain boundaries, and stays in channels on grain edges. That means the path for electrical conductivity is more tortuous and in result, m increases. Except in the symmetric case, for all anisotropic crystal structures cementation factor in z-direction (m_z) is smaller than x-direction (m_x) for corresponding θ and f values. That also comes from the fact that tortuosity is smaller in z-direction due to elongation of crystals and also existence of planar features. Also for same value of θ, increase in elongation factor increases the m_x and decreases the m_z. That is again due to change in tortuosity of the path that the electric current needs to take in order to get to other side of sample.

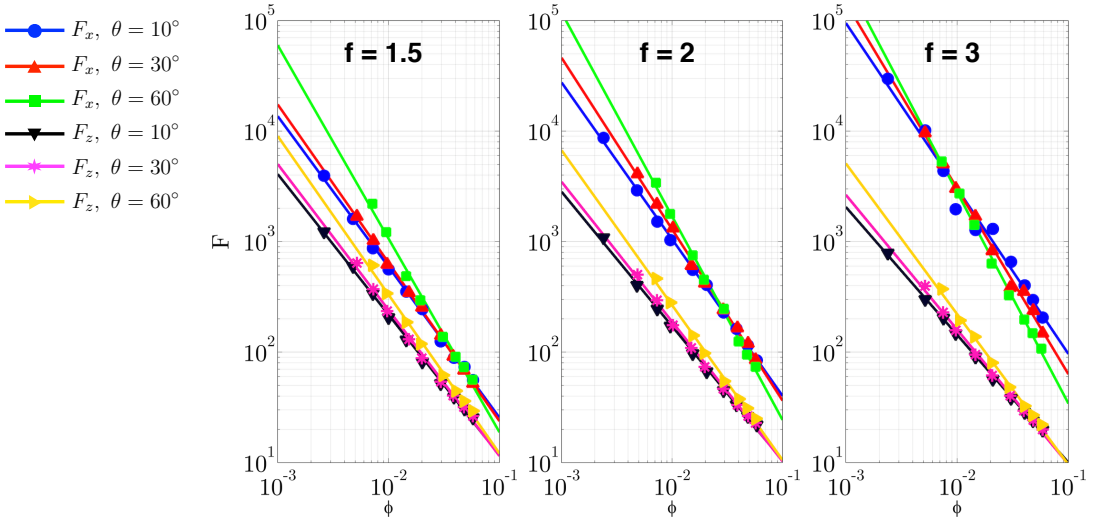


Figure 3.15: Formation factor versus porosity and dihedral angle for different elongation (anisotropy) factors.

Table 3.1: Cementation exponent in x- and z-direction for different dihedral angles and elongation (anisotropy) factors.

	m_x				m_z			
	f = 1	f = 1.5	f = 2	f = 3	f = 1	f = 1.5	f = 2	f = 3
$\theta = 10°$	1.32	1.36	1.42	1.5	1.32	1.27	1.22	1.16
$\theta = 30°$	1.41	1.43	1.55	1.68	1.41	1.32	1.26	1.22
$\theta = 60°$	1.61	1.75	1.84	1.91	1.61	1.44	1.4	1.36

Chapter 4

Pore-Scale Experimental Study of Rock Salt

4.1 Background and Literature Review

Several research studies have been performed on texturally equilibrated materials and their properties. Most of the works are done on partially molten systems while a few are available on behavior of rock salt. Similarity in crystallographic structure of salt and partially molten rock would allow us to use available knowledge in that field. This section presents a summary of the literature review of sealing capacity of rock salt.

Until recent years, rock salt has been considered to be impermeable as it seems to contains and keep gas inclusions for long time. Increasing energy demand and necessity of producing hydrocarbon reservoir enclosed or touched by salt deposit and also urgent need of safe repository sites for high-level nuclear waste have brought attention to research and study the porosity and permeability of natural rock salt. Low water content of rock salt and low permeability to brine were always considered to be the reason that salt domes can exist in sedimentary records.

Rock salt in sedimentary basins has long been considered to be impermeable and provides a seal for hydrocarbon accumulations in geological structures (Downey, 1984; Stewart, 2007). The low permeability of rock salt also has the potential to isolate nuclear waste from ambient groundwater and may provide a suitable deep geological waste repository (Hansen and Leigh, 2011; Noseck et al., 2015). This option is currently being reconsidered in the United States after the closure of the Yucca mountain repository in Nevada (Hansen and Leigh, 2011). However, field observations of oil impregnated rock salt

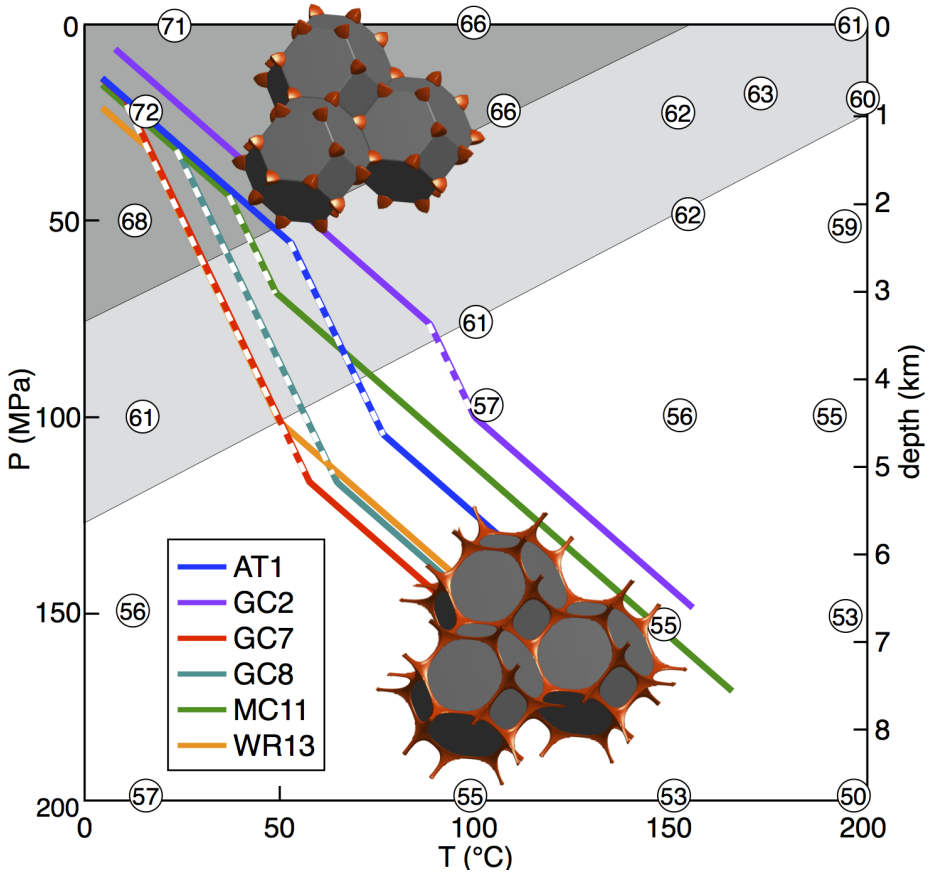


Figure 4.1: Brine percolation in rock salt. PT-trajectories of multiple sub-salt petroleum wells are shown together with experimentally measured dihedral angles, θ, for the salt-brine system Lewis and Holness (1996). The static theory predicts that fluid must overcome a percolation threshold in the gray area, whereas fluids are predicted to percolate at any porosity in the white area. The light gray area highlights the transition zone, $60° < \theta < 65°$, between percolating and disconnected pore space Lewis and Holness (1996). The segment of each well that is located within the salt has a lower geothermal gradient due to the high conductivity of salt and is shown as a dashed line. The depth axis is only for illustration and assumes an overburden with constant density, $\rho = 2300$ kg/m^3.

(Schoenherr et al., 2007), geochemical evidence for the replacement of the in-situ brines (Land et al., 1988), as well as the drainage of brine from mining induced fractures and dilatant microcracking (Hansen and Leigh, 2011; Davison, 2009) demonstrate that the permeability of natural rock salt may not be negligible.

Brine-filled pore networks in rock salt approach textural equilibrium due to fast reaction kinetics of salt dissolution and re-precipitation (Lewis and Holness, 1996). Percolation in these networks is controlled by the dihedral angle, θ, at the solid-solid-liquid triple junctions

$$\theta = 2\cos^{-1}\left(\gamma_{ss}/(2\gamma_{sl})\right), \tag{4.1}$$

where γ_{ss} and γ_{sl} are the solid-solid and solid-liquid surface energies (Ghanbarzadeh et al., 2015b; von Bargen and Waff, 1986; Wark and Watson, 1998; Ghanbarzadeh et al., 2014). The dihedral angle is therefore a thermodynamic property that changes with pressure, P, and temperature, T. The static pore-scale theory shows that texturally equilibrated pore networks percolate at any porosity if $\theta \leq 60^\circ$, while a finite porosity is required for percolation if $\theta > 60^\circ$ (von Bargen and Waff, 1986; Wark and Watson, 1998; Ghanbarzadeh et al., 2014).

The experimentally measured θ in salt-brine systems decreases with both increasing P and T (Fig. 4.1), suggesting that fluids at shallow depth must overcome a percolation threshold whereas fluids at greater depth are likely to percolate at any porosity. The PT-trajectory of multiple petroleum wells in the Gulf of Mexico crosses this transition and therefore provides an opportunity to test the static pore-scale theory in a realistic field setting (See Chapter 5).

4.2 Methodology

4.2.1 Undrained Hydrostatic Experiments

A set of experiments was devised to investigate the connectivity of texturally equilibrated brine in synthetic salt samples at different PT conditions. Previous experimental work (Lewis and Holness, 1996) on salt-brine systems mapped the variation of the dihedral

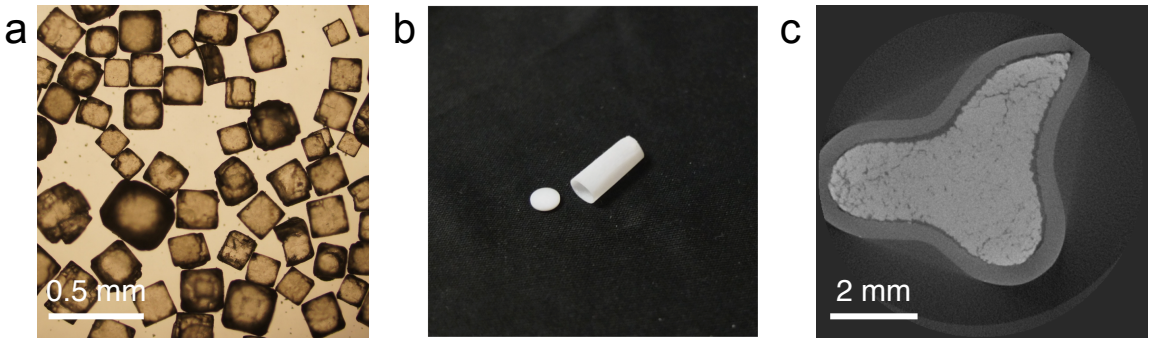


Figure 4.2: Experimental materials. (a) Reflected light microscopy image of the initial cubic halite grains. (b) A Teflon capsule with outer diameter of 5 mm, used as container for the salt sample. (c) A cross-section of the deformed salt sample inside Teflon capsule with the resolution of 8 μm.

angle with P and T, but did not provide the 3D reconstruction of pore network topology required to detect percolation. This study utilizes non-destructive X-ray microtomography to image the salt samples. Analytical grade halite (99.9% pure) with dimensions of 0.2-0.4 mm was used in all samples (Fig. 4.2*a*). For each experiment, about 150 mg of halite and 7-15 mg of distilled water were placed into a Teflon capsule and covered with a Teflon lid (Fig. 4.2*b*), which was then positioned inside a platinum tube (5 mm outer diameter). The platinum tube and the Teflon capsule were weighed, the platinum tube was then welded shut on both ends, and weighed again, to ensure that no sample was lost.

The samples were held at temperatures from 100°C to 275°C, and pressures from 20 MPa to 100 MPa in externally heated, cold-seal pressure vessels for 120 hours to achieve textural equilibrium. The use of pressure vessels made of a Ni-based alloy and Ni filler rods inside the vessels ensured that the experiments were run at an oxygen fugacity approximately equal to the Ni-NiO buffer reaction. To avoid any re-equilibration, samples were then quenched to room temperature in less than one minute by removing the pressure vessel from the furnace and immersing it in water. The samples were immediately removed from the pressure vessel, weighed to ensure that they did not leak, and then the platinum

tube was peeled away from the Teflon capsule. The samples were left in their Teflon capsules (Fig. 4.2*c*), and then scanned by non-destructive X-ray microtomography within 24 hours of being quenched.

4.2.2 Pore-Scale Imaging

The salt samples were imaged using a Zeiss (formerly Xradia) microXCT 400 scanner at UTCT, the University of Texas High-Resolution X-ray Computed Tomography Facility (Ghanbarzadeh et al., 2015a). This scanner uses cone-beam geometry to gather up to 2000 slices in a single rotation. For these acquisitions the camera was binned to acquire 928 slices per scan. The scan volumes are equant, thus the distance across the field of view is roughly equivalent to the vertical throw of the scan and the voxels are cubic, measuring 1.1 m on a side. The scans were acquired using the 20X objective detector, with the X-ray source set to 120 kV and 10 W with no beam filtering. A total of 1441 views, at 22 seconds per view, were acquired over 180 degrees of rotation. The scan volumes were reconstructed as stacks of 16-bit TIFF images with byte scaling set to [-50, 2000] and a smoothing kernel of 0.7 applied. Ring artifacts were minimized by dithering the sample stage during data acquisition, and also in some cases by the post-acquisition application of a secondary reference image taken through a CaF_2 filter with an attenuation value similar to the sample.

The salt samples were fragile and thus had to be imaged inside their Teflon capsule. Because the capsule also attenuates X-rays, it added some image noise, which was partly ameliorated with longer scanning durations. Reviews and details on important parameters and considerations for X-ray microtomography imaging of porous media and naturally deformed rock salt are available in (Wildenschild and Sheppard, 2013; Thiemeyer et al., 2015), respectively.

4.2.3 Image Analysis

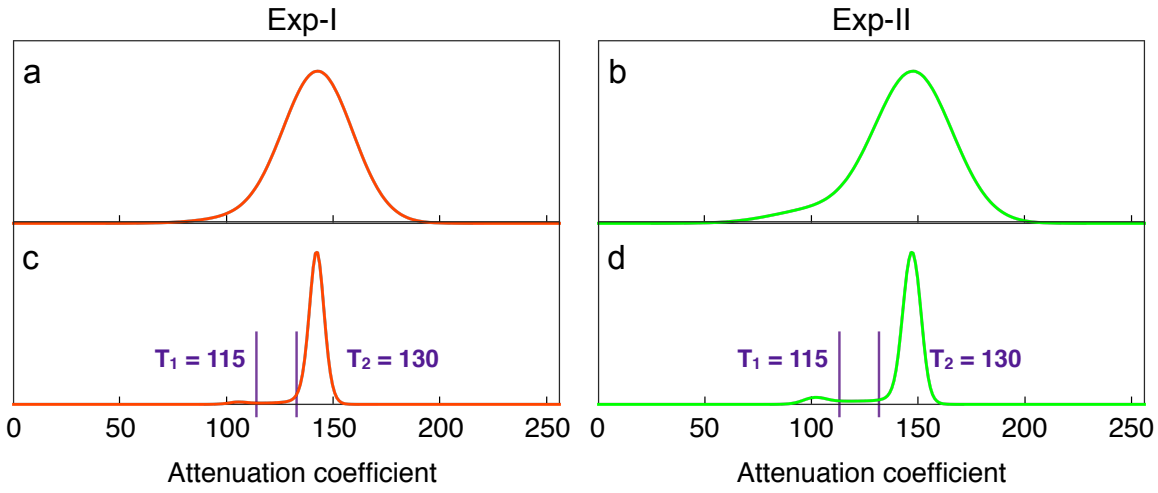


Figure 4.3: Experiments on synthetic rock salt have been performed at P = 20 MPa and T = 100°C (Exp-I) and P = 100 MPa and T = 275°C (Exp-II). Histogram of attenuation coefficient obtained from raw 3D image data of (a) Exp-I and (b) Exp-II. Histogram after applying the 2D anisotropic diffusion as grayscale filter on image data of (c) Exp-I and (d) Exp-II.

Analysis of X-ray image data has become an integral component of pore scale investigations of porous media (Wildenschild and Sheppard, 2013). Image analysis consists of reducing the noise level from grayscale image data, converting grayscale image data into segmented images, filtering the segmented data, quantification and post processing (Ghanbarzadeh et al., 2010a,b; Hanafizadeh et al., 2011a,b; Ghanbarzadeh et al., 2012). Although filtering affects the original image information (Fig. 4.3*a* and 4.3*b*), sophisticated noise reduction methods are necessary in order for quantification algorithms to work correctly. Grayscale image data filtering was performed using 2D anisotropic diffusion (Perona and Malik, 1990) which identifies the edges in the image, then smooths the dataset along those edges, while doing minimal smoothing across them. Grayscale filtering was done using open source software ImageJ (Schneider et al., 2012).

We used the indicator kriging thresholding (Oh and Lindquist, 1999) to perform

segmentation in this study. This method requires input of two thresholds, T_1 and T_2. Voxels with grayscale values below T_1 and above T_2 are assumed pore and solid phase, respectively. For all the voxels with values between these thresholds, the method determines the probability of belonging to either phase utilizing a local two-point correlation function. The local, 3D data-informed adaptivity of the threshold makes the indicator kriging one of the best performing segmentation methods for porous media. The indicator kriging segmentation is implemented in 3DMA-Rock software package (Lindquist et al., 2005). The threshold values chosen for both Exp-I and Exp-II are 115 and 130 (Fig. 4.3*c* and 4.3*d*).

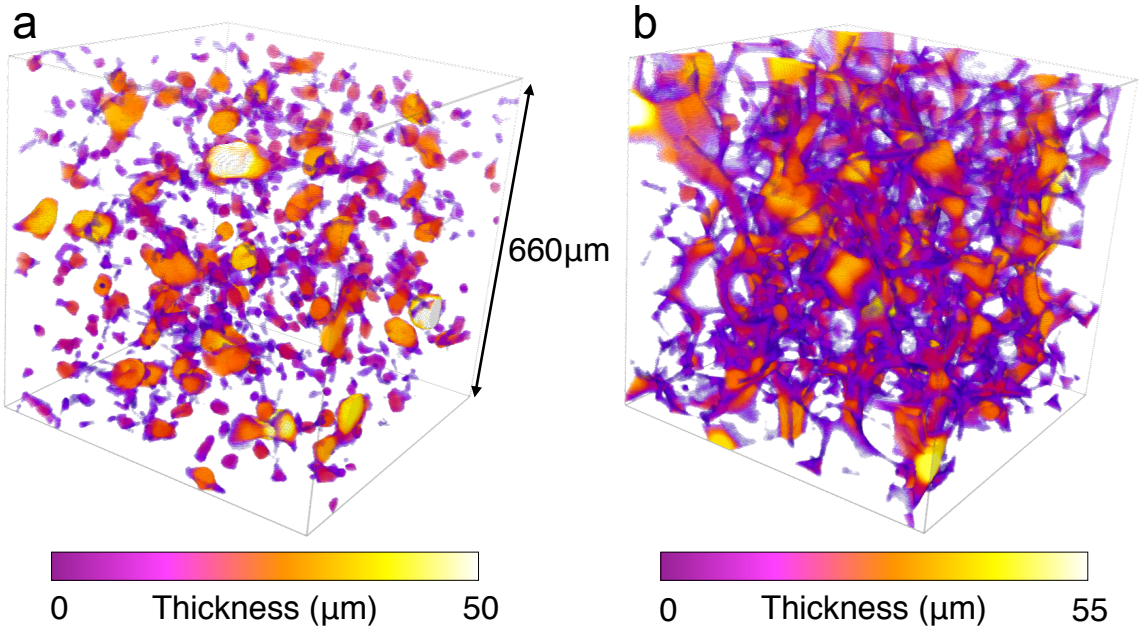


Figure 4.4: Experiments on synthetic rock salt have been performed at P = 20 MPa and T = 100°C (Exp-I) and P = 100 MPa and T = 275°C (Exp-II). Pore space inscribed radius map. The pore space is colored by the thickness of the pore bodies and pore throats. (a) Exp-I, (b) Exp-II.

Medial axis representation of the pore space reduces the complexity but preserves the topology (Lindquist et al., 1996). The medial axis representation of a digitized object is a 26-connected centrally-located skeleton of the void space. This enables effective visualization of the pathways and connections in 3D and provides search tool to find throats and

subsequently the pore-throat network. Here we used the 3DMA-Rock software package to skeletonize the pore space and quantify the brine connectivity in salt samples. The results of this analysis (Fig. 4.7*c* and 4.7*d*) show the pore space is disconnected in Exp-I and demonstrates that the pore space in Exp-II is connected. We should note that independent of the imaging method and resolution and image processing techniques, there is always the possibility of resolving features that introduce errors in final results. In Exp-I the average diameter of the pore bodies is 25 voxels and hence well resolved. We see no evidence for connecting pore throats along the grain edges. Theoretically, the connecting channels at high dihedral angle (if they exist) should be more compact and hence easier to detect in image data (Fig. 4.4*a*). In Exp-II the average diameter of the pore bodies are 15 voxels and the channels are approximately 10 voxels across which is sufficient to detect connectivity (Fig. 4.4*b*).

Pore space topology and connectivity can be further quantified by the pore coordination number distribution. The coordination number, z, of a pore is the number of neighboring pore bodies in the constructed pore-throat network from segmented image data (Lindquist et al., 2005). A porous media with the mean coordination number of $3.5 < \bar{z} < 4.5$ is considered very well connected while a porous material is characterized as non-percolating when $\bar{z} \approx 1$ (Raoof and Hassanizadeh, 2009). The disconnected pore space in Exp-I is indicated by coordination numbers of 1 for majority of nodes, while the connectivity of the pore space in Exp-II is implied by the most abundant coordination numbers of 3 and 4 (Fig. 4.5).

4.2.4 Dihedral Angle Measurement from Images

In order to test the static pore-scale theory and to compare the experimental results with previous works, the dihedral angles in the experiments are determined from 2D sec-

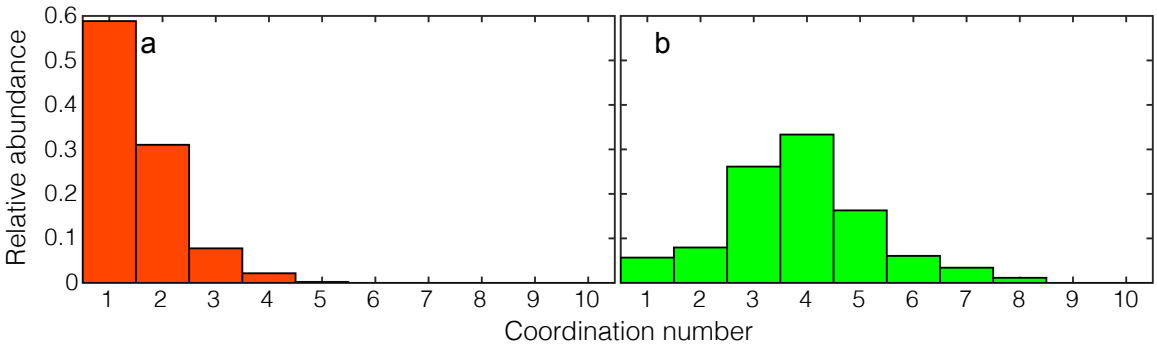


Figure 4.5: Experiments on synthetic rock salt have been performed at P = 20 MPa and T = 100°C (Exp-I) and P = 100 MPa and T = 275°C (Exp-II). Coordination number of skeletonized pore space. The distributions of coordination numbers for (a) Exp-I and (b) Exp-II.

tions. The first step is to establish corner points from segmented image data, which is done using the Harris algorithm implemented in the MATLAB Image Processing Toolbox (red dots in Fig. 4.6*a* and 4.6*b*). Then the normals in a 3×3 neighborhood of each corner on the solid-liquid edges are calculated and divided to two groups of opposing directions (light and dark blue vectors in Fig. 4.6). The apparent dihedral angle is then given by the angle between the mean vectors of light and dark blue vector groups. To have a better representation of the apparent dihedral angle, this procedure is performed with slicing the 3D image data in x, y and z planes. The method is fast and unbiased and produces a distribution of apparent angles where the true dihedral angle is approximated by the median of the distribution (Jurewicz and Watson, 1985). The distribution of the apparent dihedral angles has a median of $65^\circ \pm 5^\circ$ for Exp-I and a median of $52^\circ \pm 6^\circ$ for Exp-II (Fig. 4.7*e*).

4.3 Results and Discussion

We present the results of two representative experiments in Fig. 4.7, performed at $P = 20$ MPa and $T = 100$°C (Exp-I) and $P = 100$ MPa and $T = 275$°C (Exp-II). The 3D reconstruction (Fig. 4.7*a* and 4.7*b*) and medial axis representation of the pore space

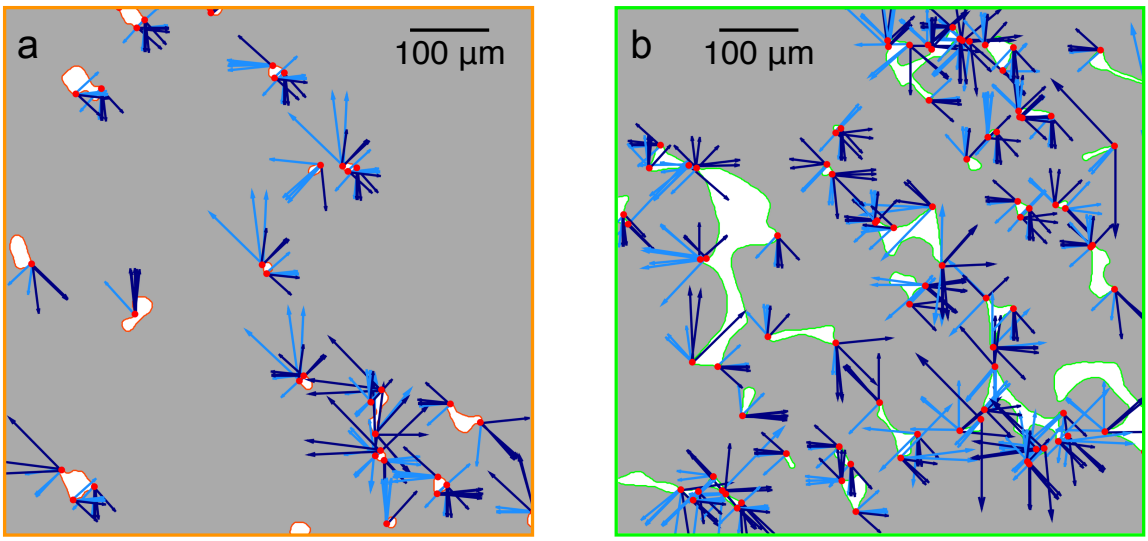


Figure 4.6: Experiments on synthetic rock salt have been performed at P = 20 MPa and T = 100°C (Exp-I) and P = 100 MPa and T = 275°C (Exp-II). Dihedral angle measurement. The slices are selected from middle of samples in (a) Exp-I and (b) Exp-II.

(Fig. 4.7*c* and 4.7*d*) show the brine network is disconnected in Exp-I and is connected in Exp-II. This is confirmed by statistical analysis of the coordination number distributions that show almost all nodes in Exp-I have coordination number 1 whereas the coordination numbers of 3 and 4 are most abundant in Exp-II (Fig. 4.5). The distribution of the apparent dihedral angles has a median of $67° \pm 5°$ for Exp-I and $52° \pm 6°$ for Exp-II (Fig. 4.7*e*). Distributions with a single narrow peak, as well as similarity to previously reported values of dihedral angle (Lewis and Holness, 1996), indicate that the experiments are approaching textural equilibrium. Comparison of experiments with the regime diagram for fluid percolation show that static pore-scale theory successfully predicts the connectivity of the pore space (Fig. 4.7*f*).

The experimental results confirm the static pore-scale theory in undrained laboratory experiments on synthetic salt samples that have been imaged with non-destructive X-ray microtomography after quenching to ambient conditions. These results confirm the first-order control of the dihedral angle on brine percolation and serve as a baseline for the

field observations of fluid distributions in deformed rock salt.

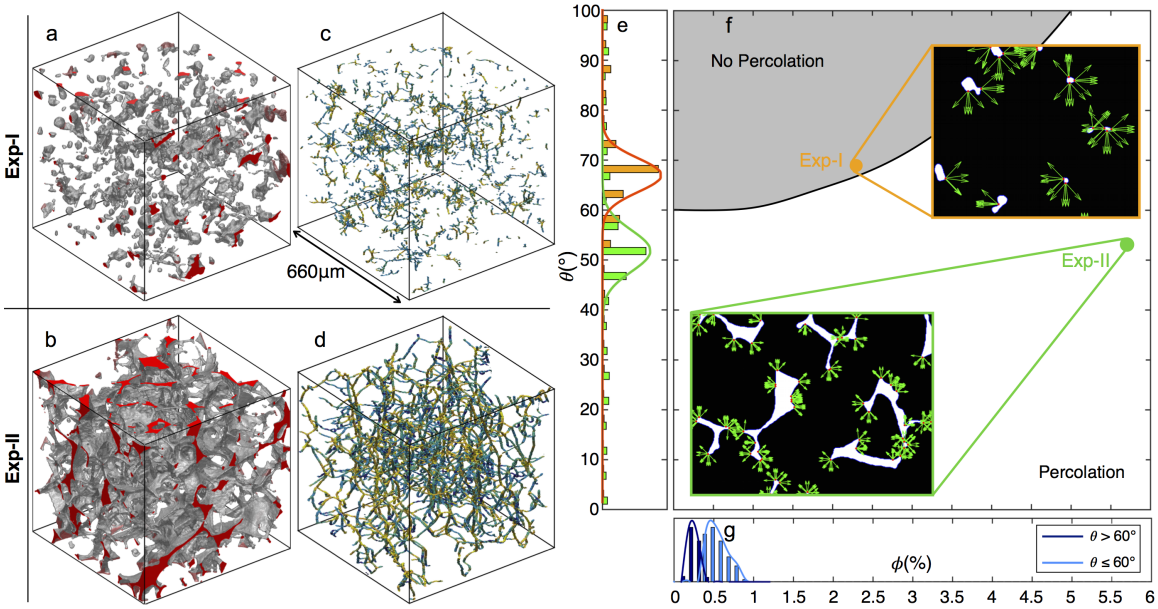


Figure 4.7: Hydrostatic experiments on synthetic rock salt have been performed at P = 20 MPa and T = 100°C (Exp-I) and P = 100 MPa and T = 275°C (Exp-II). (a, b) 3D reconstruction of the pore network at textural equilibrium, all edges of the 3D volumes correspond to 660 μm. (c, d) The skeletonized pore network extracted from the reconstructed 3D volume; colored according to local pore space inscribed radius, with warmer colors indicating larger radius. (e) Distribution of apparent dihedral angles in the experiments. (f) Exp-I and Exp-II in the $\theta\phi$ space regime diagram with the percolation threshold obtained from the static pore-scale theory (von Bargen and Waff, 1986; Ghanbarzadeh et al., 2014). Inserted images show the details of automated dihedral angle extraction from 2D images. We report the median value of dihedral angles and the estimated errors based on the 95% confidence interval. (g) Porosity of natural rock salt inferred from resistivity logs (Fig. 5.5*b*).

Chapter 5

Fluid Percolation in Ductile Rock Salt in Gulf of Mexico

5.1 Background and Literature Review

Three key properties of rock salt make it the best seal for hydrocarbon accumulations (Schoenherr et al., 2007; Downey, 1984). First, as rock salt behaves like a fluid with high viscosity under pressure, minimum principal stress (σ_3) is very close to maximum principal stress (σ_1). Thus fracturing in a highly pressurized salt is an effect of an almost-isotropic stress state, as opposed to situation typical in shale layers. (Hildenbrand and Urai, 2003). Second, the porosity and permeability of uncompacted rock salt drop rapidly with burial process even at a very shallow depth (i.e. depth $<$ 70 m, (Casas and Lowenstein, 1989)). Third, rock salt deforms plastically in nature making it more resistant to fluid penetration (Popp et al., 2001).

Downey (1984) ranked the geological seals as following (first is the most reliable and strongest seal): salt, anhydrite, kerogen-rich shale, clay shale, silty shales, carbonate mudstone and chert. Experimental measurements of rock salt porosity and permeability at range of 0.1% to 1.4% and 10^{-9} to 10^{-6}D (Bredehoeft, 1988), respectively. Nevertheless, salt, like any other rock, can lose its sealing capacity under particular conditions, but the theoretical reasons for the seal loss of rock salt are not well understood. Diverse pieces of evidence indicate that the permeability of rock salt may increase once burial becomes sufficiently deep (Lewis and Holness, 1996). Several studies show that brine inclusions in natural halite are extremely common (i.e. (Lewis and Holness, 1996)), and salt may have abundant brine seeps (Land et al., 1988).

Two processes are known to increase permeability in rock salts. The first is the formation of topologically connected pores and tubes on grain edges due to changes in interfacial tension between brine and rock salt with increasing p-T (Lewis and Holness, 1996). The change in the ratio of salt-salt and salt-brine interfacial tension changes the water-halite dihedral angle. In cases with dihedral anglegreater than 60°, pore fluids are distributed in small pockets where four grains meet. On the other hand, when the dihedral angle is less than 60°, brine imbibes in salt wets the crystal edges. Lewis and Holness (1996) showed that even at low porosities, permeability of salt can increases to the order of $10^{-16}m^2$ due to the change in the dihedral angle subject to p-T conditions at depths greater than 3km. This means a 5 order of magnitude increase in permeability compared to reported values (Bredehoeft, 1988).

The second is micro-cracking and associated dilation due to dynamic recrystallization of rock salt (Peach and Spiers, 1996). Sufficient confining pressure can enhance dynamic recrystallization because of increased potential for micro-cracking and grain boundary disruption (Peach et al., 2001). As a result, the permeability of the rock salt can increase by up to six order of magnitude in higher confining stress states, possibly affecting the trap integrity of the salt basin (Schoenherr et al., 2007). As Fig. 5.1a shows, the dilatant behavior of rock salt can be detected as a region in the confining-differential stress plane (Peach et al., 2001; Popp et al., 2001). Also a rapid increase in fluid pressure and low effective stress during the burial process of rock salt can result in hydro-fracturing and loss of sealing capacity (Peach and Spiers, 1996).

Commercial interest in the large hydrocarbon accumulations below extensive bodies of allochthonous salt in the deep water Gulf of Mexico provides an opportunity to test the static pore-scale theory in slowly moving natural rock salt. In order to do so, we studied field data from the salt section of 48 wells crossing the predicted transition zone from

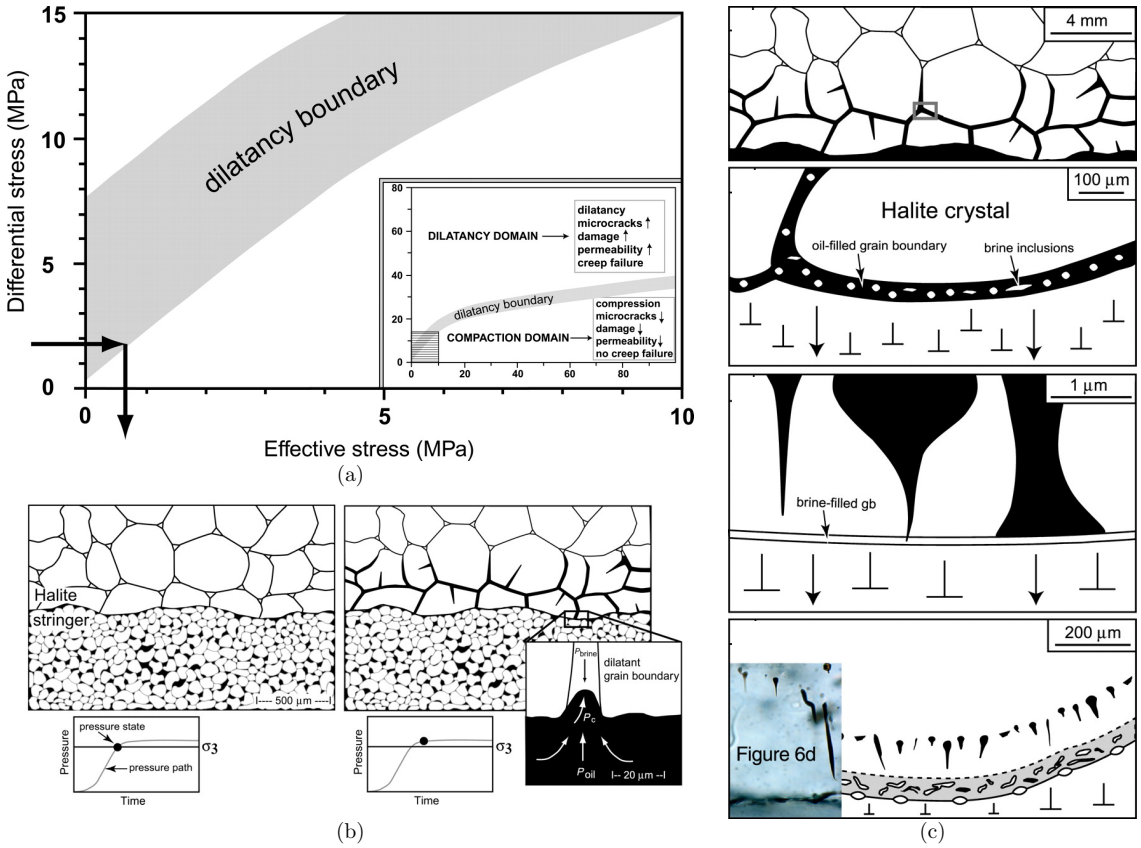


Figure 5.1: (a) Dilatancy boundary in effective vs. differential stress plane for rock salt, reproduced after Popp et al. (2001), (b) Diagram of the mechanism of diffuse dilatancy of the Ara Salt from Schoenherr et al. (2007), (c) A schematic image sequence to show the possible evolution of the observed microstructures during dynamic recrystallization of halite from (Schoenherr et al., 2007).

disconnected to percolating pore space (Fig. 4.1) to constrain the brine and hydrocarbon connectivity.

5.2 Methodology

Typically no intact core is recovered from the salt section of wells and the available data set consist of wireline well logs and mud logs. Wireline well logs, obtained by lowering a measurement tool into the well, characterize different properties of the formation

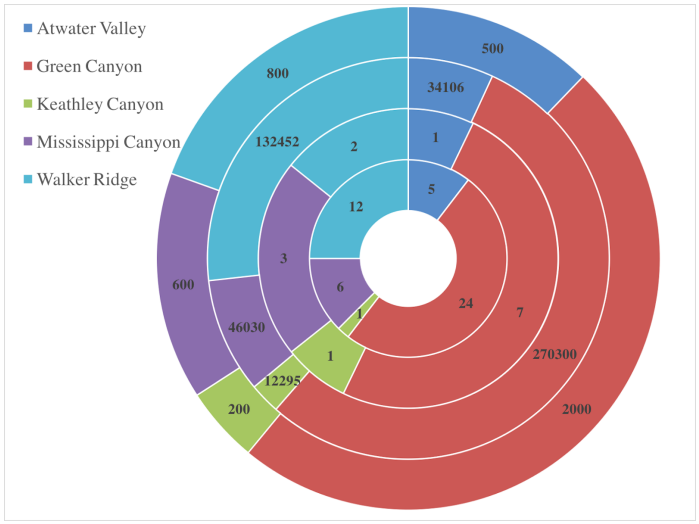
rock and fluids. Mud logs, which record the hydrocarbon gas content and observations from the drill cuttings brought to the surface, provide direct constraints on the presence of hydrocarbons in salt. Hydrocarbon signs reported in mud logs include fluorescence, oil staining, oil cut and dead oil embedded in the salt.

5.2.1 Dataset and Well Locations

In this work, 48 wells from 14 subsalt prospects in deep water Gulf of Mexico have been chosen as case studies. This provides an extensive and comprehensive dataset that covers more than 490,000 ft of salt and studies fluid distribution in more than 4100 salt samples. Fig. 5.2 summarizes the number of prospects, wells, cumulative salt thickness and number of salt samples in different protraction areas in Gulf of Mexico. Geographically distribution and the size of the dataset is sufficient for a meaningful conclusion based on the field data analysis. In this work, gamma ray, deep resistivity, gas chromatography data (C_1-C_5) and mud log sample descriptions are plotted and compared with each other to evaluate the sealing capacity.

5.2.2 Well Logs

Well logging is the process of recording various physical, chemical, electrical, or other properties of the rock and fluid mixtures penetrated by drilling a borehole into the earth's crust. Wireline logging is performed by lowering a set of measurement tools into the well while the mud logging is based on visual and technical inspection of samples brought to the surface during the drilling process. A common combination of wireline logs includes gamma ray, resistivity and neutron porosity. Wellbore stability due to salt creep raises safety and environmental concerns for lowering neutron porosity tools in salt section of the wells, therefore this data is not recorded in salt intervals. The mud log normally includes real-time drilling parameters, total gas hydrocarbons log, gas chromatography log



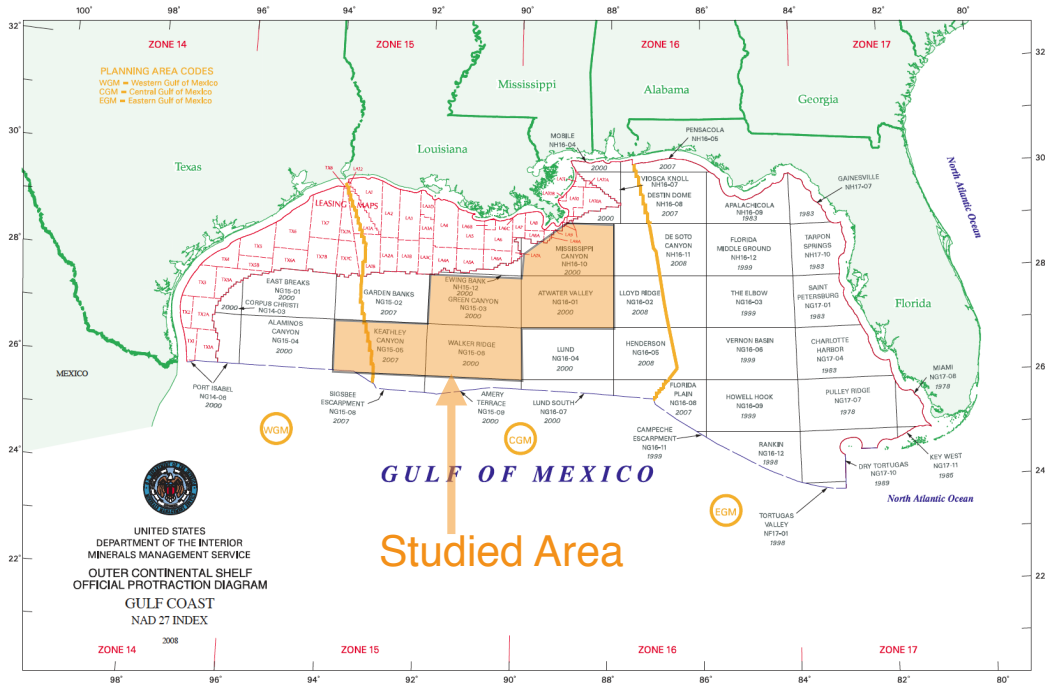


Figure 5.2: (a) Distribution of the data including number of wells(inner shell), the number of prospects (second shell from inside), cumulative salt thickness (third shell from inside) and number of salt samples (outer shell) in each protraction area. (b) Studied protraction areas are highlighted with orange color.

and lithology description.

Gamma ray log measures the natural radioactivity of the formation along the bore-

hole, in API units, which is mainly used for lithology characterization. Salt bodies in the Gulf of Mexico are almost pure sodium salts with minimal rock and sediment inclusions, which in turn show a very low gamma ray response (Fig. 3A). Resistivity logging measures the formation electrical resistivity, in ohm.m, with different depths of penetration into the rocks around the borehole. This information helps to characterize the connectivity of the pore fluid, calculate the porosity of the formation, differentiate between formations filled with either brine or hydrocarbons and compute the water saturation. One of the most widely used and firmly accepted methods to calculate porosity from resistivity logs is the Archie's law (Fig. 2G). Assuming the brine is the only connected phase in the pore space, the porosity, ϕ, correlates with the resistivity measurements:

$$\phi = \left(\frac{aR_w}{R_o} \right)^{\frac{1}{m}} \tag{5.1}$$

where R_w is the formation brine resistivity, R_o is the measured formation resistivity, m is the formation cementation exponent and a is the tortuosity factor. Typical values of the constants for rock salt are $m = 2$ and $a = 1$ (Yaramanci and Flach, 1992) and is evaluated as fully saturated brine in downhole conditions.

The total gas log measures the total amount of combustible hydrocarbon gas extracted from the drilling fluid, in the unit of total methane equivalents. The recorded values are usually scaled with an arbitrary gas unit, depending on gas-detector manufactures, and practical importance is on the relative changes in the amount of formation gas (Bourgoyne Jr et al., 1986). Gas chromatography log is the most widely used technique to detect and quantify the amount of light hydrocarbons present in the formation. The instrument separates hydrocarbon gas components from a mixture and reports the amount of methane, ethane, propane, butane and pentane in particle per million.

5.2.3 Residual Oil Formation

In order to provide a physical mechanism for the formation of dead oil observed in mud logs, we have numerically simulated the brine-oil flow within the texturally equilibrated pore network obtained from Exp-II. The simulations are done using an immiscible and capillarity controlled fluid displacement method (Prodanovic and Bryant, 2006). Initially, the interconnected pore network of the sample is saturated with brine. Hydrocarbons are introduced into the salt during a drainage process, which generates oil-impregnated salt. Then the spontaneous imbibition of the brine into the pore space, displaces the non-wetting hydrocarbons. The non-wetting phase, hydrocarbons, are then trapped in small pores with brine films around them (Fig. 5.3). This illustrates how oil phase can become trapped as disconnected residual oil by subsequent imbibition of the brine into the pore network. Therefore, we conclude that the presence of hydrocarbons requires that a connected pore space must have existed at least during the time interval when the hydrocarbons entered the rock salt. Compaction may eliminate porosity later, but the electrical resistivity log suggests that the brine has remained connected in the regions that contain trapped hydrocarbons.

5.2.4 Conversion of Depth to Dihedral Angle

To facilitate the comparison with the static pore-scale theory, the depth is converted to the dihedral angle. The experimentally measured values of dihedral angle in salt-brine system, Fig.4.1, provide a two-variable relationship with P and T. Here we have used the MATLAB Curve Fitting Toolbox to interpolate the data with the thin-plate spline method, which fits smooth surfaces and is exact at the input data points (contours in Fig. 5.4c are interpolation results). We calculated the PT trajectory of each well using the normal geopressure and geothermal gradients in Gulf of Mexico, with consideration of water column

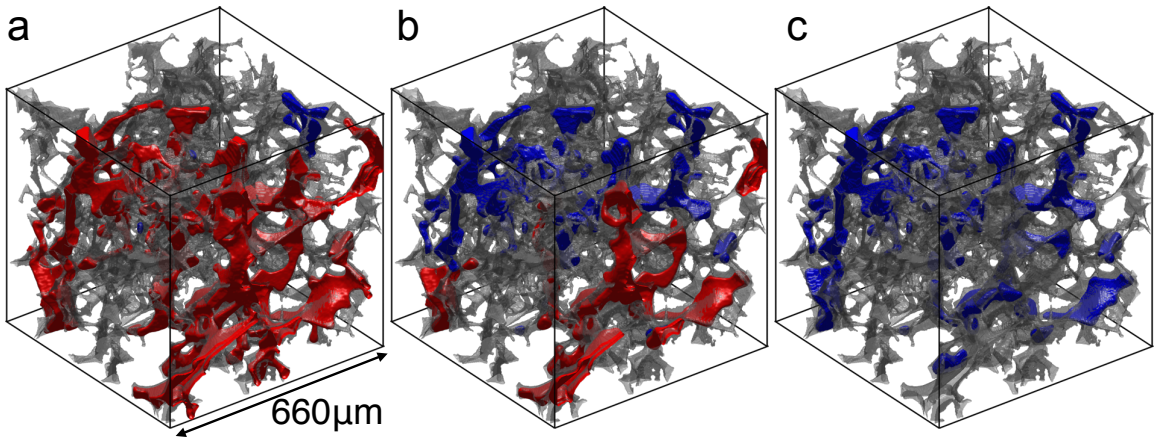


Figure 5.3: Formation of hydrocarbons residual in salt. Numerical simulations of oil phase configurations during the displacement of oil by brine in texturally equilibrated pore space. The connected mobile oil is shown in red and the disconnected (trapped) oil is shown in blue. Pore-grain surface is shown in transparent gray and water occupies the pore space where there is no visible oil. The trapped hydrocarbon saturation values are: (a) $S_{HC}^{tr} = 1.1\%$, (b) $S_{HC}^{tr} = 16.5\%$, (s) $S_{HC}^{tr} = 27.8\%$.

pressure and the seabed temperature (Forrest et al., 2005)

$$P(z) = PG_w z_{sb} + PG_s(z - z_{sb}) + PG_h(z - z_{sb} - z_s) \tag{5.2}$$

$$T(z) = T_{sb} + TG_s(z - z_{sb}) + TG_h(z - z_{sb} - z_s) \tag{5.3}$$

where the subscripts w, s, h and sb denote water, sediments above salt section, halite and seabed, respectively. Normal pressure gradient in each section is represented by PG, normal temperature gradient by TG and z is the depth below sea level. For example, the P and T profiles in well GC8 are calculated considering water depth of 1360 m below sea level and temperature of 4.4°C (Fig. 5.4*a* and 5.4*b*) on the seabed. Typical values of pressure gradients (PG) and temperature gradients (TG) are: $PG_w = 10075$ Pa/m, $PG_s = 22570$ Pa/m, $PG_h = 21190$ Pa/m, $TG_s = 0.0255$°C/m and $TG_h = 0.010$°C/m. Then the PT data are converted to dihedral angle by evaluating the fitting function (Fig. 5.4*c*).

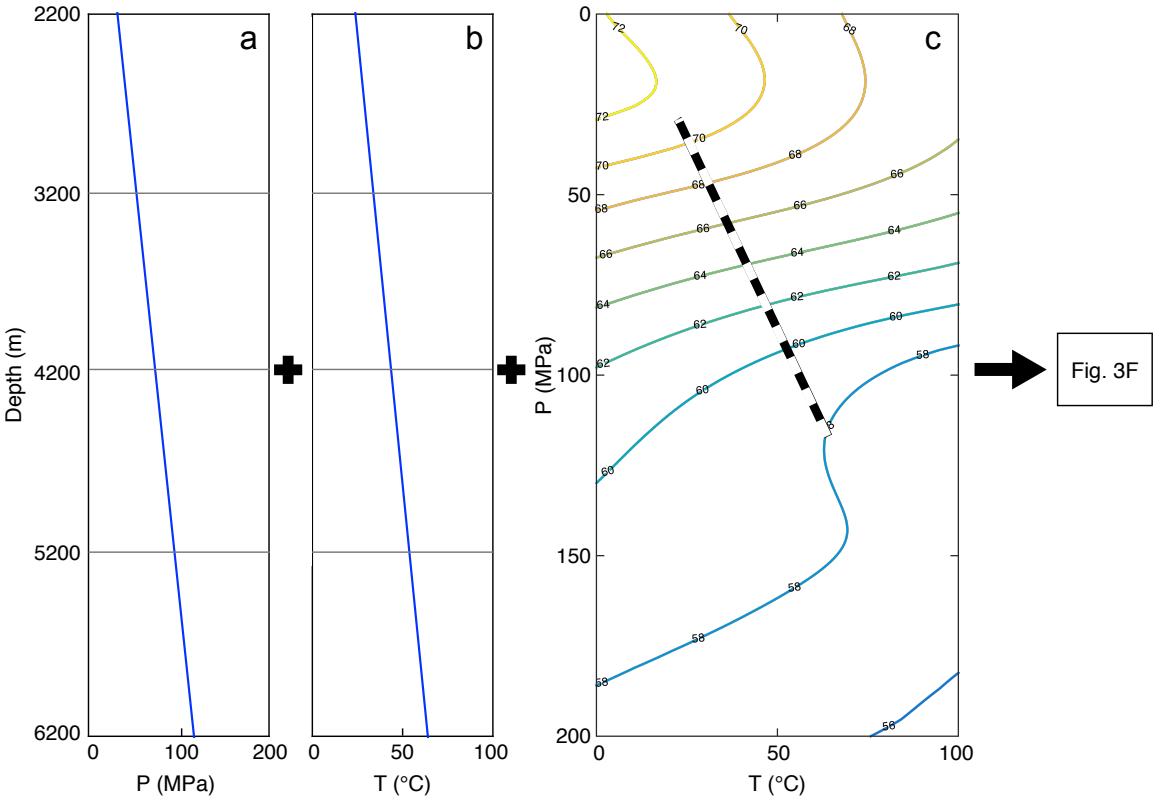


Figure 5.4: Conversion of depth to dihedral angle. Normal geopressure and geothermal gradients are used to calculate (a) the pressure and (b) temperature profile in well GC8 (see Fig. 4.1). (c) The PT trajectory of well GC8 on contour map of the dihedral angle interpolated from experimental data (Fig. 4.1).

5.3 Results and Discussion

5.3.1 Example of Studied Wells

We chose only those salt sections for analysis that were free of other rock fragments, as indicated by low values of naturally occurring gamma radiation (Fig. 5.5*a*). In contrast to the uniform gamma ray signature, all other logs (Fig. 5.5*b*-5.5*e*) show a distinct change in the bottom third of the salt. The very high electrical resistivity in the upper two-thirds of the salt section implies that the conductive brine is not connected (Fig. 5.5*b*, Watanabe and Peach (2002)). In this region, the porosity calculated from Archie's law is below 0.4% (Fig. 4.7*g*). The reduction of electrical resistivity by an order of magnitude in the bottom

third suggests that brine is connected at porosities below 0.8% (Fig. 4.7g). The salt-brine dihedral angle inferred from the PT-trajectory of the well (Fig. 5.4) drops below $60°$ in the bottom third of the salt (Fig. 5.5f), consistent with the static pore-scale theory.

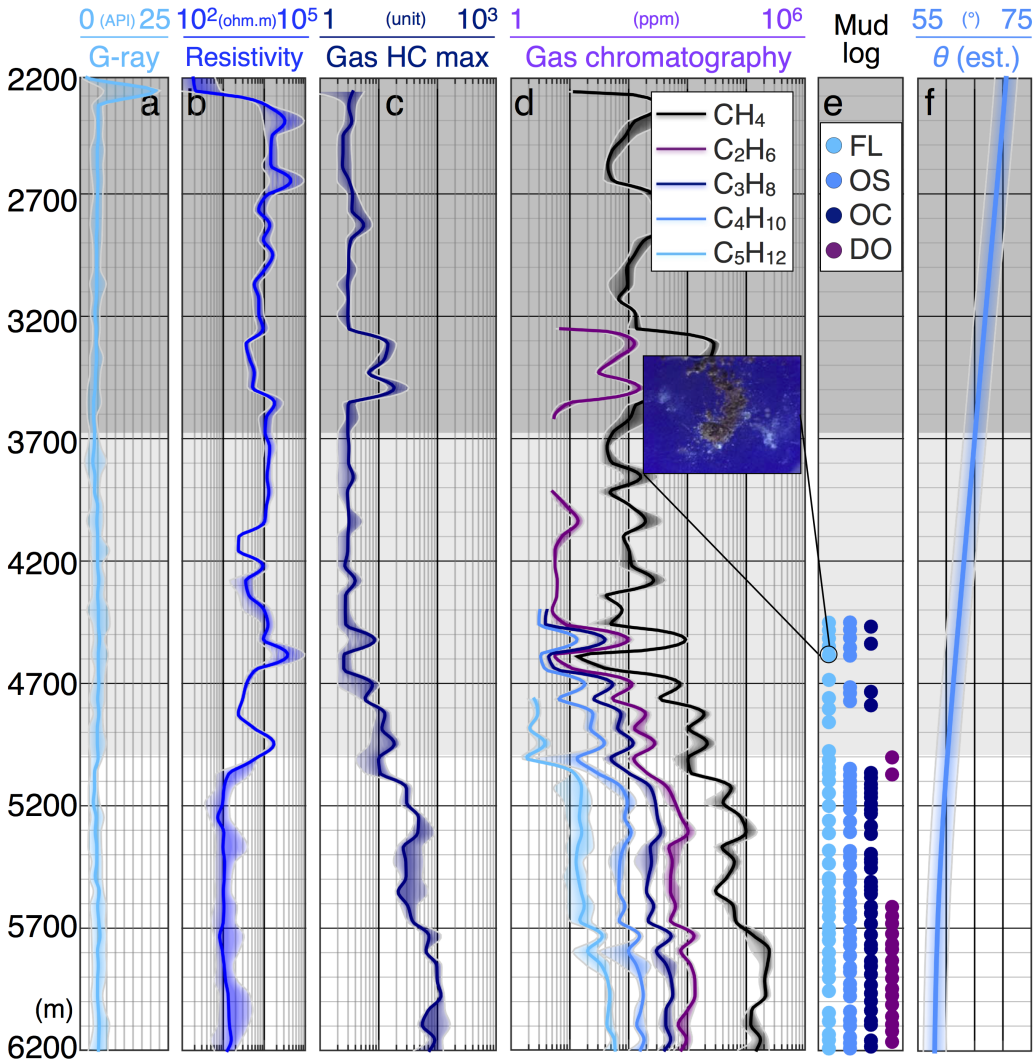


Figure 5.5: Petrophysical observations in salt. Wireline well logs and mud logs data constraining the fluid distribution and connectivity in the well GC8 from the deep water Gulf of Mexico. (a) Gamma-ray log, (b) electrical resistivity, (c) total hydrocarbons gas, (d) gas chromatography, (e) hydrocarbon signs (FL: fluorescence, OS: oil stain, DO: dead oil and OC: oil cut) in mud logs and (f) the dihedral angle inferred from experimental data. Shading around each curve shows the measurement error and average fluctuations in data. The gray background corresponds to shaded areas in experimental data (Fig. 4.1).

In addition to a connected brine phase, the total gas hydrocarbons and gas chromatography logs, indicate a substantial increase in the amount of natural gas in the lower third of the salt (Fig. 5.5*c* and 5.5*d*). We observe this general pattern also in the mud logs that contain no indications of hydrocarbons in the top two-thirds, but show multiple signs of hydrocarbons in the bottom third (Fig. 5.5*e*). In the presence of brine, hydrocarbons are the non-wetting phase, so that the textural equilibration of the pore network occurs through brine mediated dissolution and re-precipitation of the salt. The dihedral angle of the brine-salt system governs the connectivity of pore space, consistent with observations in wireline well logs and mud logs. Once hydrocarbons overcome the capillary entry pressure (Schoenherr et al., 2007), they can enter the salt in regions where the brine network is connected. Subsequent imbibition of the brine can trap the hydrocarbons in the pore space (Fig. 5.3). The presence of hydrocarbons therefore indicates that a connected pore space existed during the entry of the hydrocarbons into the rock salt. This interpretation is consistent with previous work reporting direct observations of oil stained salt cores recovered from conditions where $\theta < 60^{\circ}$ (Schoenherr et al., 2007).

5.3.2 Embedded Hydrocarbons in Salt

Due to space limitation and repetitive nature of the data analysis in other wells, we avoid presenting the well logs of other 47 wells. Instead, here we present some statistics on the entry and presence of the hydrocarbons in to the salt section of the all other wells. For better visualization and simplification of the problem, we group spatially associated wells (prospects) to look at the distribution of hydrocarbons in salt sections. Fig. 5.6 presents the vertical extent (true vertical depth) of the hydrocarbons by gas, fluorescence, oil staining and oil cut along with the vertical extension of the salt in the same prospects. As can be seen, in almost all the cases, there is a considerable amount of hydrocarbons embedded in salt. This also is confirmed by percentage of the salt sample showing fluorescence, oil

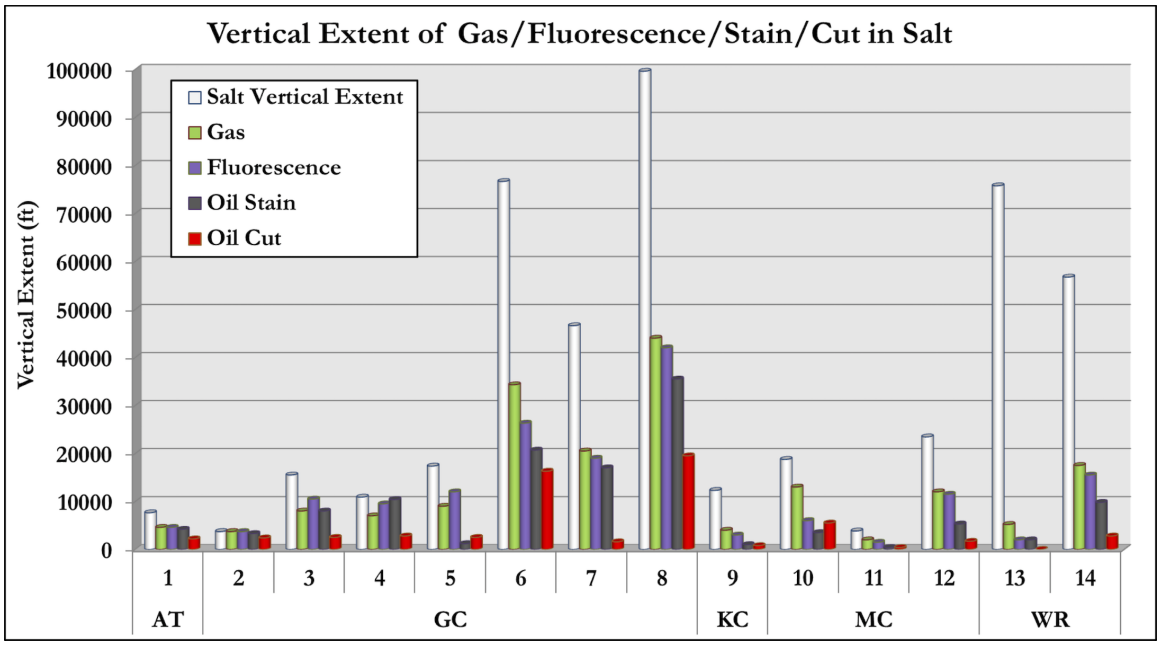


Figure 5.6: Cumulative vertical extent of salt, gas, fluorescence, oil staining, and oil cut in each well group or protraction area.

staining, oil cut and dead oil (Fig. 5.7).

Nonetheless, we should note that not all the evidence of percolating pore space in salt agrees with static pore scale theory and static experiments. In fact from 48 wells studied in this work, ≈60% of them agree with experimental work (Lewis and Holness, 1996), 10% disagree and the rest, ≈30% do not provide enough evidence to decide whether the brine network in salt is percolating or disconnected. There are several reasons that we see such a deviation from expected theoretical analysis. In the group of wells that do not proved enough evidence, the hydrocarbons source is very from the bottom of the salt. Therefore, if we do not observe hydrocarbons in salt, it does not mean that the pore network in salt is disconnected. Furthermore, rubble zone at the bottom of the salt may provide additional sealing that may not allow entry of hydrocarbons in a connected brine network. In addition to this group of wells (≈30% of wells), almost 10% of them do not agree with theory. This mean that the entire salt section is located in the "no percolation"

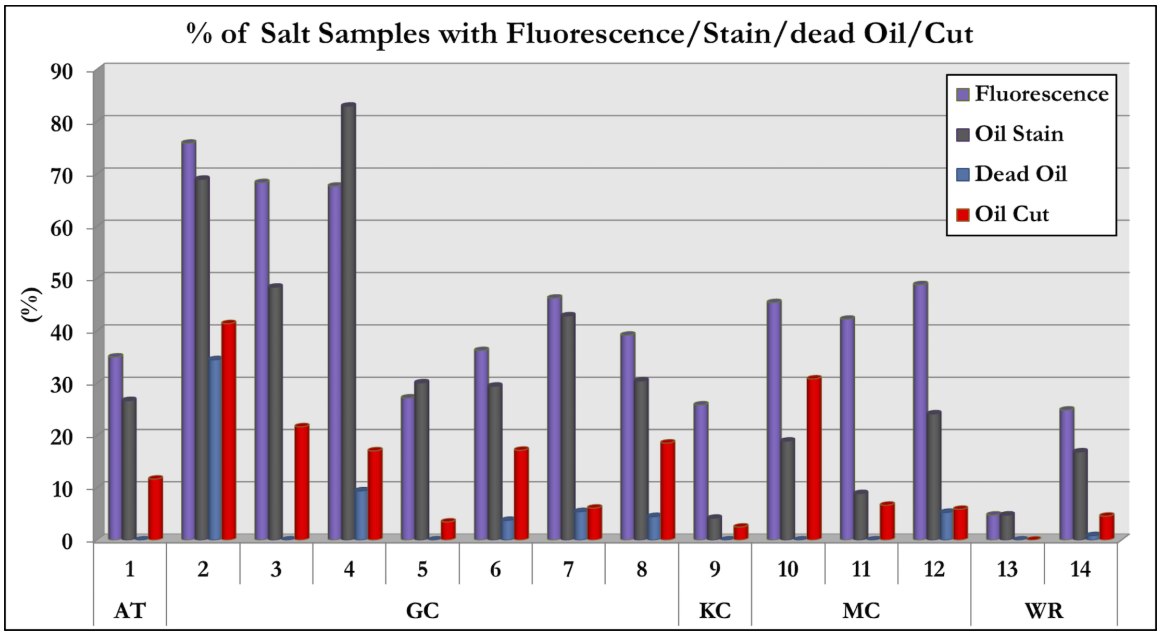


Figure 5.7: Percentage of the salt samples showing fluorescence, oil staining, oil cut and dead oil in in each well group or protraction area.

gray area, but we see many hydrocarbon signs in the salt. Below, we study the effects of the rubble zone thickness, distance between hydrocarbons source and bottom of the salt, and also deformation-assisted fluid percolation, on fluid distribution in a deforming and dynamic rock salt.

5.3.3 Rubble Zone

Rubble zone is comprised of salt, shale, sandstone, limestone, clay and any neighboring sediments immediately adjacent to the base of salt (Saleh et al., 2013). Rubble zone is highly disturbed and mixed due to salt movement, resulting in extensive fracturing (Saleh et al., 2013). The zone is characterized by highly sheared sediments that can be at or near the pore-collapse state (Saleh et al., 2013). Therefore, sediments in rubble zone are usually undergone shear compaction and have much less permeability that the background sediment. As a results, the salt dome or body may be surrounded by a rubble zone layer

that can add to the sealing capacity at the bottom of salt.

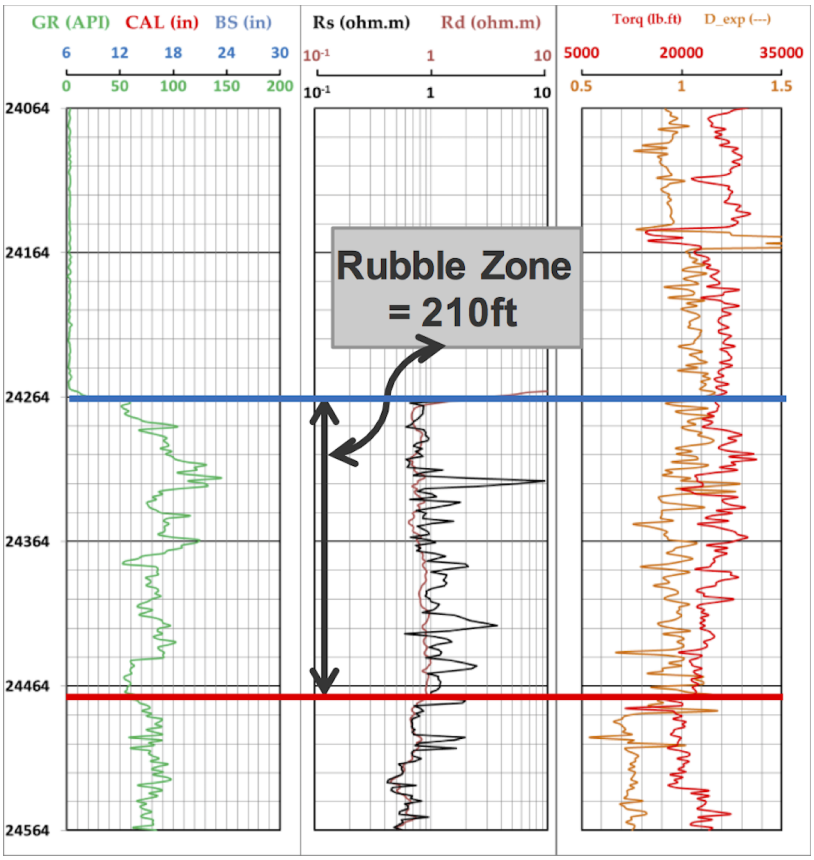


Figure 5.8: Diagnosis of the rubble zone with gamma ray, resistivity and drilling logs. The rubble thickness at the salt exit is 210 ft.

In order to diagnose the rubble at the bottom of salt, we look at the logs at the salt exit. Saleh et al. (2013) showed that any discordance or disagreement between the gamma ray log and the resistivity log might be a sign of the rubble zone at the base of salt. Below the rubble zone, gamma ray and resistivity logs should track each other. There are also other well logs that can help the diagnosis of the rubble zone including caliper log, sonic and density logs. In this study, we also have used the drilling log, torque and d-exponent to characterize a layer below the salt that is behaving differently from the background sediment. Fig. 5.8 shows the gamma ray, resistivity and drilling logs of a well at the salt

exit. A 210 ft thick rubble zone can be detected in this well.

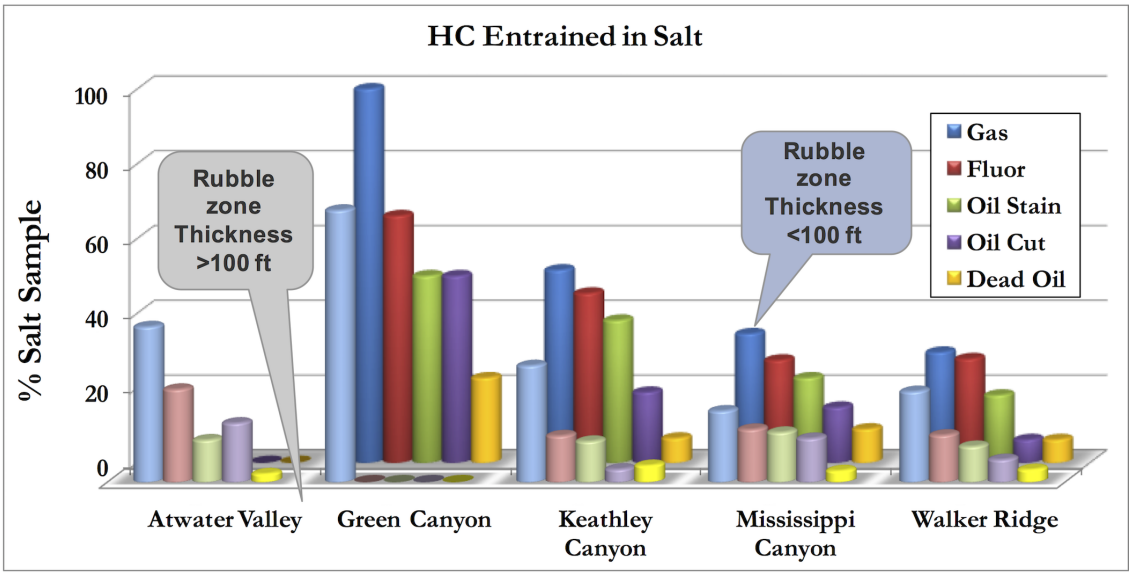


Figure 5.9: Distribution of the hydrocarbon signs in salt for two groups of wells. The group one, back row, has thin rubble zone and group two, front row, has thick rubble zone.

As mentioned, rubble zone has a lower permeability due to shear compaction and also pore-collapse state. This can provide an additional seal at the base of salt. In order to examine this hypothesis, we divided the wells into two group: Group one has a thin rubble zone, with thickness less than 100 ft. This number is the average thickness of the rubble zone in all 48 wells. Wells in group two, however, have thick rubble zones, thicker than 100 ft. The percentage of the salt samples that show hydrocarbons signs are plotted for this two groups in different protraction areas (Fig. 5.9). As can be seen, the wells in group two, with thick rubble zones, have a lot less hydrocarbons entrained into the salt. This shows that the rubble zone adds to the sealing capacity at the base of the salt.

5.3.4 Distance Between Hydrocarbon Source and Base of Salt

The other important factor that affects the distribution of hydrocarbons in salt is the distance between the hydrocarbons source and the base of the salt. If the hydrocarbon bearing layer (or layers) are very close to the base of salt, we can expect the entry of the oil and gas into connected pore space in salt. The farther the hydrocarbon bearing layer is from base of the salt, the harder it would be for the oil and gas to migrate through the other sediments and therefore, the less hydrocarbon we might see in salt. Fig. 5.10 shows the percentage of the salt samples with oil stain on vertical axis and the distance between hydrocarbon bearing layer and the base of the salt on horizontal axis. As expected, we see an exponential decay in the amount of hydrocarbons embedded in salt as the this distance increases.

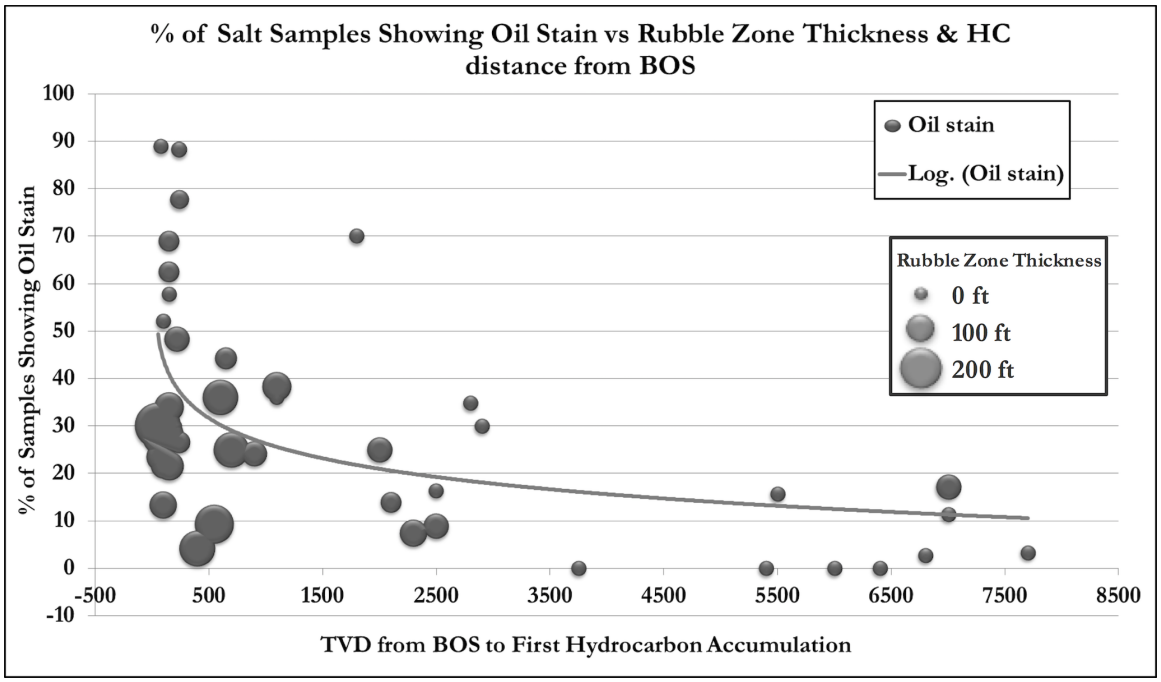


Figure 5.10: Percentage of the salt samples with embedded hydrocarbons (oil staining in this graph) as a function of the true vertical distance between the base of salt and first hydrocarbon bearing layer below the salt.

The size of the dots in the Fig. 5.10 is proportional to the thickness of the rubble zone at the corresponding well. As illustrated in the figure, most of the wells with thick rubble zone fall below the average line. This, again, explains the fact that the rubble zone adds to the sealing capacity at the base of salt.

5.3.5 Deformation Assisted Fluid Percolation

High quality resistivity logs (Fig. 5.5*b*) are only available in 2 wells due to technical difficulties and lack of commercial interest in the salt section of wells. Therefore, we rely on the logs that detect hydrocarbons to infer the connectivity of the brine in the remaining 46 wells. We group spatially associated wells to look at the distribution of hydrocarbons in salt sections (Fig. 5.11). The abundance of hydrocarbons is affected by the distance of the nearest hydrocarbon source from the bottom of the salt. For example, the first oil source is more than 2,000 m below the base of salt in the wells of group WR13, justifying the sparsity of hydrocarbon signs (see section 5.3.4).

We convert the depth to dihedral angle using available experimental data (Figs. 4.1 and 5.4). All the wells we considered show signs of connected pore space at depths where the dihedral angle is below 60°, except the shallow wells of group MC11. Using the two electrical resistivity logs and Archie's law, we estimate that the porosity of these connected regions are less than 1% (Fig. 4.7*g*). This provides direct field evidence that dihedral angles below 60° allow the percolation of texturally equilibrated pore networks at porosities below the transport limit in more typical porous media that originated as clastic sediments (van der Marck, 1999).

Nonetheless, field data also shows evidence of percolating pore space at shallower depths where the dihedral angle is substantially above 60° (Fig. 5.11). Under these conditions the porosity must increase above a threshold to allow percolation. Static pore-scale

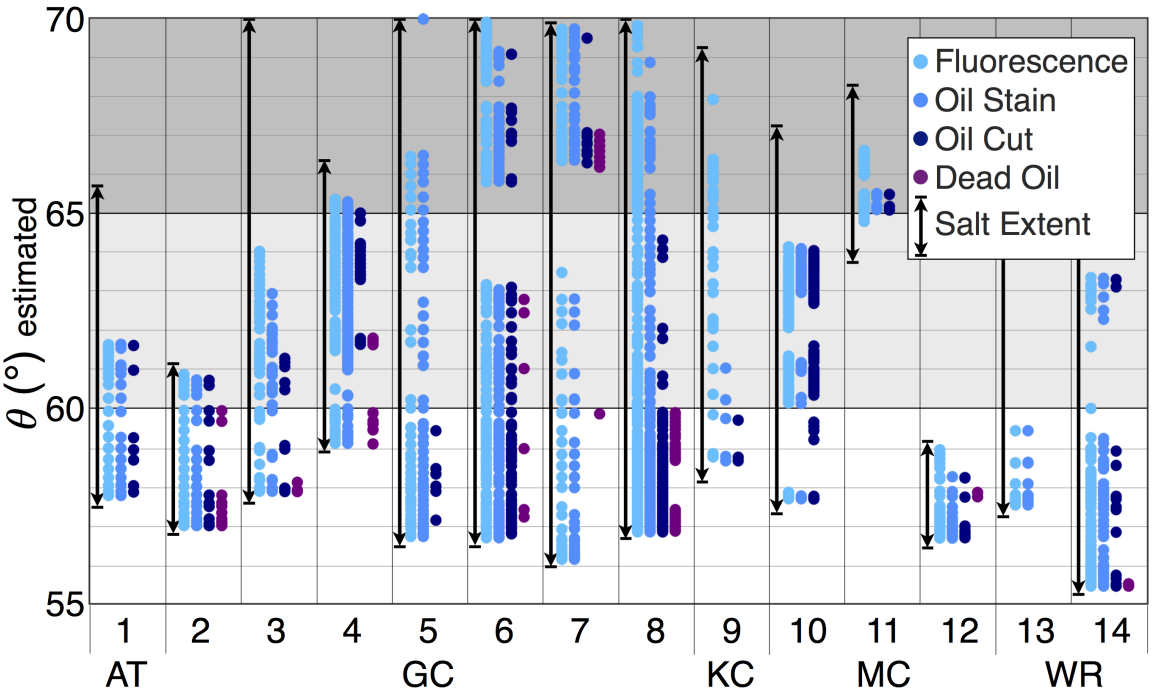


Figure 5.11: Fluid distributions in salt wells. Hydrocarbons signs from mud logs of all 48 wells covering 150,000 m of salt are shown as function of dihedral angle. Wells are divided into 14 groups based on spatial proximity. Salt extent is shown by arrow in each region. Theoretical fluid connectivity is indicated by gray scale (Fig. 4.1). Abbreviations denote the following protraction areas in Gulf of Mexico: AT: Atwater Valley, GC: Green Canyon, KC: Keathley Canyon, MC: Mississippi Canyon and WR: Walker Ridge.

theory requires porosities between 2-3% to allow percolation at dihedral angles between 65° and 70° (Fig. 4.7*f*). However, none of the porosities inferred from the available resistivity logs exceed 1% and most are substantially lower (Fig. 4.7*g*), which is consistent with direct measurements of rock salt porosity (Yaramanci, 1994; Yaramanci and Flach, 1992). The observation of percolating fluids at high dihedral angles and low porosities is not consistent with the static theory.

Viscous flow of rock salt due to the density contrast with the surrounding sediments may explain the failure of the static pore-scale theory to predict the percolation of pore space at high dihedral angles. At low effective mean stress, deformation induced

microcracking can lead to the formation of a percolating pore space (Schoenherr et al., 2007). This microcracking-induced percolation is commonly observed in the zone of disturbed rock around openings in salt mines or nuclear waste repositories and under high overpressures in nature (Hansen and Leigh, 2011; Schoenherr et al., 2007). At the depth of petroleum wells considered here the effective mean stress is sufficient that deformation occurs in the compaction regime where existing microcracks close and heal (Cristescu and Hunsche, 1998; Schulze et al., 2001).

However, deformation may induce permeability even in the absence of microcracking. At high effective mean stress and in the presence of small amounts of brine the dislocation creep of salt is accompanied by fluid-assisted dynamic recrystallization and pressure solution creep (Urai, 1983; Urai et al., 1986; Peach et al., 2001). Both static and dynamic recrystallization are associated with transformation of the isolated grain boundary fluid inclusions into grain boundary fluid films (Desbois et al., 2012; Drury and Urai, 1990). The dynamic wetting of the grain boundaries and compaction have been observed in deformation experiments under conditions where $\theta \approx 64^{\circ}$ (Peach et al., 2001). This suggests that dynamic grain boundary wetting induced fluid percolation and drainage at porosities below the percolation threshold.

These laboratory results must be extrapolated to natural conditions using appropriate micro-physical models and suggest that fluid-assisted dynamic recrystallization becomes important at strain rates below 10^{-10} s^{-1} (Urai et al., 1986). This is consistent with the recrystallized microstructures and X-ray microtomography of grain boundary brine films in natural rock salt, as well as estimated natural strain rates between $10^{-15} - 10^{-11}$ s^{-1} (Schoenherr et al., 2007; Thiemeyer et al., 2015; Jackson and Talbot, 1986). This confirms earlier suggestions that dynamic grain boundary wetting associated with grain boundary migration is a plausible mechanism in natural rock salt.

This conclusion is also supported by the comparison of the relative magnitude of shear stresses, $\Delta\sigma$, and the capillary pressure introduced by surface tension forces, Δp, given by capillary number

$$Ca = \frac{\Delta\sigma}{\Delta p} = \frac{\Delta\sigma}{2\gamma_{sl}/r}, \tag{5.4}$$

where r is the mean radius of disconnected pores. Micro-structural evidence preserves records of differential stresses up to 1 MPa in sub-horizontal bedded salts (Schléder and Urai, 2005) and 2 MPa in salt domes (Schoenherr et al., 2007; Carter et al., 1993). In comparison, the capillary pressure for $r = 10^{-4}$ m and $\gamma_{sl} = 0.1$ N/m is on the order of 10^3 Pa (Tromans and Meech, 2002). Therefore, $Ca \approx 10^3$ and the shear stresses in rock salt may exceed capillary pressures and hence facilitate deformation-assisted percolation. This provides an explanation for the penetration of hydrocarbons into shallow regions of the salt, where $\theta > 60^\circ$ and porosity is below the static percolation threshold (Figs. 4.7*g* and 5.11).

Beyond the direct application to salt-brine systems, the field observations reported here also provide an important test of a general theory that underlies our understanding of fluid percolation and flow in ductile regions of the Earth. This is of particular interest to the debate whether moderate dihedral angles can prevent the segregation of core-forming melts in the deforming lower mantle (Bruhn et al., 2000; Shannon and Agee, 1998; Shi et al., 2013). The inaccessibility of the Earth's mantle to field observations has prevented the resolution of this debate. The observations of fluid distribution in rock salt reported here show that deformation-assisted percolation is possible and suggest that core formation by percolation may be a viable mechanism, even if the dihedral angle is above 60°.

Chapter 6

Percolative Core Formation Due to Hysteresis in Melt Connectivity

6.1 Background and Literature Review

Rapid core formation in early planetary bodies is required by geochemical data from extinct radionuclides (Minarik, 2003). The most obvious mechanism for metal-silicate differentiation is the segregation of dense core forming melts by porous flow. However, experimental observations show that the texturally equilibrated metallic melt resides in isolated pockets that prevent percolation towards the center (Shannon and Agee, 1996; Minarik et al., 1996). The accretion of planets occurs very rapidly after birth of the central star (Trinquier et al., 2008; Briceño et al., 2001). Hf-W chronometry evidence requires core formation in planetesimals within a few million years (Kleine et al., 2002; Yin et al., 2002), consistent with radiogenic heating by decay of short-lived radio isotopes (Dauphas and Chaussidon, 2011). A natural mechanism for segregation of core forming liquids in a partially molten planetesimals is buoyancy-driven porous flow. However, it is thought that core forming melt does not form an interconnected pore network, because the interface between the melt and silicates has much higher energy than the grain boundaries (Minarik et al., 1996). The percolation in a texturally equilibrated partially molten material is determined by the dihedral angle,

$$\theta = \frac{1}{2}\cos^{-1}\left(\frac{\gamma_{sl}}{2\gamma_{ss}}\right), \tag{6.1}$$

where γ_{sl} and γ_{ss} are the solid-liquid and solid-solid surface energies, respectively (von Bargen and Waff, 1986; Ghanbarzadeh et al., 2014). For $\theta < 60^\circ$, the pore network percolates at any porosity, but a percolation threshold exists for larger dihedral angles.

Experimentally determined dihedral angles relevant to metal-silicate differentiation in planetesimals are typically larger than 60° and commonly between 70° to 110° (Shannon and Agee, 1996, 1998; Minarik et al., 1996; Ballhaus and Ellis, 1996; Gaetani and Grove, 1999). Dihedral angles smaller than 60° that have been measured under conditions of the Earth's lower mantle (Shi et al., 2013; Shannon and Agee, 1998) are not applicable to lower pressures in planetesimals. Similarly, it is unlikely that planetesimals experience mantle convection that can lead to deformation-assisted fluid percolation (Bruhn et al., 2000; Ghanbarzadeh et al., 2015c).

6.2 Level Set Method and Percolation Threshold

Estimates of percolation threshold range from 5% to 50%, based on theoretical calculations in simplified geometries (von Bargen and Waff, 1986), experiments (Yoshino et al., 2003) and textural observations in meteorites (Taylor, 1992). Therefore, it is important to quantify the minimum required porosity to allow percolative core formation. Here we use pore-scale computations of texturally equilibrated pore networks in a real polycrystalline material obtained by X-ray diffraction contrast tomography (Ludwig et al., 2009) (Fig. 6.1*a*). The computational results confirm a percolating melt network along the grain edges for any porosity, ϕ, when $\theta < 60^\circ$ (Fig. 6.1*b*). For larger dihedral angles, the percolation threshold was determined by systematically varying the porosity and dihedral angle (Fig. 6.1*c*-6.1*k*). Our results show that this threshold in a real texturally equilibrated material is much higher than previous estimates assuming idealized grains (Fig. 6.1*l*). Given the range of dihedral angles for core forming melts in a silicate matrix, 70° to 110° , the melt

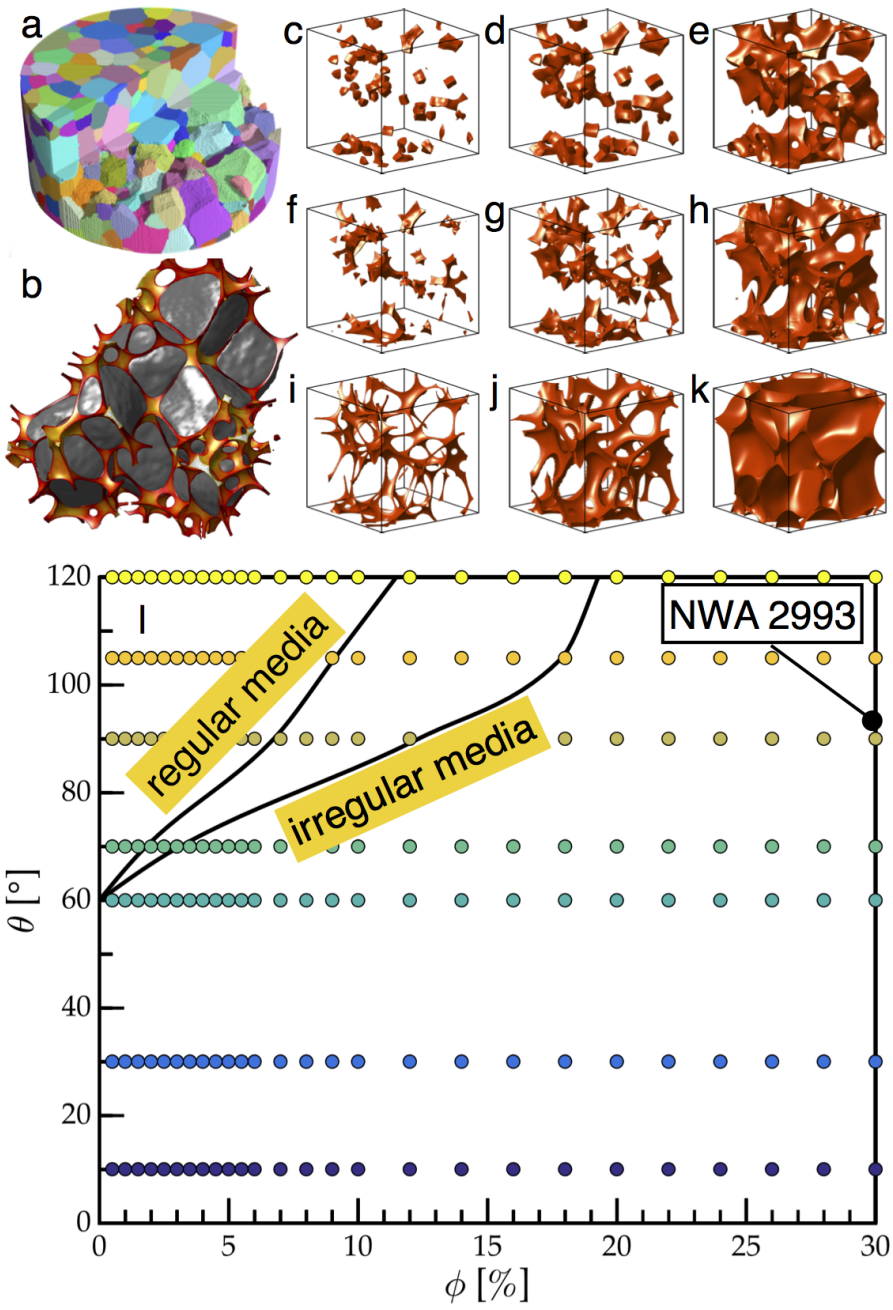


Figure 6.1: (a) Visualization of grains in a real polycrystalline material obtained by X-ray diffraction contrast tomography. (b) Fluid distribution on grain edges. (c-k) Visulization of pore networks at $\phi = 2\%$, 5% and 15% with $\theta = 10°$, $70°$ and $90°$. (l) Percolation threshold for regular and irregular media. Dots show the porosity and dihedral angle values that the connectivity is tested. Black dot shows where the meteorite NWA 2993 plots in $\phi\theta$-space.

fraction must exceed 10% to 19% to create an interconnected network.

6.3 Can Porosity Exceed Percolation Threshold in Nature?

To demonstrate that the percolation threshold can be overcome in a natural system, we studied the microstructure of meteorite NWA 2993 (Fig 6.2*a*). This coarse-grained lodran-like achondrite has undergone partial melting and now comprises orthopyroxene (37 vol.%), olivine (32 vol.%) and metal (31 vol.%) (Bunch et al., 2007). The oxygen isotopic composition suggests that this meteorite is a deeper plutonic sample from the winonaite parent body (Bunch et al., 2007). To determine the distribution of the metallic phases, we use X-ray microtomography (Fig 6.2*b* and 6.2*c*). The imaging procedure, except the resolution of 9.3μm, and image processing steps are similar to the methods presented in Sections 4.2.2 and 4.2.3. Assuming this distribution represents the original melt network, the pore space is clearly connected and thus the porosity has exceeded the percolation threshold. The mean curvature of the metal-silicate interface has a distribution with a single narrow peak, suggesting the pore space is approaching textural equilibrium (Fig. 6.2*d*). The distribution of the apparent dihedral angles between metal and silicate grains has a median of $93^\circ \pm 12^\circ$, consistent with experiments on synthetic materials (Fig. 6.2*e*). The connectivity of melt network inferred from microtomography is consistent with the computationally determined percolation threshold of 12% (Fig. 6.1*l*).

The volume fraction of metallic cores in the terrestrial planets varies between 12% for Mars and Venus up to 40% for Mercury. This emphasizes the importance of determining the appropriate dihedral angles and the associated percolation threshold. However, at least some of the planetesimals that accreted to form these planets contained enough metallic phases to overcome the percolation threshold. The current assumption is that these melt networks become disconnected as the porosity drops below the percolation threshold during

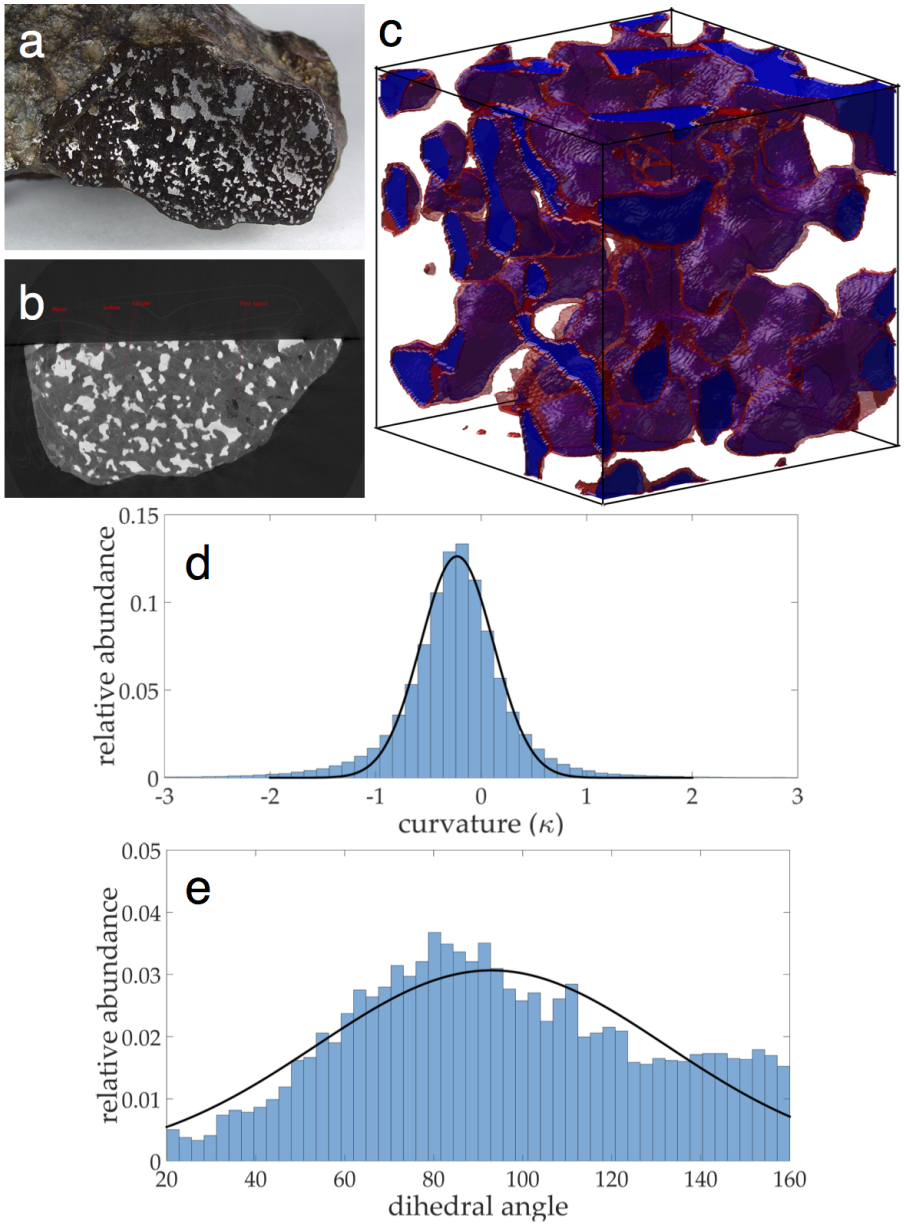


Figure 6.2: Evidence of texturally equilibrated iron percolation in meteorite NWA 2993. (a) Optical photograph of the meteorite. (b) X-ray microtomography slice shows the existence of three phases: metal, sulfide and silicate. Metal and sulfide are the pore fluids, and sulfide is a wetting fluid for silicate matrix and iron is non-wetting fluid. (c) The surface of iron (blue interface), therefore, is coated with a thin layer of sulfide (red interface). (d) distribution of solid-liquid mean curvature shows a single narrow peak. (e) distribution of apparent dihedral angels has a median of $93^\circ \pm 12^\circ$.

drainage. Given the large percolation threshold in real materials, this would strand the majority of core forming liquid in the mantle (Fig. 6.1*l*).

6.4 Hysteresis in Pore Network Topology and Permeability

It is currently not recognized that texturally equilibrated melts exhibit hysteresis in network connectivity (von Bargen and Waff, 1986). This hysteresis arises due to nonlinearity of the governing equations (Ghanbarzadeh et al., 2015b), which allows for the multiple pore space configurations, some percolating and some not, having the same volume fraction and dihedral angle. The history of the material determines the relevant solution. Therefore, pore-scale simulations can be used to track the evolution of melt connectivity during partial melting and subsequent drainage (Fig. 6.3*a*). As the porosity increases, the neighboring melt pockets merge and form a connected path at the percolation threshold. Further increases in porosity reduce the volume fraction of isolated pockets by connecting them to the interconnected melt network. As the melt begins to segregate, compaction reduces the porosity but the melt network remains connected, even as the porosity drops below the percolation threshold. This occurs because the connected network has a lower surface energy than the equivalent disconnected pockets. As drainage and compaction continue, the melt eventually disconnects, but only 1-2% is trapped (Fig. 6.3*b*).

It is generally assumed that an interconnected melt network with a large dihedral angle has a low permeability (Minarik et al., 1996; Shannon and Agee, 1996). However, Lattice Boltzmann simulations on the computed pore networks show that the permeability has a weak dependence on dihedral angle once the percolation threshold is passed (Fig. 6.3*b*). Furthermore, permeability remains considerable in the region between the trapping and percolation threshold (Fig. 6.3*b*). Therefore, hysteresis provides a mechanism for separation of iron-rich metallic melt from an olivine-rich solid mantle by porous flow, if

the percolation threshold is exceeded. The trapping of small amounts of melt also provides a mechanism for incomplete metal-silicate segregation, which is required by the amount of heavy siderophile elements in the Earth's mantle (Jones and Drake, 1986; Stevenson, 1990; Murthy, 1991).

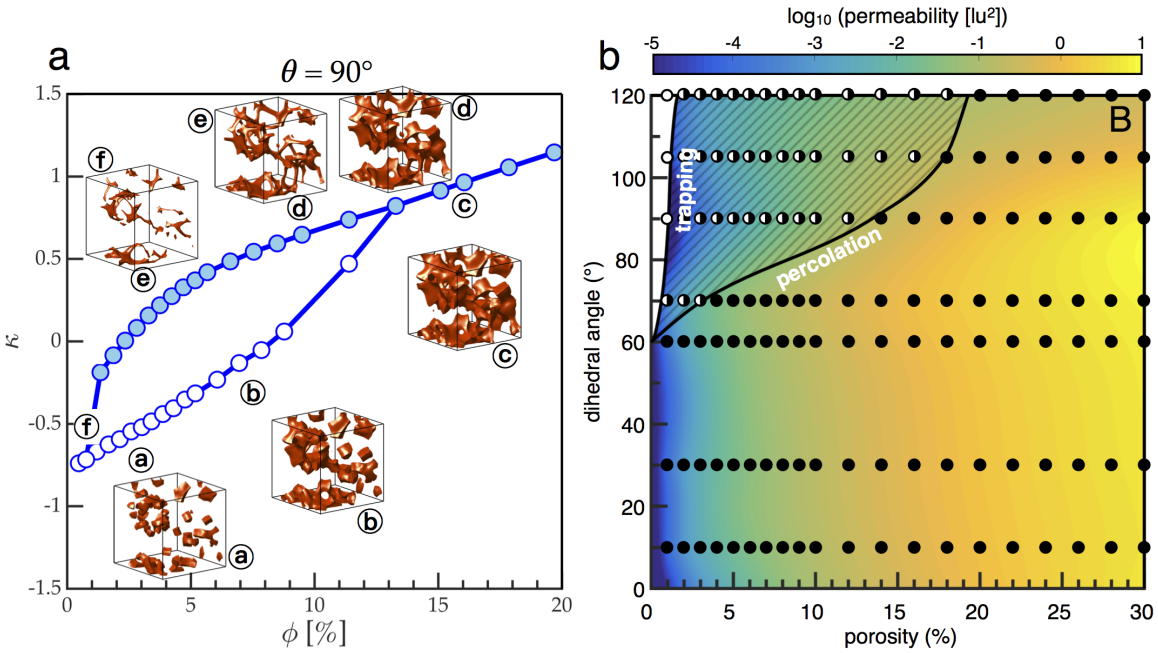


Figure 6.3: Hysteresis in pore network connectivity. (a) As porosity increases, the initially disconnected pore fluid becomes connected and form percolating pore network. The segregation of the heavy metallic core starts in this moment and porosity reduces with drainage toward core. The pore network remains connected to porosities much below percolation threshold due to hysteresis. (b) Percolation and trapping thresholds plotted in $\phi\theta$-space. The normalized permeability of the irregular medium (in lattice units) is shown in colored background. The permeability can be converted to SI for average grain size of 1 mm by multiplying in 1.0367×10^{-10}.

6.5 Planetesimal-Scale Continuum Model for Melt Segregation

A planetesimal-scale continuum model based on mass, momentum and energy conservation is used to investigate the time scales required for core formation by porous flow with hysteresis. This simplified model considers melting of the metallic phase due to ra-

diogenic heat generation and melt segregation due to porous flow coupled to viscous compaction of the silicate matrix in an evolving gravitational field. Below we outline the details of the model.

6.5.1 Gravitational Potential

The gravitational potential, Φ, and the gravitational field, g of a planetesimal with density $\bar{\rho}$ are given by

$$\nabla \cdot \mathbf{g} = 4\pi G\bar{\rho}, \tag{6.2}$$

$$\mathbf{g} = -\nabla\Phi, \tag{6.3}$$

where G is the gravitational constant. Here $\mathbf{g}$ points down towards the center of the planetesimal.

6.5.2 Conservation Laws

The planetesimal is assumed to be composed to consist of three possible phases: olivine (o) , solid iron (i) and the core-forming iron melt (m). Olivine and iron comprise the solid (s). If the volume fraction of phase j is denoted ϕ_j, the volume fraction constraint, assuming constant density for each phase, requires that

$$\phi_o + \phi_i + \phi_m = \phi_s + \phi_m = 1. \tag{6.4}$$

Assuming that the solid phases move together, $\phi_s = \phi_o + \phi_i$, and separate from the melt phase, m, the mass conservation equations for the phases are given by:

$$(\rho_m \phi_m)_t + \nabla \cdot (\rho_m \phi_m \mathbf{v_m}) = \Lambda, \tag{6.5}$$

$$(\rho_i \phi_i)_t + \nabla \cdot (\rho_i \phi_i \mathbf{v_s}) = -\Lambda, \tag{6.6}$$

$$(\rho_o \phi_o)_t + \nabla \cdot (\rho_o \phi_o \mathbf{v_s}) = 0, \tag{6.7}$$

where ρ_j is the density of phase j, Λ is the mass melt production rate and $\mathbf{v_m}$ and $\mathbf{v_s}$ are the melt and solid velocities, respectively.

6.5.3 Constitutive Relations

The density is given by

$$\rho = \phi_o \rho_o + \phi_i \rho_i + \phi_m \rho_m \tag{6.8}$$

The volumetric flux of the melt relative to the solid is given by Darcy's law

$$q_r = \phi_m(\mathbf{v_m} - \mathbf{v_s}) = -\frac{k(\phi_m)}{\mu_m}(\nabla p + \Delta\rho \mathbf{g}) \tag{6.9}$$

where μ_m is the melt viscosity, $\Delta\rho = \rho_s - \rho_m$, and the permeability-porosity relationship, $k(\phi_m)$, is determined by the LBM computations in the realistic, image-based pore networks (see Chapter 3). The permeability value changes with dihedral angle, as well, but for simplicity in notation, we show permeability with $k(\phi_m)$. The difference in melt and solid pressures is given by the compaction equation

$$p = p_m - p_s = \xi(\phi_m) \nabla \cdot \mathbf{v_s}, \tag{6.10}$$

where ξ_0 is the reference bulk viscosity of the solid and p_m is the solid pressure (McKenzie, 1984, 1985; Hewitt and Fowler, 2008). Finally, we assume the rotational component of $\mathbf{v_m}$ is negligible that there exists a solid velocity potential, U, such that (Ribe, 1985)

$$\mathbf{v_s} = \nabla U. \tag{6.11}$$

6.5.4 Enthalpy Model with Radiogenic Heat Generation and Melting

6.5.4.1 Enthalpy Equations for One Component Substance

Assuming negligible pressure changes, the specific enthalpy of the solid phase of iron, phase i, is given by

$$h_i = h_i^0 + \int_{T^0}^{T} c_{p,i} \mathrm{d}T, \tag{6.12}$$

$$= h_i^0 + c_{p,i}(T - T^0) \tag{6.13}$$

where h_i^0 is the enthalpy of iron at the reference state (298 °K, 1 atm), T^0 is the reference temperature, $c_{p,i}$ is the specific heat capacity of iron at constant pressure and the temperature of the system is below the melting temperature $T < T^m$. If enthalpy is added to the solid iron, and temperature rises so $T > T^m$ the specific latent heat of melting the iron is given by,

$$\Delta H = \Delta H_{T^m}^0 + \Delta c_p (T - T^m), \tag{6.14}$$

where $\Delta c_p = c_{p,m} - c_{p,i}$. The specific enthalpy of the resulting molten phase, m, assuming local thermodynamic equilibrium at the melting temperature $T = T^m$, is given by

$$h^m = h_i^0 + \Delta H \tag{6.15}$$

$$= h_i^0 + c_{p,i}(T^m - T^0) + \Delta H_{T^m}^0. \tag{6.16}$$

In the equilibrium model presented here, $T = T^m$ is the only temperature at which solid and molten iron may coexist simultaneously. At temperatures higher than the melting temperature $T > T^m$, the enthalpy of the molten iron phase is given as

$$h^m = h_i^0 + c_{p,i}(T^m - T^0) + \Delta H + \int_{T_m}^{T} c_{p,m} \mathrm{d}T. \tag{6.17}$$

At the melting temperature, the bulk enthalpy of the pure iron-melt solution is the pore-weighted average of the specific enthalpies multiplied by their respective densities,

$$H = \phi_i \rho_i h_i + \phi_m \rho_m h_m. \tag{6.18}$$

Throughout the development of this model, densities are treated as constant across all temperatures and pressures.

6.5.4.2 Silicate-Iron Solid Solution and Iron Liquid Solution

To model the dynamic melt segregation within a planetesimal, a refractory silicate phase is added to the system. This silicate phase, hereafter referred to as "olivine" denoted by the subscript, o, is more refractory than the pure iron. For the purposes of this study it is assumed that the planetesimal is always below the melting temperature of the olivine phase for the duration of the model.

Similar to Eq. 6.18, the bulk enthalpy of the olivine-iron-melt solution is the volume-weighted average of the specific enthalpies of each component and phase multiplied by their respective densities

$$H = \phi_i \rho_i h_i + \phi_m \rho_m h_m + \phi_o \rho_o h_o. \tag{6.19}$$

where the volume fraction constraint,

$$\phi_i + \phi_m + \phi_{ol} = 1, \tag{6.20}$$

must be satisfied. The specific enthalpy of olivine is given by

$$h_i = h_o^0 + c_{p,o}(T - T^0). \tag{6.21}$$

For the duration of the model presented here, the temperature never reaches the melting temperature of olivine so there is no silicate phase change. Since only the iron

component changes phase from solid to melt, it is convenient to rearrange Eq. 6.20, for ϕ_i and rewrite Eq. 6.19 so that

$$H = (1 - \phi_m - \phi_o)\rho_i h_i + \phi_m \rho_m h_m + \phi_o \rho_o h_o. \tag{6.22}$$

The equation Eq. 6.22 is rearranged for melt fraction, and melt fraction must be zero below the melting temperature $1 - \phi_o$ above the melting temperature, so

$$\phi_m = \begin{cases} 0 & T < T^m \\ \left(\frac{H - \rho_i h_i - \rho_o \phi_o h_o + \rho_i \phi_o h_i}{\rho_m h_m - \rho_i h_i} \right) & T = T^m \\ 1 - \phi_o & T > T^m \end{cases} \tag{6.23}$$

Given the conserved quantity H, equation Eq. 6.23 can be readily solved. All functions representing specific enthalpies are linear. Thus, the melt fraction increases linearly with H during partial melting at T^m with constant melt fractions of either $1 - \phi_o$ or zero at all other temperatures.

6.5.4.3 Enthalpy Transport Model

The conservation equation for enthalpy is given by

$$\frac{\partial H}{\partial t} + \nabla \cdot \left[\left(\phi_o \rho_o h_o + \phi_i \rho_i h_i \right) \mathbf{v_s} + \phi_m \rho_m h_m \mathbf{v_m} - \bar{k} \nabla T \right] = \Gamma_T, \tag{6.24}$$

where $\mathbf{v_s}$ is the velocity of the solid and $\mathbf{v_m}$ is the melt velocity. The term on right hand side, Γ_T is the enthalpy generation due to radioactive decay of ^{26}Al, and is given by

$$\Gamma_T = \phi_s \rho_s H_0 X_{Al} \left[\frac{^{26}Al}{^{27}Al} \right]^i e^{-\lambda t}, \tag{6.25}$$

where H_0 is heating production of ^{26}Al decay, X_{Al} is initial aluminum content (%weight) in planetesimal, $\left[\frac{^{26}Al}{^{27}Al} \right]^i$ is initial ^{26}Al to ^{27}Al ratio, and γ is the decay constant in s^{-1}. The average thermal conductivity of the medium is given by

$$\bar{k} = \phi_i k_i + \phi_m k_m + \phi_o k_o. \tag{6.26}$$

For computational and coding purposes, it would be more convenient to rearrange Eq. 6.24 by adding and subtracting $\mathbf{v_s}$ terms from $\mathbf{v_m}$ to formulate the relative Darcy melt flux, $\mathbf{q_r} = \phi_m(\mathbf{v_m} - \mathbf{v_i})$, informed by the McKenzie (1984) viscous compaction model as follow

$$\frac{\partial H}{\partial t} + \nabla \cdot \left[\left(\phi_o \rho_o h_o + \phi_i \rho_i h_i + \phi_m \rho_m h_m \right) \mathbf{v_s} + \rho_m h_m \mathbf{q_r} - \bar{k} \nabla T \right] = \Gamma_T. \tag{6.27}$$

6.6 Dimensionless Continuum Model for Melt Segregation

Due to the enormous contrast between the solid (olivine) and liquid (molten iron) properties, including viscosity, heat capacity and density, numerical implementation of the presented model in the previous section is challenging. The contrast in properties results in extremely large matrix condition numbers for the corresponding linear system of equations. This leads to a matrix that is ill-conditioned and practically is almost singular. In order to avoid such a problem in numerical implementation of the continuum model, we present the dimensionless form of the equations, solve the dimensionless system numerically and then convert the results to dimensional variables.

6.6.1 Dimensionless Gravitational Potential

To scale the evolving gravitational field for the viscous compaction model for planetesimal segregation, the average density is normalized by density of iron

$$\rho_D = \phi_i + \frac{\rho_m}{\rho_i}\phi_m + \frac{\rho_{ol}}{\rho_i}\phi_{ol} \tag{6.28}$$

$$\nabla^2 \Phi_D = \rho_D, \tag{6.29}$$

$$\mathbf{g}_D = -\nabla \Phi_D, \tag{6.30}$$

where,

$$\rho_C = \rho_i, \quad \Phi_C = 4\pi\rho_C r_C^2, \quad \mathbf{g}_C = \frac{\Phi_C}{r_C} \tag{6.31}$$

Here, all the properties and notations with subscript C denote the characteristic value. Term r_C is the characteristic length scale, which is derived from dimensionless compaction equation (next section, Eq. 6.32) and is called the compaction length.

6.6.2 Dimensionless Compaction Equation

Using the characteristic scales for density, gravity and the radius of the planet, the dimensionless fluid overpressure and scale for pressure and are

$$-\nabla \cdot (k_D \nabla p_D) + \frac{p_D}{\xi_D} = \nabla \left(k_D \Delta\rho_D \mathbf{g}_D\right), \tag{6.32}$$

$$p_C = \rho_C \mathbf{g}_C r_C, \tag{6.33}$$

$$r_C = \left(\frac{k_C \xi_C}{\mu}\right)^{\frac{1}{2}}, \tag{6.34}$$

where k_C and ξ_C are characteristic permeability and bulk viscosity evaluated at initial porosity. k_D is dimensionless permeability, p_D is dimensionless pressure and ξ_D is dimensionless matrix bulk viscosity. The dimensionless change in density is given as

$$\Delta\rho_D = \frac{\phi_i + \frac{\rho_{ol}}{\rho_i}\phi_{ol}}{\phi_i + \phi_{ol}} - \frac{\rho_m}{\rho_i}. \tag{6.35}$$

The density change is the primary driving force for Eq. 6.32.

In this formulation $p_D = p_{fD} - p_{sD}$. This means that dimensionless pressure is equal to the lithostatic pressure when $p_D = 0$. Any positive number indicates that there is an overpressure in the fluid and decompaction may occur via diseggregation of the solid matrix. Conversely, a negative number in p_D indicates a solid overpressure. In this case, the solid phase re-compacts forcing the fluid phase out of the pore space. In this model, differences in density will drive the heavier melt downwards and allow the lighter solid phases to be squeezed upwards.

6.6.3 Dimensionless Relative Darcy Flux, Melt and Solid Velocities

The equation for the dimensionless relative Darcy flux is obtained by taking the gradient of the fluid overpressure term obtained via Eq. 6.32, using Eq. 6.30 and Eq. 6.35,

$$\mathbf{q_r}_D = -(\nabla p_D + \Delta\rho_D \mathbf{g}_D), \tag{6.36}$$

where

$$\mathbf{q_r}_C = \frac{k_C \rho_C \mathbf{g}_C}{\mu}. \tag{6.37}$$

The characteristic permeability-porosity, $k(\phi_C)$, is given by the LBM computations at initial porosity. The characteristic porosity ϕ_C is chosen to be the initial volume fraction of solid iron in the planetesimal before melting occurs,

$$\phi_C = \phi_i. \tag{6.38}$$

Momentum is conserved and the velocity of the solid phases are found via the compaction potential and its gradient,

$$\nabla^2 U_D = \frac{p_D}{\xi_D(\phi_D)}, \tag{6.39}$$

$$\mathbf{v}_{sD} = \nabla U_D, \tag{6.40}$$

Eqs. 6.39 and 6.40 are formulated with the following characteristic scales

$$U_C = \frac{\rho_C \mathbf{g}_C r_C^2}{\xi_C}, \quad \mathbf{v}_{\mathbf{s}C} = \frac{k_C \rho_C \mathbf{g}_C}{\mu}. \tag{6.41}$$

6.6.4 Dimensionless Evolution Equations

The volume fractions of olivine and melt evolve through the divergence of melt and solid flow fields respectively,

$$\frac{\partial \phi_o}{\partial t_D} + \nabla \cdot (\phi_o \mathbf{v}_{\mathbf{s}D}) = 0, \tag{6.42}$$

$$\frac{\partial \phi_m}{\partial t_D} + \nabla \cdot (\phi_m \mathbf{v}_{\mathbf{m}D}) = \frac{p_D}{\xi_D}. \tag{6.43}$$

Here, the characteristic time scale depends on viscous properties and density of the matrix, the characteristic gravitational field and the viscosity of the molten iron,

$$t_C = \sqrt{\frac{\mu \xi_C}{k_C}} \frac{1}{\rho_C \mathbf{g}_C}. \tag{6.44}$$

6.6.5 Dimensionless Enthalpy Transport Model

To nondimensionalize the enthalpy transport equation, temperature is scaled as

$$T_D = \frac{T - T^0}{T^m - T^0}. \tag{6.45}$$

With temperature scaled as in Eq. 6.45,

$$\phi_m = \begin{cases} 0 & T_D < 1 \\ (0,1) & T_D = 1 \\ 1 & T_D > 1. \end{cases} \tag{6.46}$$

In this formulation, all three phases (one melt and two solid phases) may only coexist at local thermodynamic equilibrium when dimensionless temperature is unity.

Next, characteristic scales must be chosen for each specific enthalpy term in Eq. 6.24. Here each specific enthalpy is scaled to iron specific heat capacity multiplied by the difference in the two reference temperatures, $\Delta T^0 = T^m - T^0$, so

$$h_C = c_{p,i}\Delta T^0. \tag{6.47}$$

The bulk enthalpy of the partially molten solid solution with liquid iron solution is scaled to the bulk enthalpy component associated with the solid iron phase, $H_C = \rho_i h_i$ so

$$H_D = \frac{H}{\rho_i h_i}. \tag{6.48}$$

A characteristic scale for average conductivity of the medium is chosen as

$$\bar{k}_C = \frac{r_C^2 H_C}{\Delta T^0 t_C} \tag{6.49}$$

based on the characteristic enthalpy and temperatures chosen in this section along with the characteristic radius and timescale chosen for the planetesimal via the viscous compaction model.

By inserting Eqs. 6.47, 6.48 and 6.49, along with characteristic melt flux and solid velocity (informed by the three phase viscous compaction model Eqs. 6.37 and 6.41) into Eq. 6.27 a dimensionless enthalpy transport equation is obtained,

$$\begin{aligned}\frac{\partial H_D}{\partial t_D} + \nabla_D\cdot\Bigg[\mathrm{Pe_s}\left(\frac{\rho_o}{\rho_i}\frac{h_{oC}}{h_i}h_{oD}\phi_{ol} + \frac{\rho_m}{\rho_i}\frac{h_{mC}}{h_i}h_{mD}\phi_m + \phi_i\right)\mathbf{v_s}_D \\ + \mathrm{Pe_m}\frac{\rho_m}{\rho_i}\frac{h_{mC}}{h_i}h_{mD}\mathbf{q_r}_D - \bar{k}_D\nabla_D T_D\Bigg] = \frac{\Gamma_T}{\rho_i h_i}.\end{aligned} \tag{6.50}$$

The Peclet numbers, which describe the characteristic ratio of advective to diffusive transport rate, for the mobile solid and modified Darcy melt flux are given as

$$\mathrm{Pe_s} = \frac{\mathbf{v_s}_C t_C}{r_C}, \quad \mathrm{Pe_m} = \frac{\mathbf{q}_C t_C}{r_C}. \tag{6.51}$$

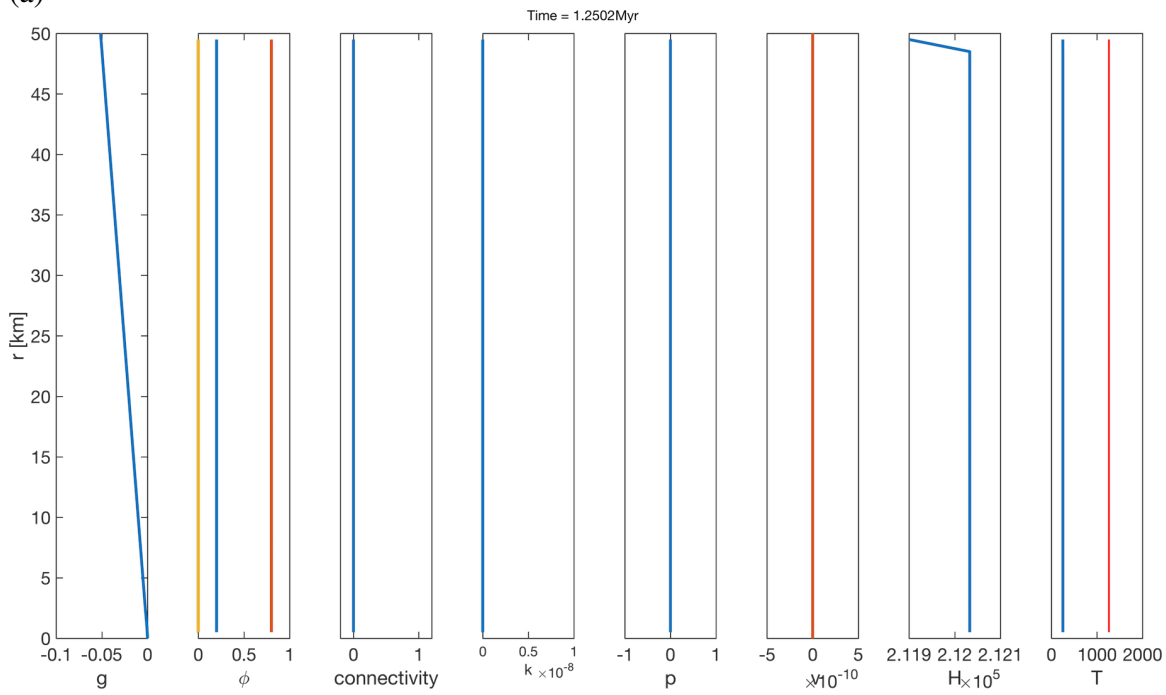
6.7 Results and Discussion

In all simulations below the dihedral angle is assumed to be 90°. Initially the metal is uniformly distributed and its volume fraction is 20%, which exceeds the percolation threshold. The planetesimal radius is 50 km, density of iron, molten iron and olivine are considered constant and 8000, 7000 and 2600 kg/m^3, respectively. Bulk solid viscosity and molten iron viscosity are assumed 10^{19} and 1 Pa.s and average olivine grain size is assumed 1 mm. Thermal conductivity of iron, molten iron and olivine are considered constant and 80, 35 and 2.5 W/mK, respectively. Heat capacity of iron, molten iron and olivine at constant pressure are considered constant and 400, 66 and 1000 W/mK, respectively. The time of formation (accretion time) in simulations presented in Fig. 6.4 is assumed to be 1.25 Myr, initial ^{26}Al/^{27}Al ratio of 5×10^{-5}, decay constant of 3.012×10^{-14} s^{-1}, aluminum content of 1.5 %wt, and heating decay of 0.355 W/kg. The initial temperature of the planetesimal, the surface temprature and the melting point of iron are, 250, 250 and 1261°K, respectively. Simulation results show an undifferentiated shell due to the cold thermal boundary layer at the surface (Elkins-Tanton et al., 2011).

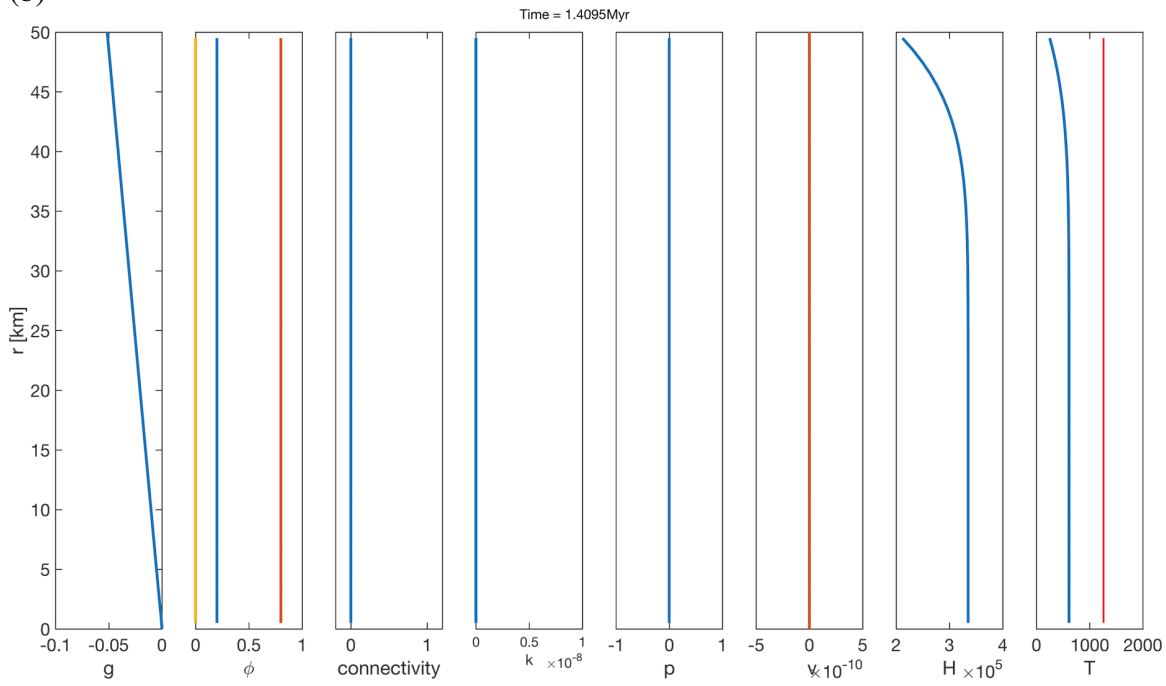
Fig. 6.4 shows a series of gravity, porosity, permeability, overpressure, solid and relative Darcy flux, enthalpy and temperature profiles for the conditions mentioned above. Fig. 6.4*a* represents the initial condition, at $t = 1.25$ Myr. Planet initially heats up due to decay of radioactive elements. Fig. 6.4*b*- Fig. 6.4*d* shows that all parameters other than enthalpy and temperature do not change until the initiation of melting. Iron starts to melt at the time between 1.73 and 1.88 Myr. The melting happens very fast and all the iron in the region with temperature above melting point becomes molten iron (Fig. 6.4*e*). The melt starts to move downward, while the olivine pushes outward. Heating continues and melting region grows (Fig. 6.4*f*). The growth of melting region stops at some point due to cold boundary condition at the surface, but melt continues to drain toward center (Fig. 6.4*g*-

6.4*j*). A distinctive core is formed at the time 4.12 Myr up to radius 14 km. After this time, the heat source becomes weak and the planet starts to cool down (Fig. 6.4*l*-6.4*o*). At the time of 6.35 Myr, a solid metallic core with radius of 15km is formed.

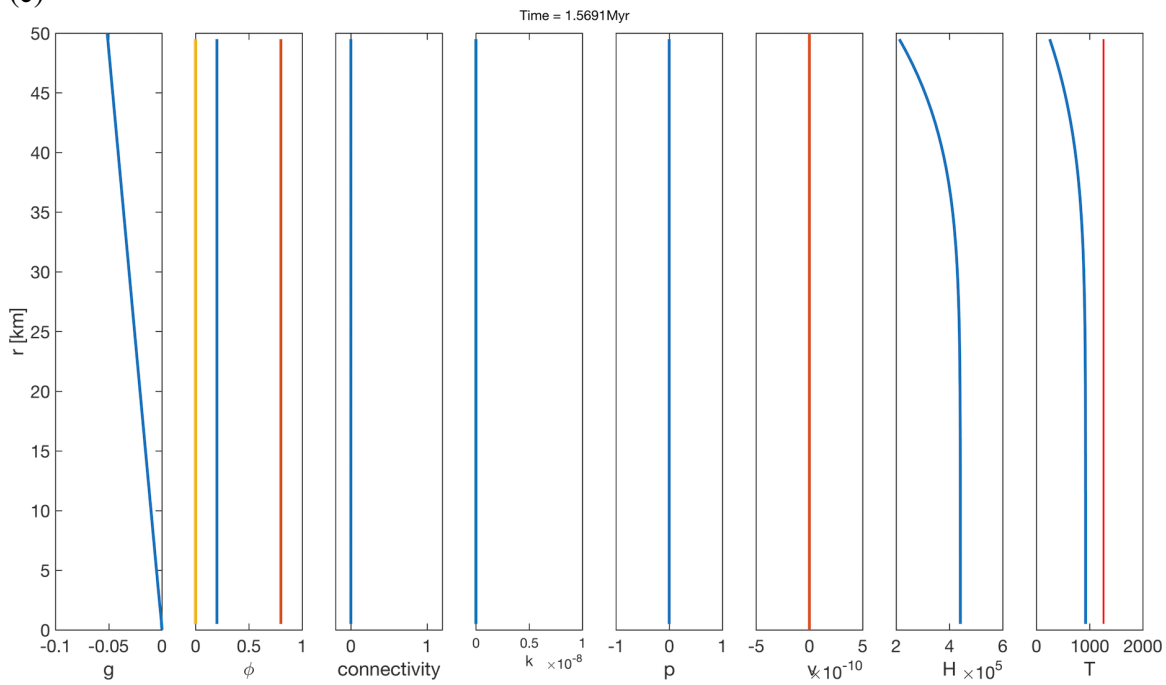
(a)


Time = 1.2502Myr
r [km]
g
ϕ
connectivity
k ×10^-8
p
v ×10^-10
H ×10^5
T

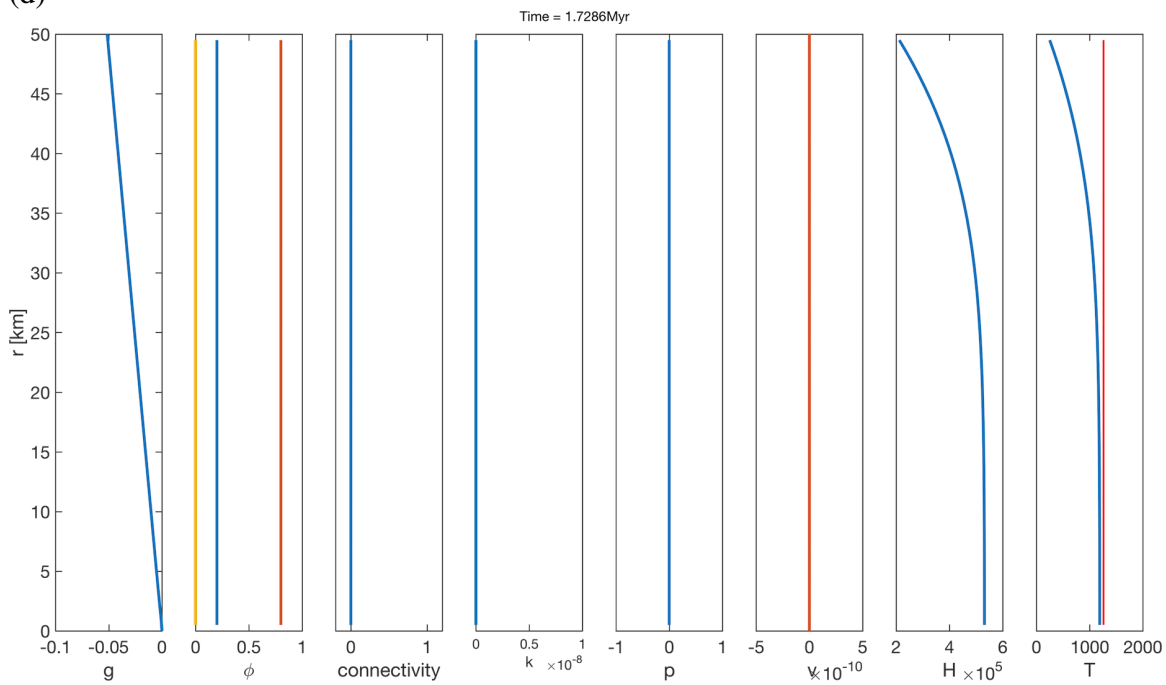
(b)


Time = 1.4095Myr
r [km]
g
ϕ
connectivity
k ×10^-8
p
v ×10^-10
H ×10^5
T

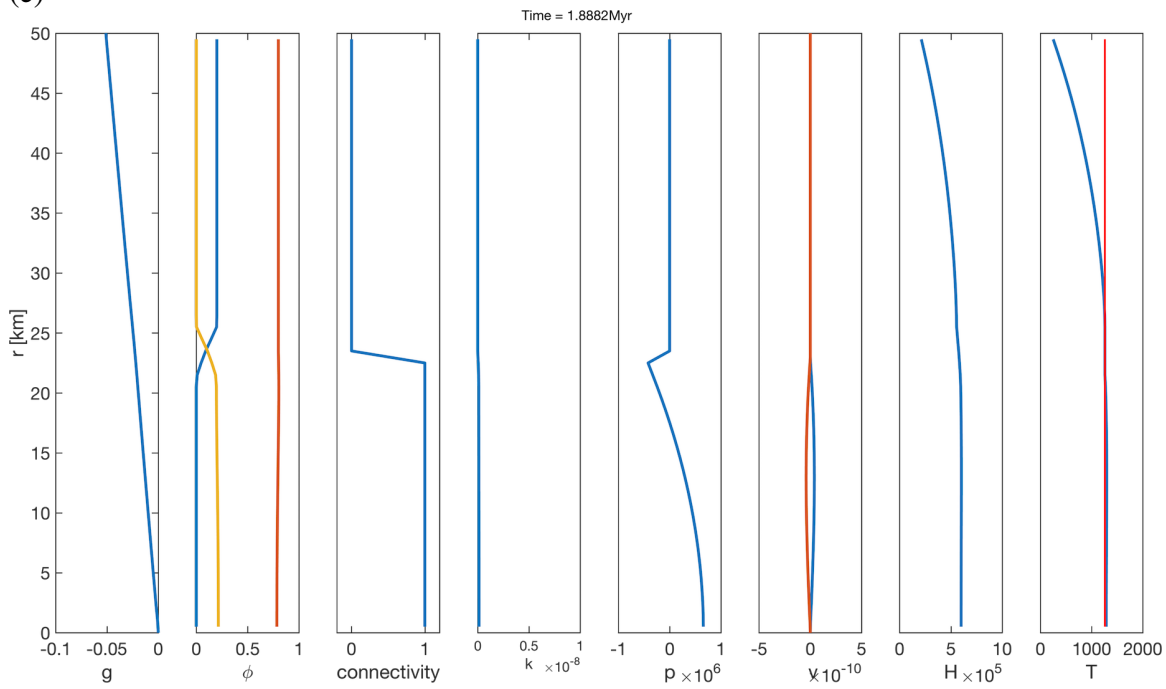
(c)


Time = 1.5691Myr
r [km]
g
ϕ
connectivity
k ×10^-8
p
v ×10^-10
H ×10^5
T

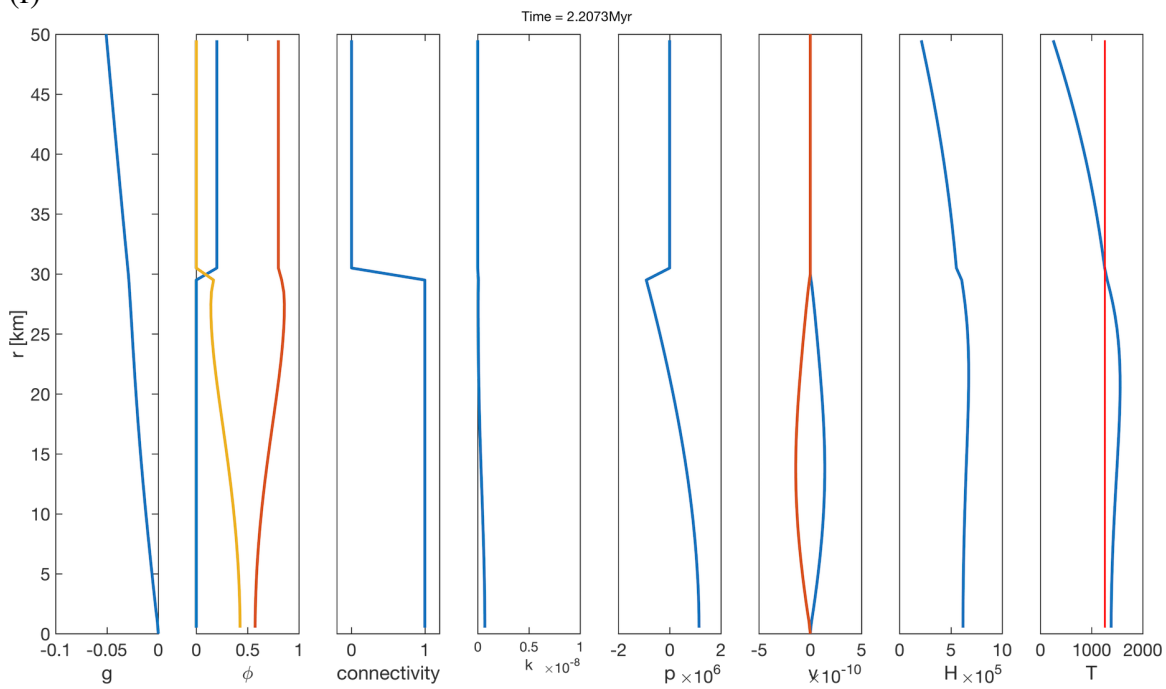
(d)


Time = 1.7286Myr
r [km]
g
ϕ
connectivity
k ×10^-8
p
v ×10^-10
H ×10^5
T

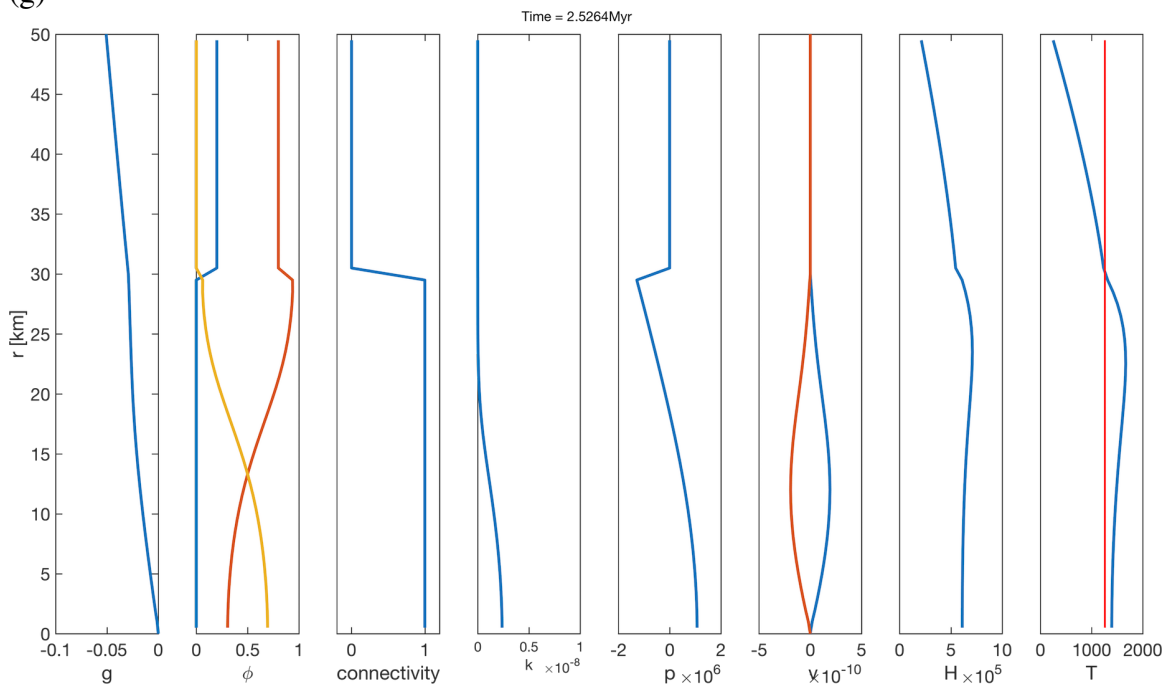
(e)


Time = 1.8882Myr
r [km]
g
ϕ
connectivity
k ×10^-8
p ×10^6
V ×10^-10
H ×10^5
T

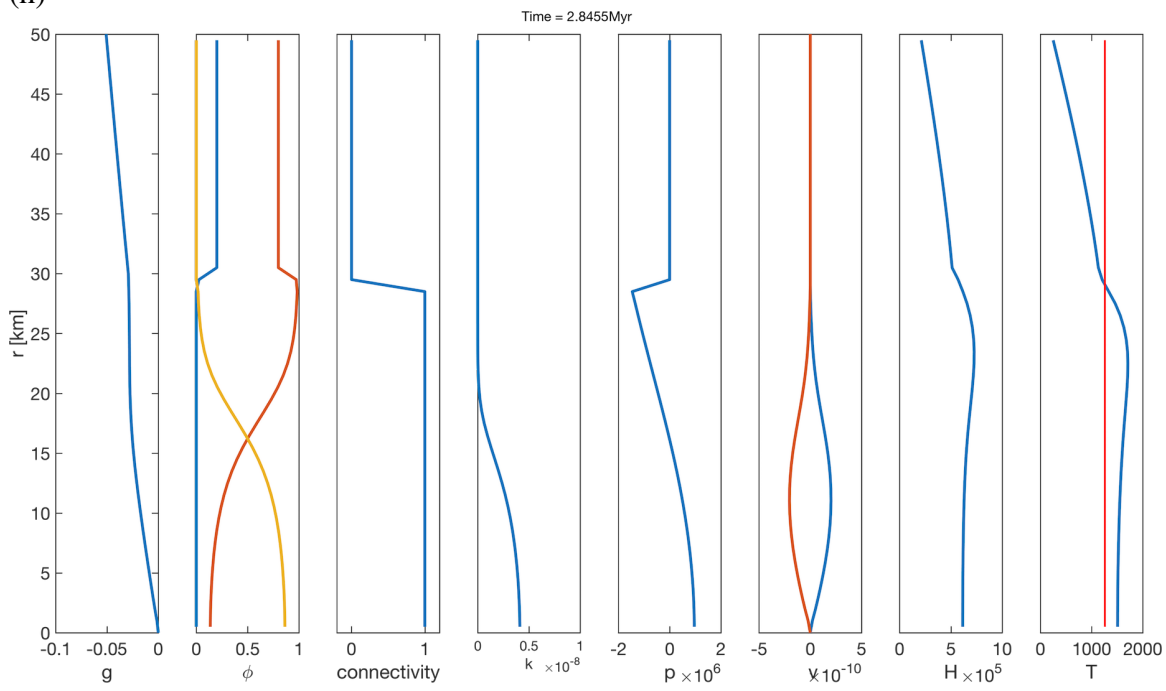
(f)


Time = 2.2073Myr
r [km]
g
ϕ
connectivity
k ×10^-8
p ×10^6
V ×10^-10
H ×10^5
T

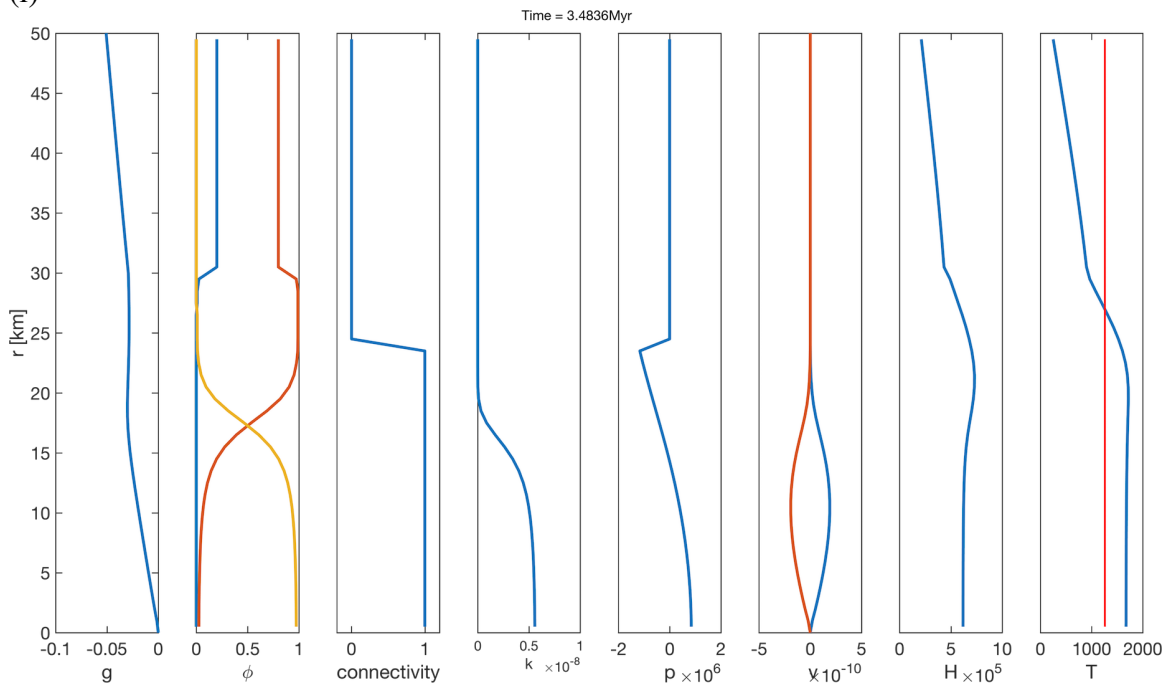
(g)


Time = 2.5264Myr
r [km]
g
ϕ
connectivity
k ×10^-8
p ×10^6
Ψ ×10^-10
H ×10^5
T

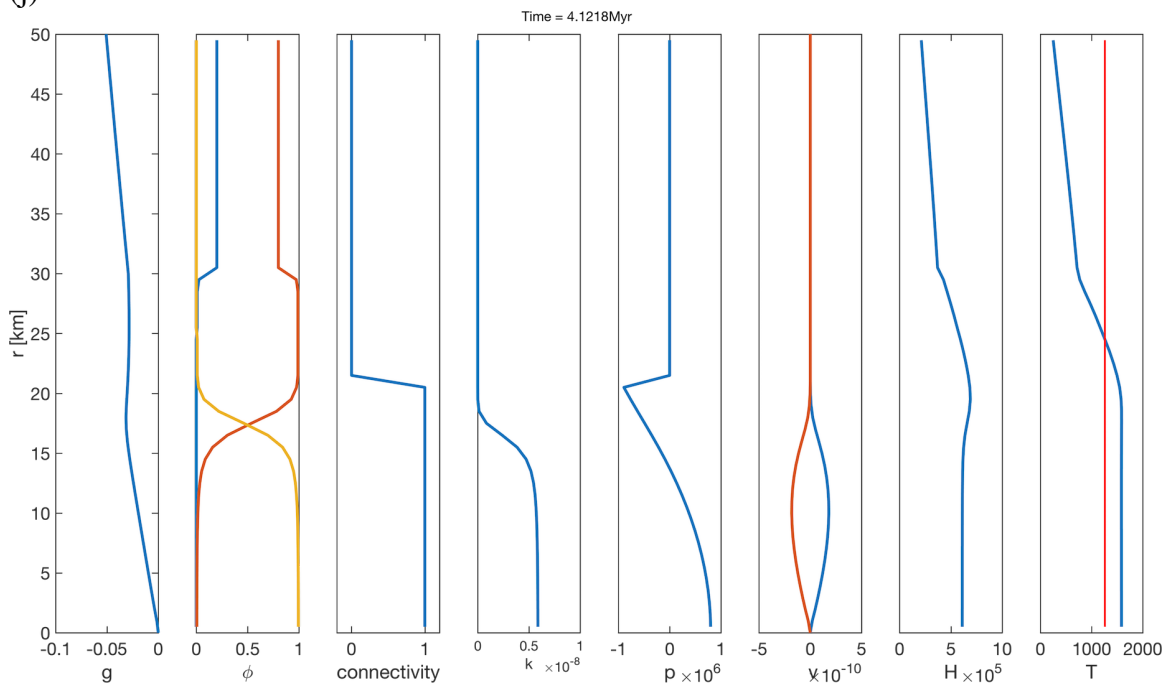
(h)


Time = 2.8455Myr
r [km]
g
ϕ
connectivity
k ×10^-8
p ×10^6
Ψ ×10^-10
H ×10^5
T

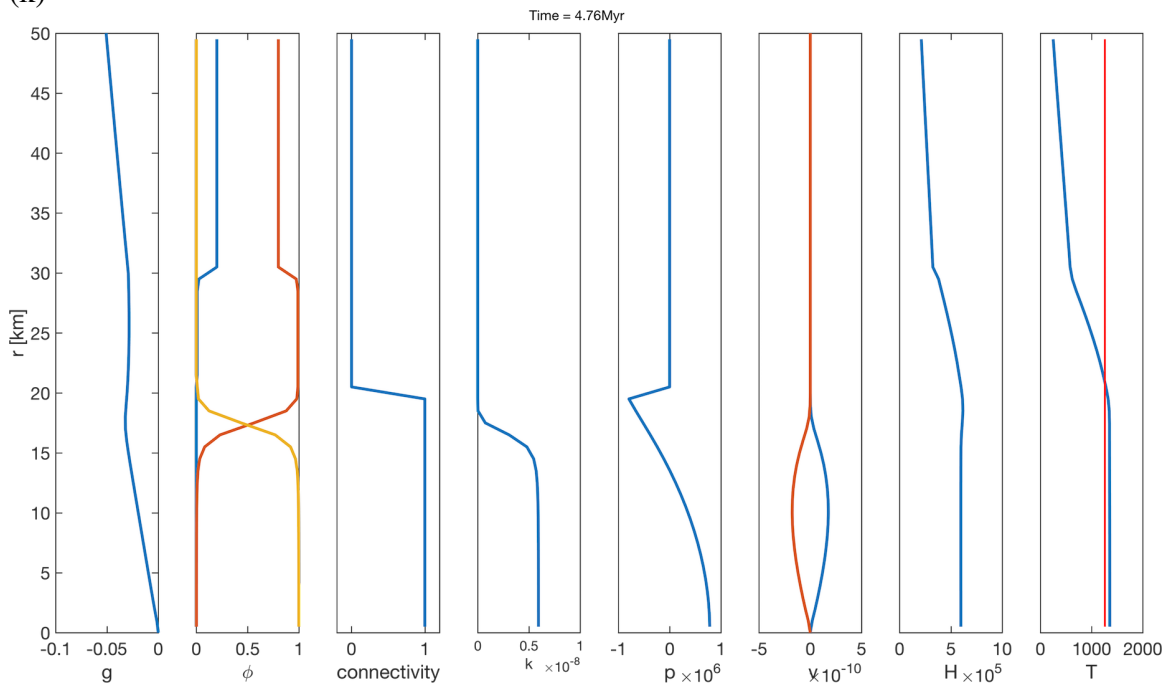
(i)


Time = 3.4836Myr
r [km]
g
ϕ
connectivity
k ×10^-8
p ×10^6
V ×10^-10
H ×10^5
T

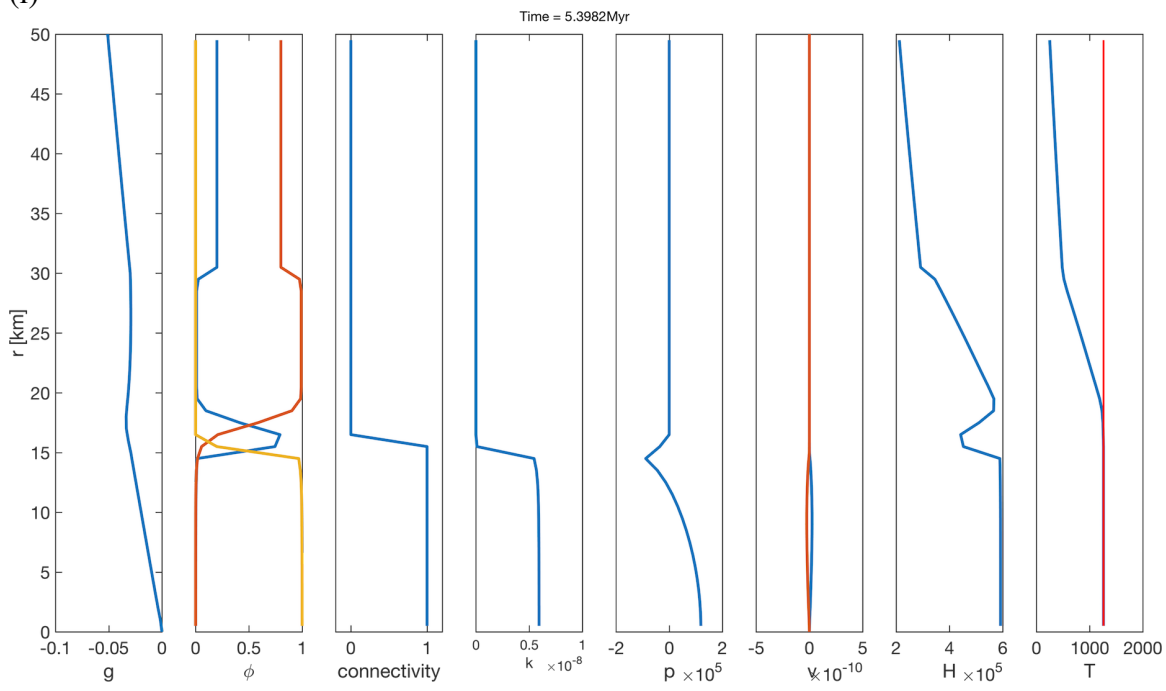
(j)


Time = 4.1218Myr
r [km]
g
ϕ
connectivity
k ×10^-8
p ×10^6
V ×10^-10
H ×10^5
T

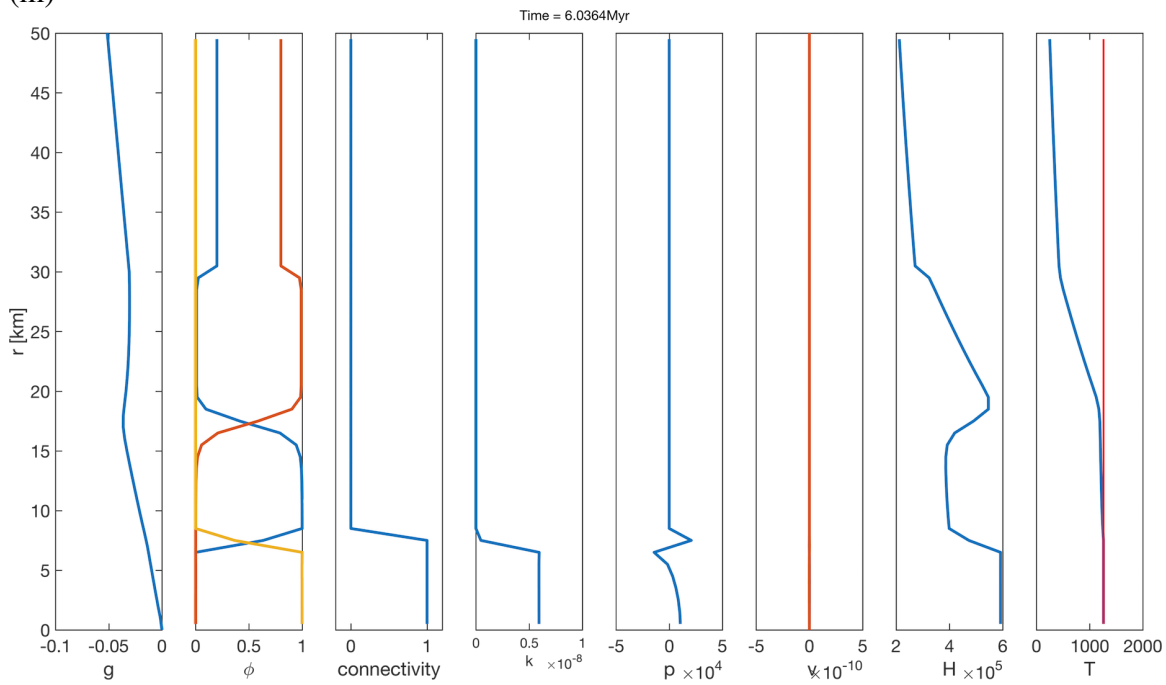
(k)


Time = 4.76Myr
r [km]
g
ϕ
connectivity
k $\times 10^{-8}$
p $\times 10^{6}$
v $\times 10^{-10}$
H $\times 10^{5}$
T

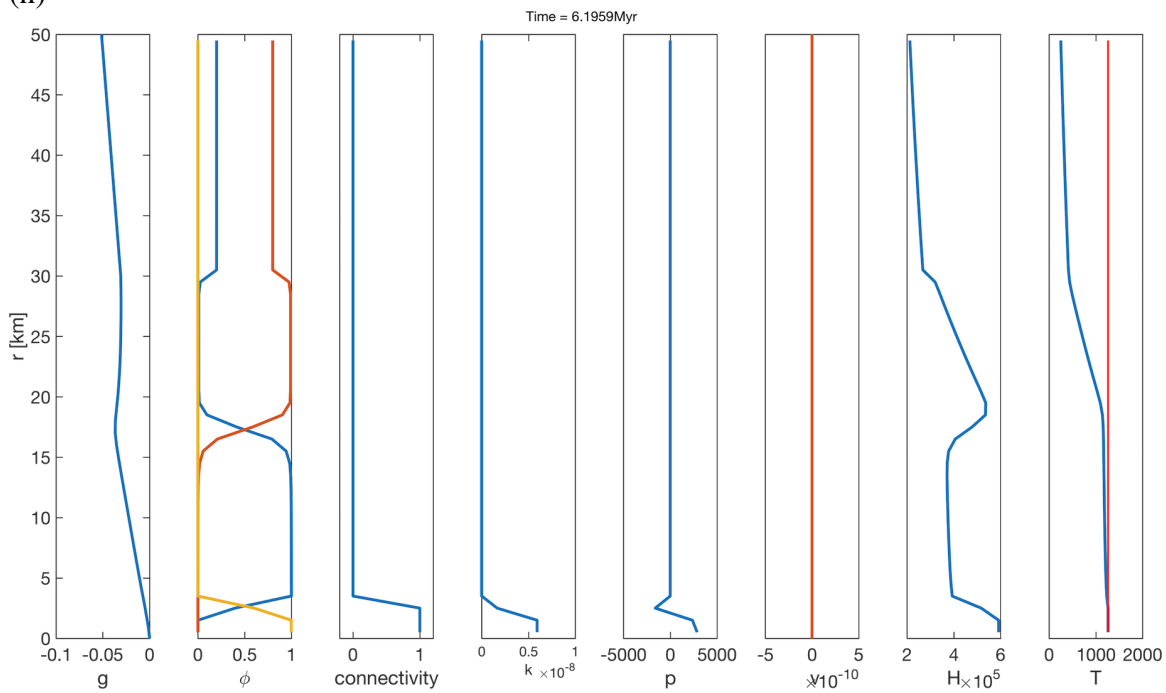
(l)


Time = 5.3982Myr
r [km]
g
ϕ
connectivity
k $\times 10^{-8}$
p $\times 10^{5}$
v $\times 10^{-10}$
H $\times 10^{5}$
T

(m)


Time = 6.0364Myr
r [km]
g
ϕ
connectivity
k ×10^-8
p ×10^4
v ×10^-10
H ×10^5
T

(n)


Time = 6.1959Myr
r [km]
g
ϕ
connectivity
k ×10^-8
p
v ×10^-10
H ×10^5
T

(o)

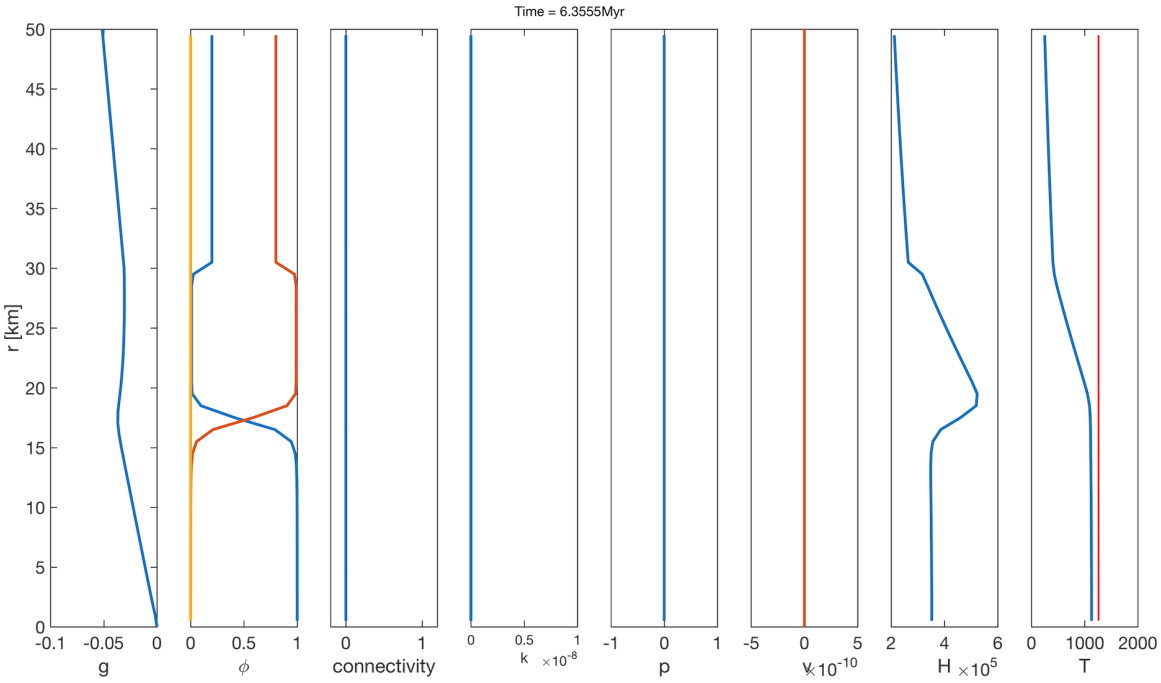


Figure 6.4: From left to right in each plot, 1: gravity, 2: volume fraction of molten iron (yellow), volume fraction of solid iron (blue) and volume fraction of olivine (red), 3: Connectivity of melt considering hysteresis, 4: permeability, 5: overpressure, 6: velocity of solid and relative Darcy flux, 7: enthalpy, 8: temperature. The unit of all variables is in SI, and connectivity is considered to be 0 or 1. The time, in million of years is shown in top of each figure.

For a planetesimal of 50 km radius, the efficiency of differentiation depends strongly on the time of accretion due to the rapid decay of 26Al (Fig. 6.5*a*). Only early planetesimals allow the formation of a distinct core surrounded by a silicate mantle containing 1-2% of trapped metal. This process is completed within 3-5 Myr of CAI and almost 2.2 Myr after the onset of melting (Fig. 6.5*b*). This demonstrates that percolative core formation with hysteresis can occur rapidly.

As the accretion time increases, the heat source decays before the segregation of interconnected melt is completed. This leaves increasing amounts of connected solidi-

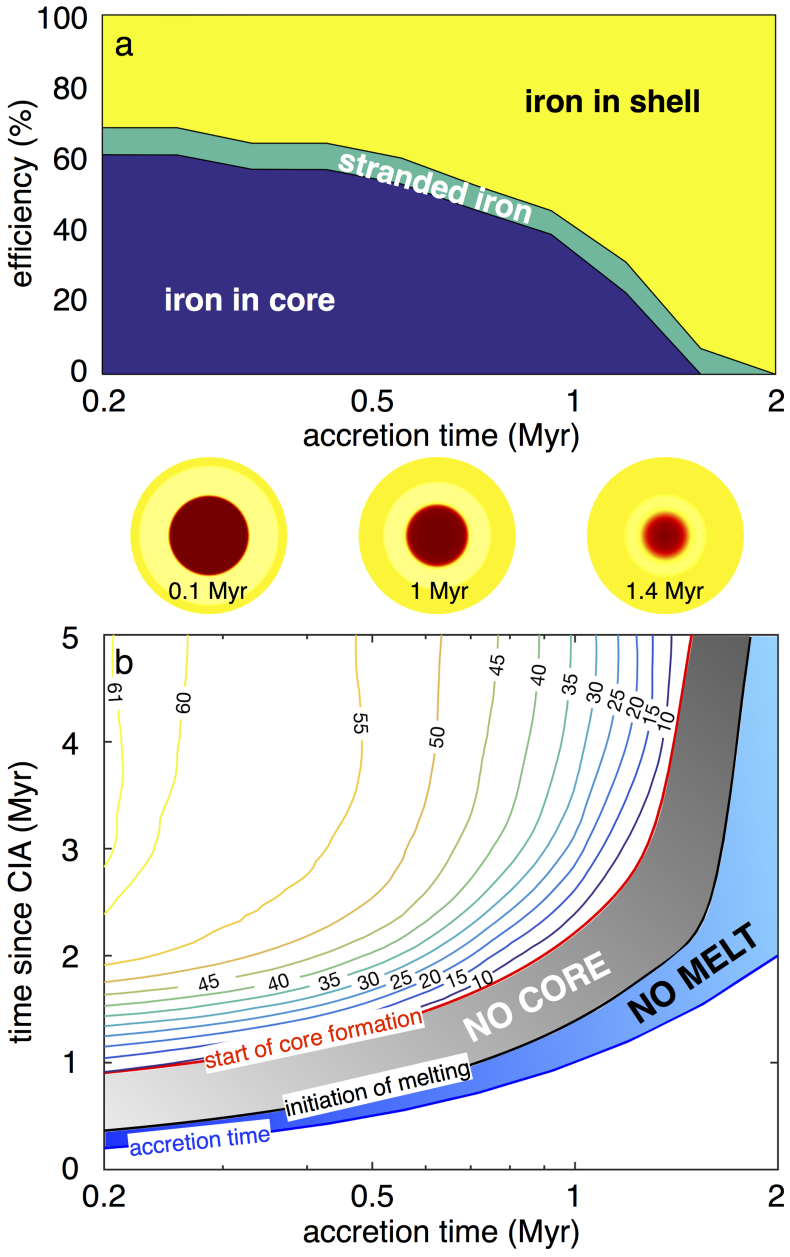


Figure 6.5: Time scales and iron segregation efficiency for a planetesimal with radius of 50 km. (a) Effect of accretion time on core formation efficiency, amount of stranded iron and size of not-molten shell. (b) The time taken from accretion time (blue curve) to initiate melting is shown with blue area. The gray area shows the time that taken from initiation of melting to initial time of formation of a distinctive core. Contours show the percentage of the total iron in planet that is segregated to core.

fied metal stranded in the silicate mantle, resulting in a smaller core. In late planetesimals, segregation of melt may begin, but subsides without ever forming a distinct core (Fig. 6.5*a*). Meteorite NWA 2993 may represent an example of an interconnected metal network stranded due to solidification. Similarly, pallasites may represent high melt fraction samples of stranded metal that formed closer to the center of the planetesimal.

Chapter 7

Conclusions and Recommendation For Future Work

7.1 Conclusion

7.1.1 Level Set Method for Ductile Materials with Texturally Equilibrated Pores

We presented a novel level set model to compute the liquid-solid interface in a porous medium at textural equilibrium. The geometric flexibility of the level set method overcomes the difficulties posed by the complex three-dimensional topology of the interface and allows the simulation in realistic geometries. The computational cost of the numerical method is much higher than previous methods based on explicit computation of the interface. To reduce the computational cost, we have introduced the domain decomposition while calculating coupling terms on original computational domain. This results in enormous improvement simulation time, enabling us to run the simulation with more than 1000 grains on desktop/notebook computer. While high-accuracy numerical discretization allows computations on relatively coarse grids, the results have to be interpolated to a fine grid for subsequent visualization. Three dimensional simulation results show the level set approach developed here is applicable to polycrystalline materials comprised of grains with arbitrary shapes. This method will enable the exploration of a large variety of different phenomena and complements tomographic studies of texturally equilibrated materials.

7.1.2 Properties of Ductile Rocks with Texturally Equilibrated Pores

The results presented here demonstrate the complexity and richness of texturally equilibrated pore networks. In particular, we have demonstrated the existence of wetted

grain boundaries at low porosities and finite dihedral angles as well as the dramatic effect the geometry of the pore network has on the physical properties of the medium, such as the permeability. We showed that grain boundary wetting is likely to be more common in disordered media and therefore also the induced permeability anisotropy and the increased percolation threshold for $\theta > 60^{\circ}$ in anisotropic systems. We presented the liquid configuration in textural equilibrium in an irregular media comprised of grains with different shapes and sizes. Furthermore, we presented the hypothesis of hysteresis in pore network topology and the implication it has in natural systems. The variation of permeability and electrical conductivity is also computed, presented and discussed. The pore scale simulations combined with either artificial crystal configurations or those available from an imaging study can provide a case-specific computation of constituent permeability-porosity-dihedral angle relationship.

7.1.3 Pore-Scale Experimental Study of Rock Salt

To experimentally characterize the pore network in a texturally equilibrated ductile rock, we conducted a series of experiments in different pressures and temperatures on synthetic rock salt. The brine network (pore fluid) is constructed with X-ray microtomography and connectivity of pore space is established with image analysis algorithm. The experimental results confirm the static pore-scale theory in undrained laboratory experiments on synthetic salt samples that have been imaged with non-destructive X-ray microtomography after quenching to ambient conditions. These results confirm the first-order control of the dihedral angle on brine percolation.

7.1.4 Fluid Percolation in Ductile Rock Salt in Gulf of Mexico

The static experiments serve as a baseline for the field observations of fluid distributions in deformed rock salt. These field observations have implications for ensuring

hydrological isolation of nuclear waste in rock salt. At the relatively shallow depth typically considered for geological storage, the dihedral angle is between 65° and 72° and should prevent brine percolation in rock salt, based on static pore-scale theory and experiments. However, field observations reported here show that such moderate dihedral angles do not guarantee hydrological isolation in deformed rock salt. The deformation-assisted percolation observed in salt sections of petroleum wells is not associated with man-made excavations, suggesting this mechanism is not limited to the vicinity of the repository site and the duration of room closure around the waste. Lower differential stresses recorded in shallow bedded rock salt suggest it is more likely to provide an impermeable barrier. However, tectonic forces and excavations can result in high stresses in shallow cold salt. Therefore, it is important to characterize the salt microstructure of potential repositories to determine the stress history, state of grain boundaries and the fluid distribution. Future work should also constrain the permeability that can be generated by deformation-assisted percolation and its persistence.

7.1.5 Percolative Core Formation Due to Hysteresis in Melt Connectivity

Our results show that metal-silicate differentiation by porous flow involves rich dynamical phenomena that can lead to a variety of outcomes. Understanding the efficiency and time scales of planetesimal differentiation requires a multi-scale approach that integrates the pore-scale physics that determine pore network connectivity with large scale transport phenomena that govern the segregation process. The hysteresis in melt network connectivity demonstrated here highlights the importance of determining realistic percolation thresholds for planetary materials. Determining the dihedral angle alone is not sufficient information to constrain the likelihood of percolative core formation.

7.2 Recommendations For Future Work

Below I outline a path for future research and further study of the ductile rocks with texturally equilibrated pores. I divide my vision for next steps in three subsections: complementing the computational study, especially the level set method, extending experimental study on static and also deforming rock salt and broadening the field study to other parts of Gulf of Mexico and other salt bodies including the pre-salt in offshore Brazil.

7.2.1 Computational Research

The level set method is specifically designed for ductile rocks with texturally equilibrated pores. There might be some interest to complement and enhance the method by including more physics, i.e. effects of shear forces or shear stress, or effect of gravity on pore shapes and connectivity. The computational model on planetary core formation might be extended to include more physics and a larger variety of materials and parameters.

7.2.2 Experimental Study

Static experiments on rock salt should be extended to a wider range of pressure and temperature and larger number of synthetic samples. Experimental work on dynamically deforming rock salt may also be done to characterize the effect of deformation-assisted fluid percolation. There might be interest in core-flood (in form of gas permeability) experiments in salt to further evaluate the connectivity of pore space.

7.2.3 Field Study

The presented field analysis of the deforming ductile rock salt in Gulf of Mexico can be extended to other protraction areas in Gulf, including Alaminos Canyon, Garden Banks and Keathley Canyon. A large area offshore Brazil is covered with pre-salt formations. The

area is very well penetrated by petroleum wells and they also provide a very good study opportunity. The effects of the rubble zone and deformation-assisted fluid percolation may be further analyzed. Any permeability structure in rock salt must be reassessed in petroleum exploration and subsurface waste storage.

Appendices

Appendix A

Analytical Solution for Textural Equilibrium Problem in 2-D

A.1 Problem Statement

Equilibrium state can be achieved when the total energy (expressed by Gibbs Energy Function) is minimum. It is believed that the energy minimization is equivalent to area minimization. In other words, a meniscus is in equilibrium when its area (under certain conditions of pressure, volume and interfacial tension) is minimized. For a 2-D problem we can say that energy is minimum when the perimeter of meniscus is minimum in defined condition.

A.2 Mean Curvature

The following section is a quote from book by Oprea (2000): We can define the mean curvature of function H by the relation

$$2H = k_1 + k_2, \tag{A.1}$$

where k_1 and k_2 are the normal curvatures associated to any two perpendicular tangent vectors. Considering a surface given by $z = f(x, y)$ be a function of two variables and taking a Monge parameterization for its graph: $\mathbf{x}(u, v) = (u, v, f(u, v))$. We have

$$\begin{aligned} \mathbf{x}_u &= (1, 0, f_u), & \mathbf{x}_u u &= (0, 0, f_{uu}), \\ \mathbf{x}_v &= (1, 0, f_v), & \mathbf{x}_u v &= (0, 0, f_{uv}), \\ & & \mathbf{x}_v v &= (0, 0, f_{vv}), \end{aligned}$$

$$\mathbf{x}_u \times \mathbf{x}_v = (-f_u, -f_v, 1) \qquad \Longrightarrow \qquad U = \frac{(-f_u, -f_v, 1)}{\sqrt{1 + f_u^2 + f_v^2}},$$

$$E = 1 + f_u^2, \qquad F = f_u f_v, \qquad G = 1 + f_v^2$$

$$L = \frac{f_{uu}}{\sqrt{1 + f_u^2 + f_v^2}}, \qquad M = \frac{f_{uv}}{\sqrt{1 + f_u^2 + f_v^2}}, \qquad N = \frac{f_{vv}}{\sqrt{1 + f_u^2 + f_v^2}}$$

$$H = \frac{(1 + f_v^2) f_{uu} + (1 + f_u^2) f_{vv} - 2 f_u f_v f_{uv}}{2(1 + f_u^2 + f_v^2)^{\frac{3}{2}}}$$

Theorem A.2.1. *Surface M given by the graph of $z = f(x, y)$ is minimal if and only if*

$$(1 + f_v^2) f_{uu} - 2 f_u f_v f_{uv} + (1 + f_u^2) f_{vv} = 0.$$

This equation is called the *minimal surface equation*, and, in general is not solvable by simple means. However, we can sometimes hypothesize algebraic or geometric requirements about the function f which allow us to solve the minimal surface equation and thereby determine certain types of minimal surfaces.

But there are situations where mean curvature is non-zero, but constant. The first example which springs to mind is a sphere because, at each point, the normal curvatures are the same in any direction. Therefore, the mean curvature is also this constant normal curvature.

Theorem A.2.2. *A closed surface which minimizes the surface area subject to a fixed enclosed volume must have constant mean curvature. If M is a compact surface of constant mean curvature embedded in $\mathbb{R}^3$, then M is a standard sphere. The equivalent domain in $\mathbb{R}^2$ has the form of a circle.*

A.3 General Variational Problem

The following section is a quote from book by Oprea (2000): The standard variational problem having extra constraints has the form:

$$Minimize \int_{x_0}^{x_1} f(x, y, y')\, dx \tag{A.2}$$

Subject to the endpoint conditions $y(x_0) = y_0$, $y(x_1) = y_1$ and the requirement that

$$I = \int_{x_0}^{x_1} g(x, y, y')\, dx = c, \tag{A.3}$$

where c is a constant.

We will now derive a form of the Euler-Lagrange equation which is necessary condition for the solution of the constrained problem. Just as before, assume $y = y(x)$ is a minimizer for the problem and take a variation $\widehat{y} = y+\epsilon(a\eta+b\xi)$, where $\eta(x_0) = 0 = \eta(x_1)$ and $\xi(x_0) = 0 = \xi(x_1)$. We take two perturbations η and ξ, because taking only one would not allow us to vary J while holding I constant. By taking η and ξ, we can vary J while offsetting the effects of one perturbation with the other in I. The usual Euler-Lagrange argument gives

$$0 = \left.\frac{dJ}{d\epsilon}\right|_{\epsilon=0} = \int_{x_0}^{x_1} (a\eta + b\xi) \left(\frac{\partial f}{\partial y} - \frac{d}{dx}\left(\frac{\partial f}{\partial y'}\right)\right) dx. \tag{A.4}$$

Now, however, the usual argument must be modified because $\eta + \xi$ is not arbitrary. The requirement $I = c$ puts restrictions on a and b. If we carry through the argument above for I

$$0 = \left.\frac{dI}{d\epsilon}\right|_{\epsilon=0} = \int_{x_0}^{x_1} (a\eta + b\xi) \left(\frac{\partial g}{\partial y} - \frac{d}{dx}\left(\frac{\partial g}{\partial y'}\right)\right) dx \tag{A.5}$$

where the derivative is zero since $I = c$ is a constant. But y is not an extermal for I, so the Euler-Lagrange equation with g does not hold. Instead, we have

$$\int_{x_0}^{x_1} (a\eta + b\xi) \left(\frac{\partial f}{\partial y} - \frac{d}{dx}\left(\frac{\partial f}{\partial y'}\right)\right) dx = 0, \tag{A.6}$$

$$\int_{x_0}^{x_1} (a\eta + b\xi) \left(\frac{\partial g}{\partial y} - \frac{d}{dx}\left(\frac{\partial g}{\partial y'}\right)\right) dx = 0, \tag{A.7}$$

and solving each equation for a/b gives

$$-\frac{\int_{x_0}^{x_1} \xi \left(\frac{\partial g}{\partial y} - \frac{d}{dx}\left(\frac{\partial g}{\partial y'}\right)\right) dx}{\int_{x_0}^{x_1} \eta \left(\frac{\partial g}{\partial y} - \frac{d}{dx}\left(\frac{\partial g}{\partial y'}\right)\right) dx} = \frac{a}{b} = \frac{\int_{x_0}^{x_1} \xi \left(\frac{\partial f}{\partial y} - \frac{d}{dx}\left(\frac{\partial f}{\partial y'}\right)\right) dx}{\int_{x_0}^{x_1} \eta \left(\frac{\partial f}{\partial y} - \frac{d}{dx}\left(\frac{\partial f}{\partial y'}\right)\right) dx}. \tag{A.8}$$

Upon rearranging we obtain

$$-\frac{\int_{x_0}^{x_1} \xi \left(\frac{\partial g}{\partial y} - \frac{d}{dx}\left(\frac{\partial g}{\partial y'}\right)\right) dx}{\int_{x_0}^{x_1} \xi \left(\frac{\partial f}{\partial y} - \frac{d}{dx}\left(\frac{\partial f}{\partial y'}\right)\right) dx} = \frac{\int_{x_0}^{x_1} \eta \left(\frac{\partial f}{\partial y} - \frac{d}{dx}\left(\frac{\partial f}{\partial y'}\right)\right) dx}{\int_{x_0}^{x_1} \eta \left(\frac{\partial g}{\partial y} - \frac{d}{dx}\left(\frac{\partial g}{\partial y'}\right)\right) dx}. \tag{A.9}$$

The left-hand side is a function of ξ while the right is a function of η, so the only way for the expressions to be identical is for both expressions to be equal to the same constant λ (*the Lagrange Multiplier*). Simplifying the equation

$$\frac{\int_{x_0}^{x_1} \eta \left(\frac{\partial f}{\partial y} - \frac{d}{dx} \left(\frac{\partial f}{\partial y'} \right) \right) dx}{\int_{x_0}^{x_1} \eta \left(\frac{\partial g}{\partial y} - \frac{d}{dx} \left(\frac{\partial g}{\partial y'} \right) \right) dx} = \lambda \tag{A.10}$$

gives

$$\int_{x_0}^{x_1} \eta \left[\frac{\partial (f - \lambda g)}{\partial y} - \frac{d}{dx} \left(\frac{\partial (f - \lambda g)}{\partial y'} \right) \right] dx = 0. \tag{A.11}$$

Because η is arbitrary, the previous equation can hold only if the term in the brackets vanishes. We thus obtain the following *Euler-Lagrange necessary condition* for the constrained problem (Canham, 1970; Oprea, 2007, 2000; Courant and Hilbert, 1989; Deuling and Helfrich, 1976b,a; Hildebrandt and Tromba, 1985; Paulsen, 1994; Sagan, 1992; Thompson and Biology, 1992; Troutman, 1995; Weinstock, 1974).

Theorem A.3.1. *If $y = y(x)$ is a solution to the standard constrained problem,*

$$Minimize \int_{x_0}^{x_1} f(x, y, y')\, dx$$

such that

$$I = \int_{x_0}^{x_1} g(x, y, y')\, dx = c,$$

then

$$\frac{\partial (f - \lambda g)}{\partial y} - \frac{d}{dx} \left(\frac{\partial (f - \lambda g)}{\partial y'} \right) = 0. \tag{A.12}$$

A.4 2D Symmetric Geometry

We look at the two dimensional symmetric problem in here by three methods. First we use the theorem A.2.2 along with 2 constraints: Definite area (in 3D Volume or pore space) and definite contact angle.

A.4.1 Circle solution

From Figure A.1 we can write below equations:

$$y_2 = x_2 \tan\left(\frac{\pi}{3}\right) \tag{A.13}$$

$$y_3 = x_3 \tan\left(\frac{\pi}{3}\right) \tag{A.14}$$

$$(x_1 - x_3)^2 + (y_1 - y_3)^2 = R^2 \tag{A.15}$$

$$(x_2 - x_3)^2 + (y_2 - y_3)^2 = R^2 \tag{A.16}$$

$$A = \frac{1}{2}x_1 y_3 - \frac{\gamma}{2}R^2 \tag{A.17}$$

On the other hand, contact angle should be θ, and we assumed curve C be part of a circle, we have

$$y' = -\frac{x - x_3}{y - y_3}$$

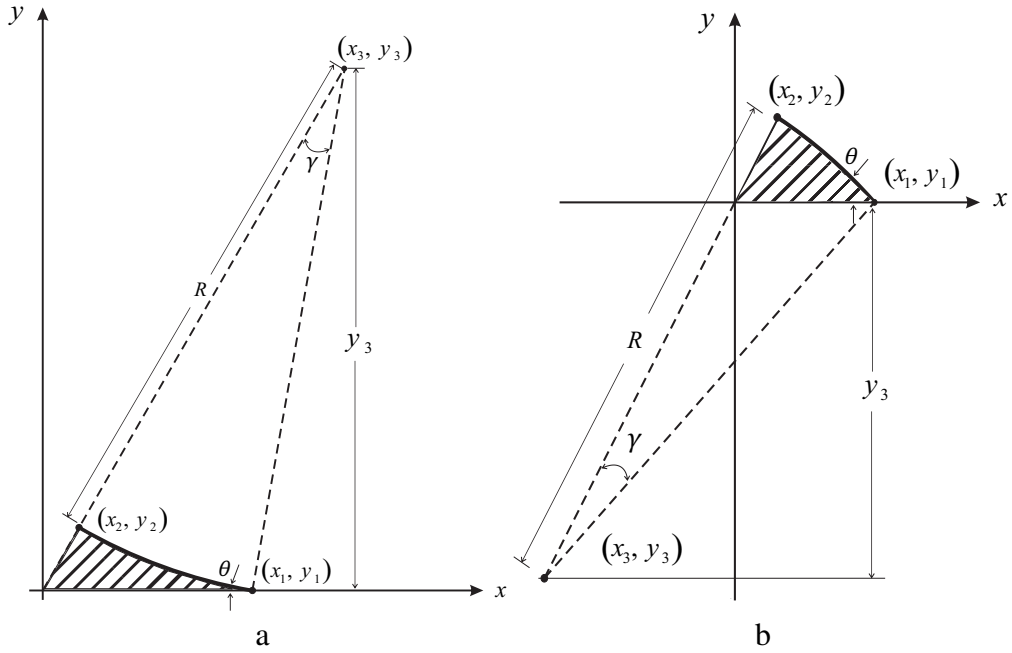


Figure A.1: Unknown parameters in circle solution assumption. a) $\theta \leq 60°$, b) $\theta > 60°$

$$\tan(\pi - \theta) = -\frac{x_1 - x_3}{y_1 - y_3} \tag{A.18}$$

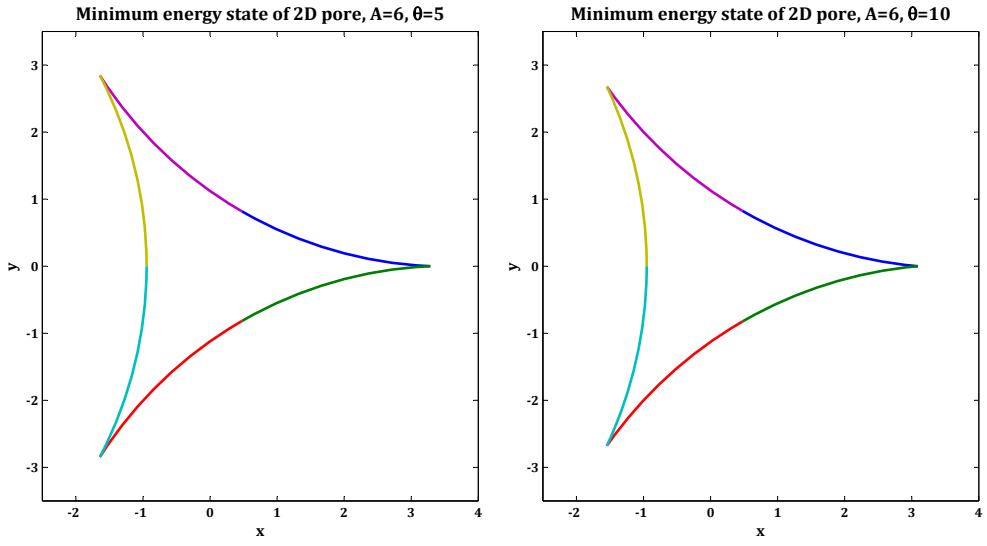
for the central angle (γ) we have

$$\sin\left(\frac{\gamma}{2}\right) = \frac{1}{2}\left(\frac{\sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}}{R}\right)$$

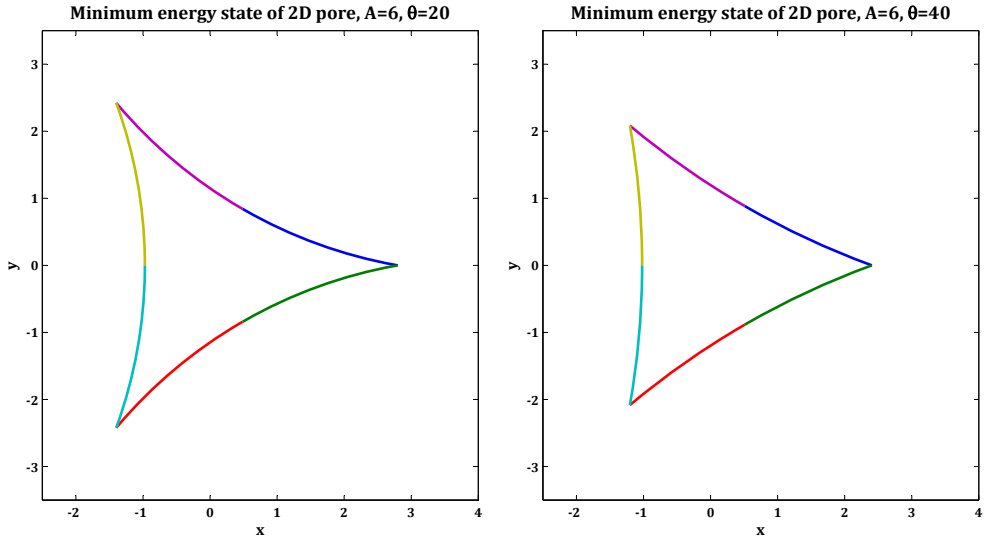
and γ will be

$$\gamma = 2\sin^{-1}\left(\frac{\sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}}{2R}\right)$$

substituting this into equation for area, we come up with



a b



c d

$$A = \frac{1}{2}\left|x_1 y_3\right| - R^2 \sin^{-1}\left(\frac{\sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}}{2R}\right) \tag{A.19}$$

Similarly for contact angle more than 60° the area can be represented as

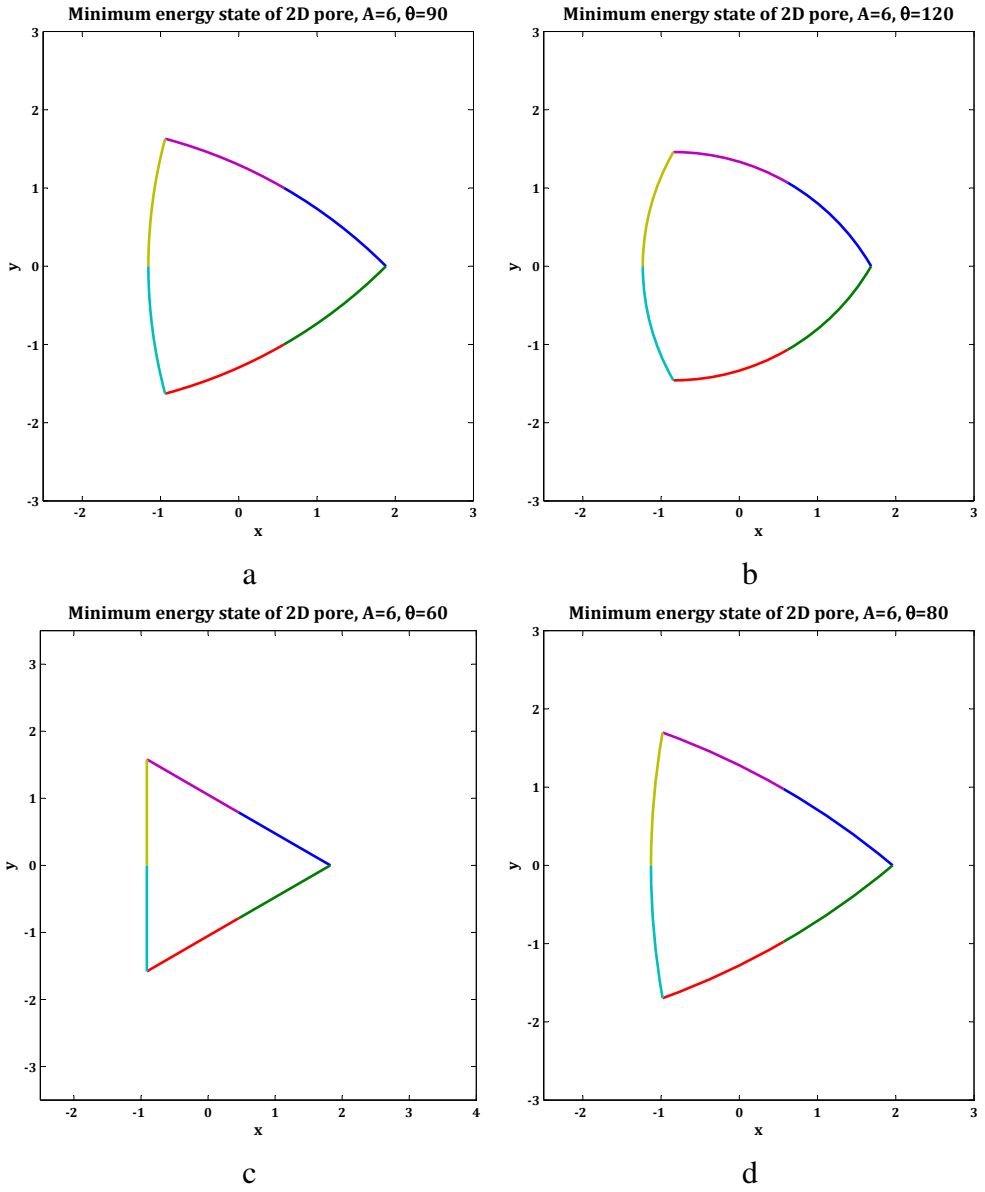


Figure A.2: Equilibrium state for a two dimension (2D) pore

$$A = R^2 \sin^{-1}\left(\frac{\sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}}{2R}\right) - \frac{1}{2}\left|x_1 y_3\right| \tag{A.20}$$

Solving equations A.13, A.14, A.15, A.16, A.18, A.19 and A.20 together we can get the equilibrium state for a two dimension (2D) pore by obtaining x_1, x_2, y_2, x_3, y_3 and R.

Figure A.2 shows some results for $A = 1$ and $\theta = 5, 10, 20, 40, 60, 80, 90$ and $120°$.

A.4.2 Von-Bargen Method

Implementing Von Bargen method to obtain textural equilibrium for a symmetric 2D case needs an initial guess for curve in v-nodes (in 3D, we need to guess a surface in $u - v$ plane nodes). In beginning of every step of iteration the boundary conditions are applied to constraint the curve to intersect the v axis with half of the wetting angle and also satisfy the continuity of derivatives at left boundary.

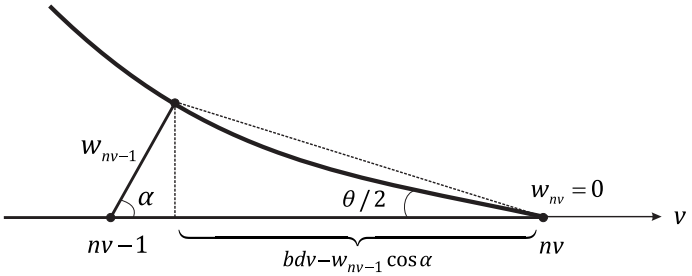


Figure A.3: Zoom on the right boundary condition (dihedral edge)

$$w_{nv} = 0$$

$$\tan\frac{\theta}{2} = \frac{w_{nv-1} sin\alpha}{bdv - w_{nv-1}\cos\alpha}$$

$$w_{nv-1} = \frac{\tan\frac{\theta}{2}}{\sin\alpha + \cos\alpha\tan\frac{\theta}{2}}\, b\, dv$$

where θ is the wetting angle. Then the curvature will be calculated

$$b \neq f(u) \qquad \Rightarrow \qquad b' = b'' = b''' = 0 \qquad \Rightarrow$$

So

$$\cos\gamma = 0 \qquad \Rightarrow \qquad \gamma = \frac{\pi}{2} = const$$

and

$$\cos\alpha = \frac{1}{2} \qquad \Rightarrow \qquad \alpha = \frac{\pi}{3} = const$$

Therefore we have $\alpha' = 0$, $\alpha'' = 0$, $\gamma' = 0$ and $\gamma'' = 0$, substtuting all above, we will come up with

$$\vec{s}_u = 1\,\vec{i}$$

$$\vec{s}_v = (b + w'\cos\alpha)\vec{j} + (w'\sin\alpha)\vec{k}$$

$$\vec{n} = (-w'\sin\alpha)\,\vec{j} + (b + w'\cos\alpha)\,\vec{k}$$

$$\vec{s}_{uu} = 0$$

$$\vec{s}_{vv} = (-w''\cos\alpha)\,\vec{j} + (w''\sin\alpha)\,\vec{k}$$

$$\vec{s}_{uv} = 0$$

$$E = 1$$

$$G = (b + w' \cos\alpha)^2 + (w' \sin\alpha)^2$$

$$F = 0$$

$$L = 0$$

$$N = -w'' \cos\alpha \, (w' \sin\alpha) + w'' \sin\alpha \, (b + w' \cos\alpha)$$

$$M = 0$$

$$\left|\vec{n}\right| = \left((-w' \sin\alpha)^2 + (b + w' \cos\alpha)^2\right)^{\frac{1}{2}}$$

so the curvature in $u - v - w$ coordinate is represented by

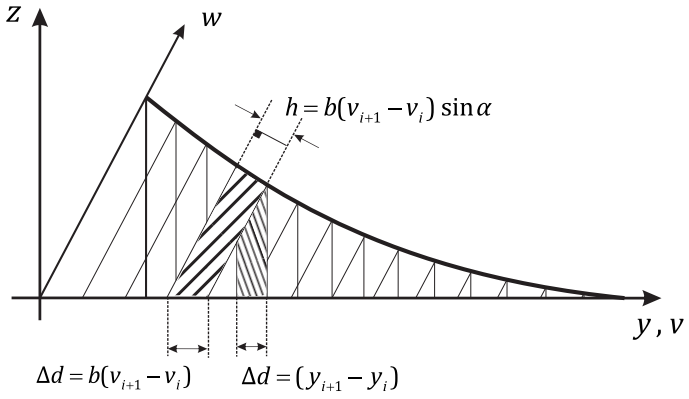


Figure A.4: Von Bargen coordinate for a symmetric 2D case

$$\overline{\kappa} = \frac{1}{\left|\vec{n}\right|}\frac{E}{2G}$$

$$= \frac{1}{2\Big(\left(b + w'\cos\alpha\right)^2 + \left(w'\sin\alpha\right)^2\Big)\left(\left(-w'\sin\alpha\right)^2 + \left(b + w'\cos\alpha\right)^2\right)^{\frac{1}{2}}}$$

and curve will be updated as follow:

$$\Delta w_j = \begin{cases} k\ln\left(1+\overline{\kappa}_j\right) & \text{if } \overline{\kappa}_j \geq 0 \\ -k\ln\left(1-\overline{\kappa}_j\right) & \text{if } \overline{\kappa}_j < 0 \end{cases} \tag{A.21}$$

As the process should be so slow, k has been chosen to be $0.00001 < k < 0.001$. After updating the curve using equation A.21 area is calculated using trapezoidal rule and m should be chosen such that the area remain constant (equal to given area) to ensure the constant porosity (or melt fraction).

$$A = \frac{1}{2}\left|y_1 z_1\right| + \sum_{i=1}^{nv}\frac{1}{2}\left(y_{i+1} - y_i\right)\left(z_{i+1} + z_i\right)$$

where $y_i = bv_i + w_i\cos\alpha$ and $z_i = w_i\sin\alpha$. For having a fixed area, we add m to Δw (Eq. A.21) such that the area doesn't change (or converge to the desirable area). So m is chosen such that the area under the curve after update by Eq. A.21 be the target area. Then, as the curve goes up or down, the new b should be calculated. Using linear interpolation and extrapolation, we can fine the new b as follow

$$b_{new} = \begin{cases} y_{i+1} - z_{i+1}\dfrac{y_{i+1} - y_i}{z_{i+1} - z_i} & \text{if } \Delta w + m < 0 \\ oldb + \dfrac{z_{nv}}{\tan\frac{\theta}{2}} & \text{if } \Delta w + m > 0 \end{cases} \tag{A.22}$$

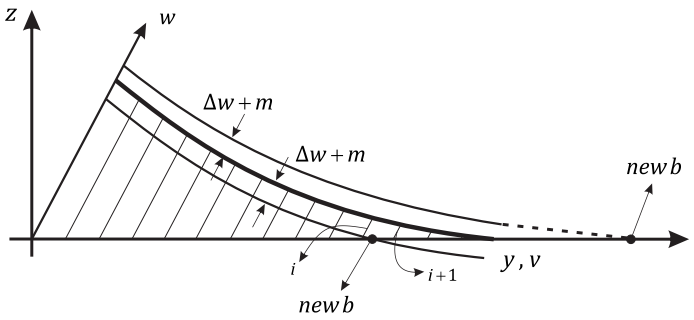


Figure A.5: Linear interpolation or extrapolation for finding new b

After finding new b, we should find the new location of grid points. This can be done using interpolation and extrapolation of the previous grid points. The interpolation is linear but extrapolation is quadratic. Regridding function also uses a curve fitting procedure to make the curve smooth. And then all these processes repeat in a loop to converge to a solution. Figure A.6 demonstrates the computational steps and their effect on the initial guess in first iteration of one case ($\theta = 10^\circ$, Target area=1). Figure A.7 shows the iteration steps for that case. As we see, the initial guess curve is a hyperbolic function and its area is bigger than 1.

The Figures A.8-A.10 illustrate change of 3 important variable during iterations. The iteration process has been set to go to reach 10^6 in number. Figure A.8 shows how b changes along iteration for one special case. As we see after 10^5 iterations, b is approximately constant. This trend is valid only for the case with $\theta = 10^\circ$, $A = 1$ and the specific initial guess which has been used.

Figure A.9 presents the value of area in different iteration steps. As it seems, the area is closed enough to target area ($= 1$) in less than 10 iterations.

Figure A.10 represents the contact angle at dihedral edge which is menat ot be 10° in this case. As it can be ssen, the contact angle tends to the target angle in less than 10 iterations.

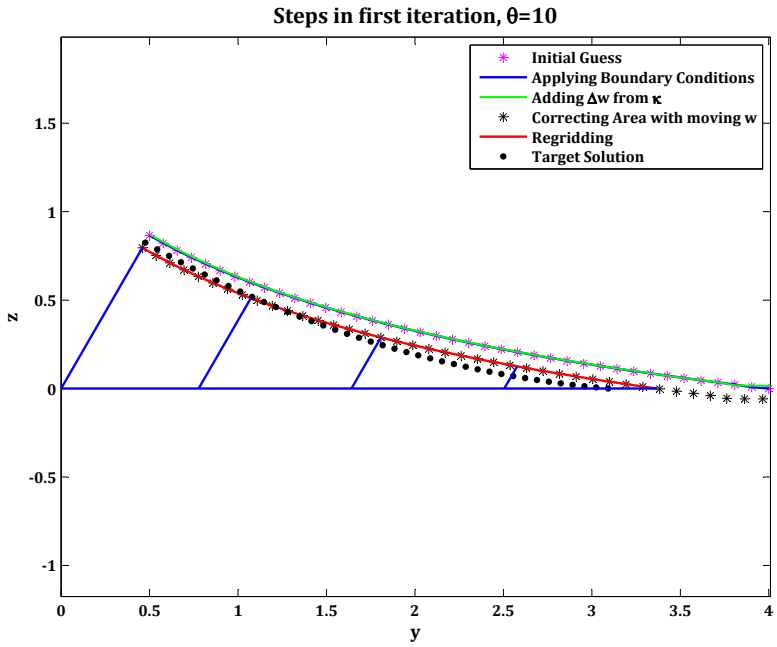


Figure A.6: The algorithm steps applied to initail guess in first iteration

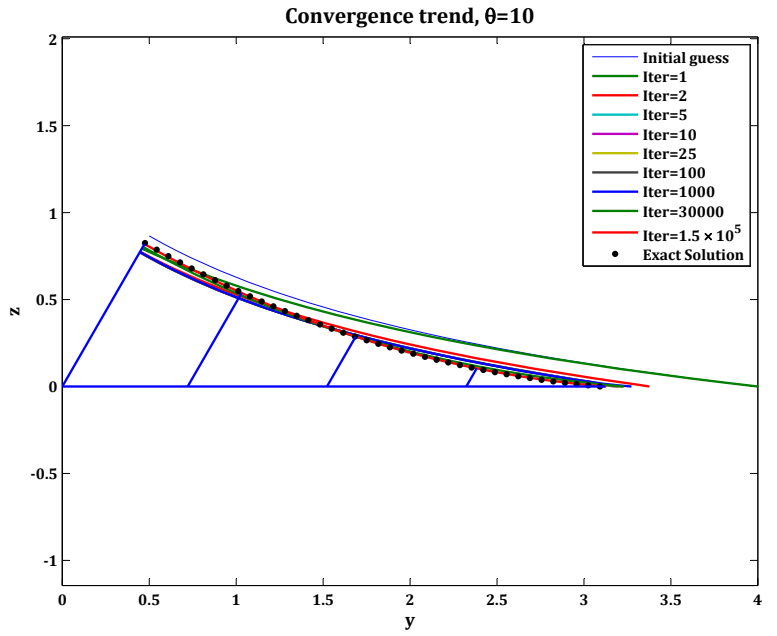


Figure A.7: Adjustments of equilibrium curve during iterative process

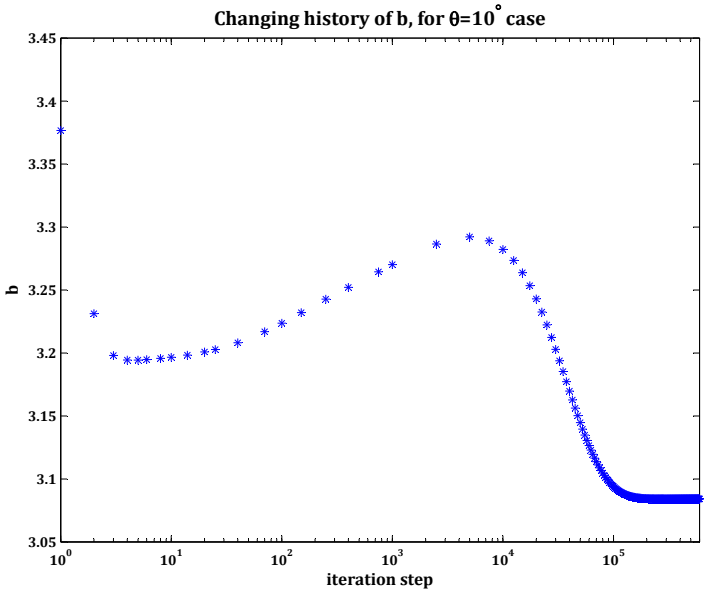


Figure A.8: Changes of b versus iteration steps

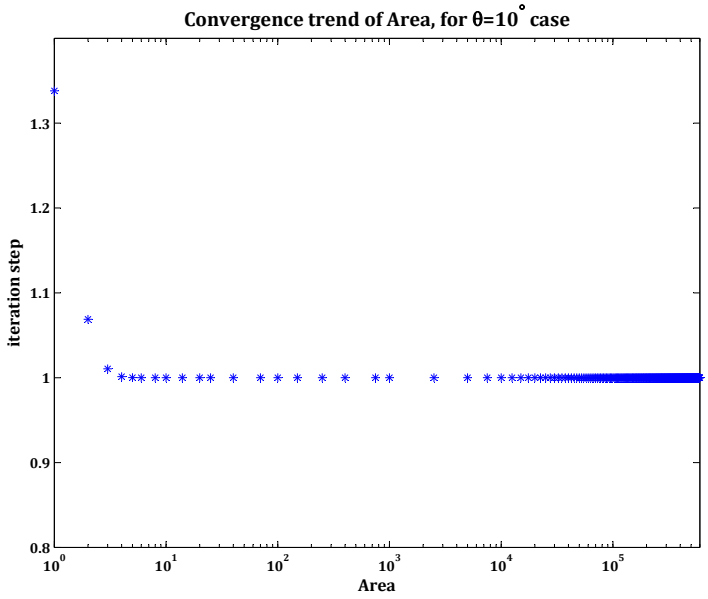


Figure A.9: Changes of area versus iteration steps

The curvature of final curve is plotted in Figure A.11 as a function of v. It shows that the curvature is constant in final solution (see Theorem A.2.2). We shall note that the Figures A.6-A.11 are valid only for case with a specific initial guess and the trend of

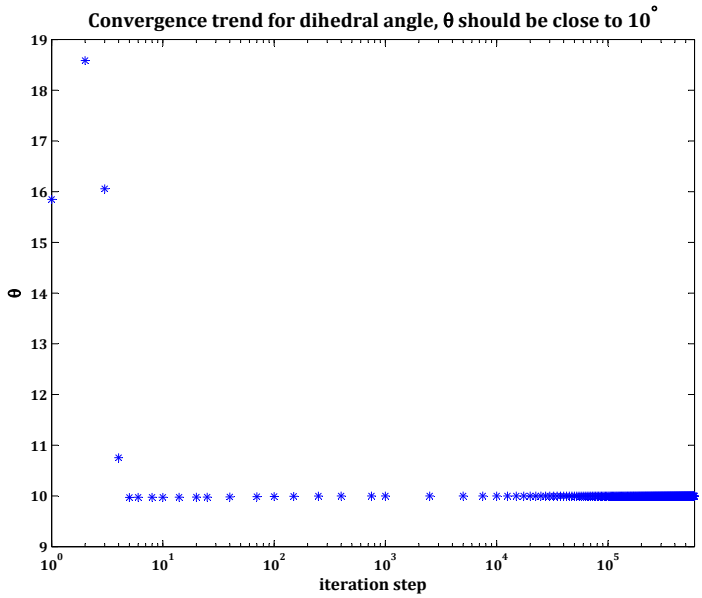


Figure A.10: Changes of θ (dihedral angle) versus iteration steps

changes might be different from case to case.

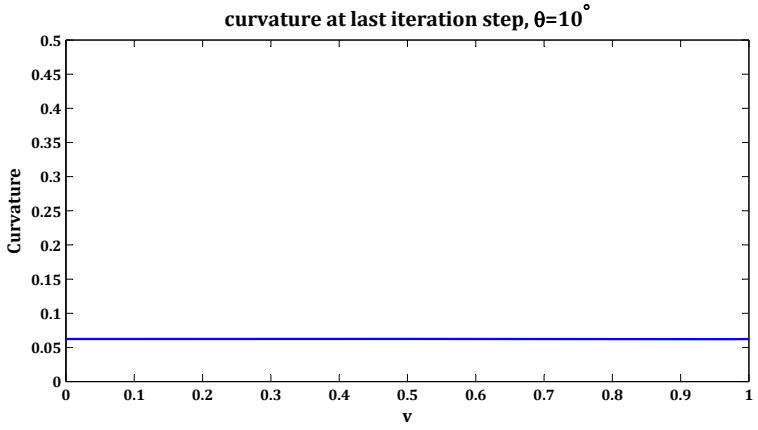
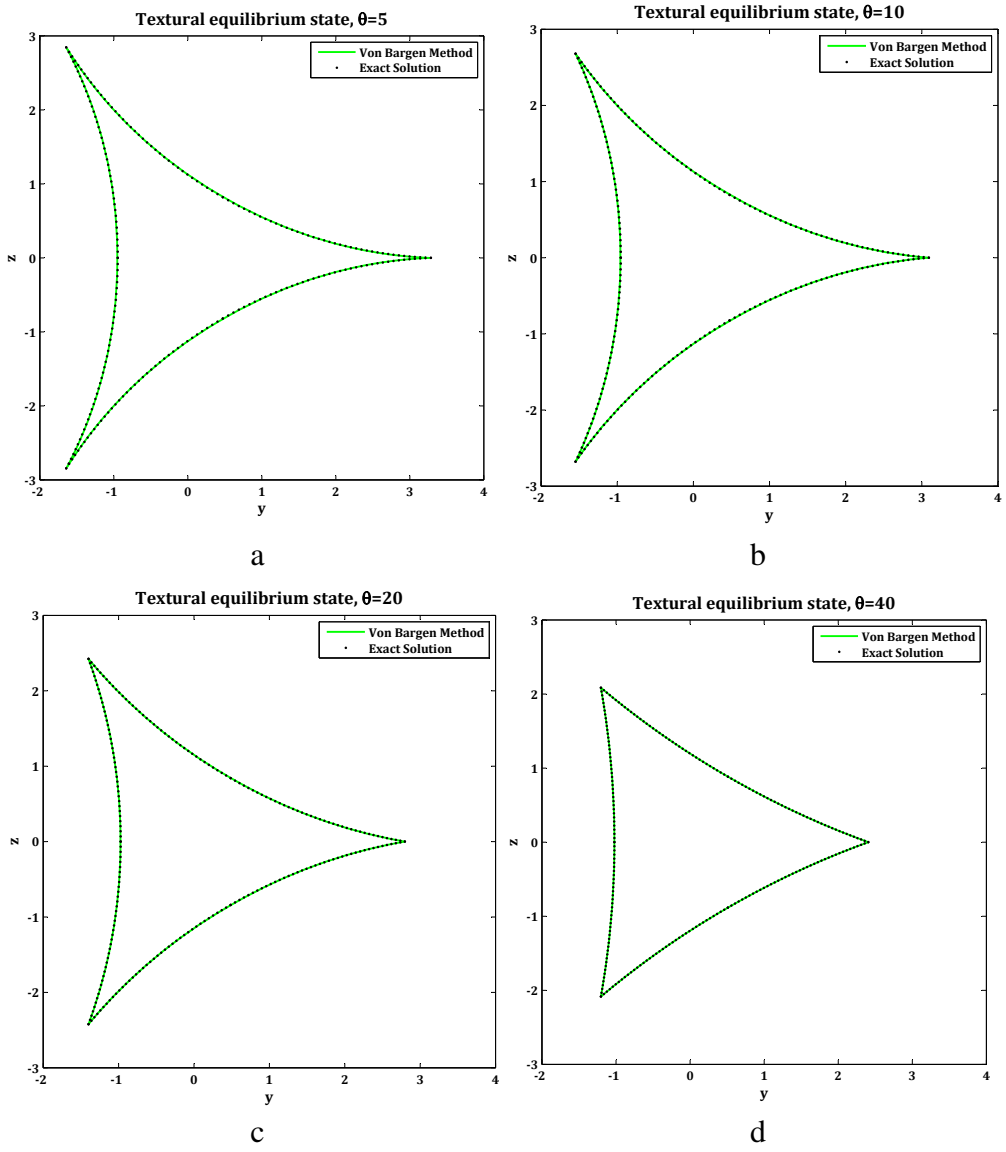


Figure A.11: Curvature as a function of v

Different contact angle cases are calculated using Von-Bargen method and are compared to the circle solution. As it can be seen in Figure A.12 the results are in perfect agreement to each other.

If we assume some conditions for convergence, we can see the effect of k at Eq.



A.21 on rate of convergence, the three important parameters are area, contact angle and b and we can say that we are close enough to answer if dA, $d\theta$ and db, defined below, are small enough. For 2D symmetric case we considered these criteria for convergence

$$dA = \frac{A - TargetArea}{TargetArea} < 10^{-6}$$

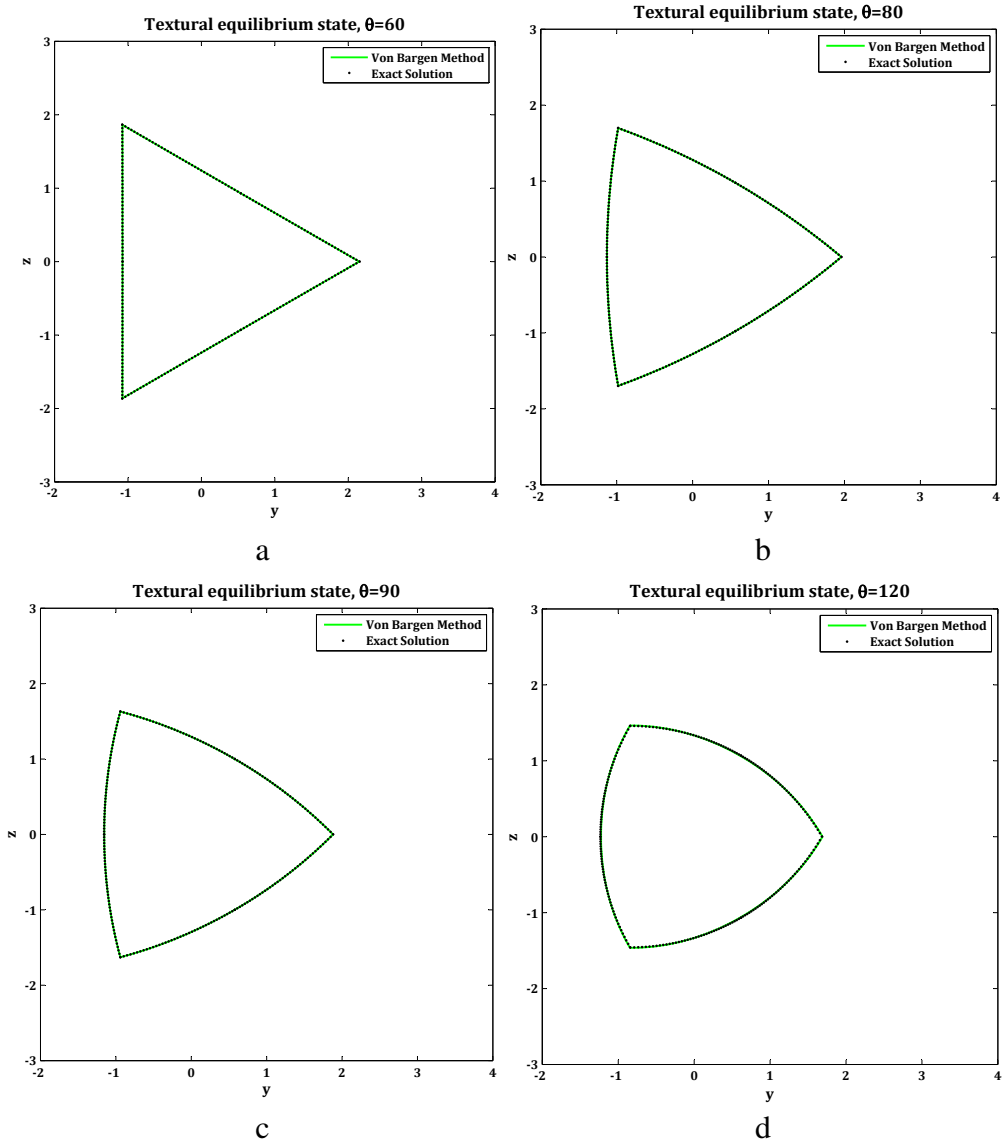


Figure A.12: Equilibrium state for a two dimension (2D) pore, using Von-Bargen method

$$d\theta = \left| \frac{2\tan^{-1}\left(\frac{z_{nv}-z_{nv-1}}{y_{nv-1}-y_{nv}}\right) - Target\theta}{Target\theta} \right| < 10^{-6}$$

$$db = \frac{b - newb}{newb} < 10^{-6}$$

Figure A.13 represents the number of iteration in which the solution is felt in the criteria mentioned above. For the special case of $\theta = 20^{\circ}$ for $k > 0.001$ the solution does not converge. The main reason is that the correction of curve due to curvature and area make each other neutral.

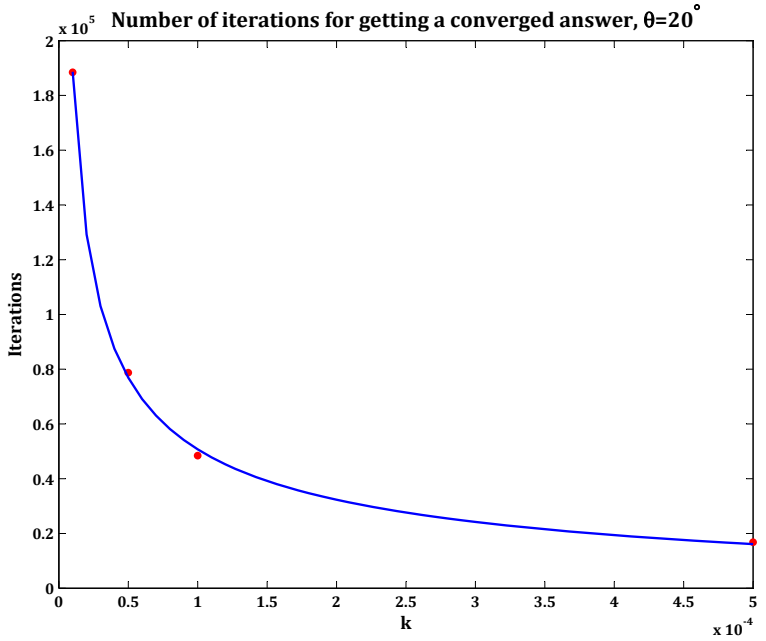


Figure A.13: Change of rate of convergence by changing k

A.4.3 Optimization using Lagrange method

Equilibrium state can be achieved when the total energy (expressed by Gibbs Energy Function) is minimum. It is believed that the energy minimization is equivalent to area minimization. In other words, a meniscus is in equilibrium when its area (under certain condition like pressure, volume and interfacial tension) is minimized. For a 2-D problem we can say that energy is minimum when the perimeter of meniscus is minimum in defined condition.

So our problem will be to

$$\text{Minimize } l = \int_{x_1}^{x_2} dl = \int_{x_1}^{x_2} \sqrt{1 + y'^2}\, dx,$$

such that

$$A = \int_{x_1}^{x_2} y\, dx + \frac{1}{2} {x_1}^2 \tan\left(\frac{\pi}{3}\right) = constant$$

other constraints or boundary condition for $y = y(x)$ are as follow

$$y(x_2) = 0$$

$$y'(x_2) = \tan\left(\pi - \frac{\theta}{2}\right)$$

$$y'(x_1) = \tan\left(\pi - \frac{\pi}{6}\right)$$

Using theorem A.3.1 to minimize the length of arc, we will have

$$f(x, y, y') = \sqrt{1 + y'^2}, \quad g(x, y, y') = y \quad \Rightarrow$$

$$h(x, y, y') = f(x, y, y') - \lambda g(x, y, y') = \sqrt{1 + y'^2} - \lambda y$$

$$\frac{\partial h}{\partial y} - \frac{d}{dx}\left(\frac{\partial h}{\partial y'}\right) = -\lambda - \frac{d}{dx}\left(\frac{y'y''}{\sqrt{1 + y'^2}}\right) = 0$$

So our problem will reduce to solve the following ODE with assigned boundary conditions

$$y''^2 + y'y'''\left(1+y'^2\right) - \lambda\left(1+y'^2\right)^{\frac{3}{2}} = 0 \quad x_1 \le x \le x_2$$

$$y'(x_1) = \tan\left(\pi - \frac{\pi}{6}\right)$$

$$y'(x_2) = \tan\left(\pi - \frac{\theta}{2}\right)$$

$$y(x_2) = 0$$

where the constant λ in above equation and function $y(x_1 \le x \le x_2)$ should satisfy the following constraint at same time

$$A = \int_{x_1}^{x_2} y\,dx + \frac{1}{2}{x_1}^2 \tan\left(\frac{\pi}{3}\right)$$

As we see here, x_1, x_2, λ and y are unknown and we have 3 boundary conditions to solve the ODE and one constraint to determine the value of λ. But the problem here is that we do not know the x_1 and x_2 to solve this problem numerically (or any other way). It seems that if we use the polar coordinates we can get rid of unknowns x_1 and x_2. The simplified problem in cylindrical coordinates can be written as

$$\text{Minimize } \; l = \int_{\theta_1}^{\theta_2} dl = \int_{\theta_1}^{\theta_2} \sqrt{r^2 + \left(\frac{dr}{d\theta}\right)^2}\,d\theta$$

such that

$$A = \int_{\theta_1}^{\theta_2} dA = \int_{\theta_1}^{\theta_2} \frac{1}{2}r^2\,d\theta$$

The slope in polar coordinates is expressed as

$$m(\theta) = \frac{\tan\theta\frac{dr}{d\theta} + 1}{\frac{dr}{d\theta} - r\tan\theta}$$

so the boundary conditions or the constraints for our problem will be

$$m(\theta_1 = 0) = \tan\left(\pi - \frac{\Theta}{2}\right) = \frac{\tan\theta_1 \left.\frac{dr}{d\theta}\right|_{\theta_1} + r(\theta_1)}{\left.\frac{dr}{d\theta}\right|_{\theta_1} - r(\theta_1)\tan\theta_1}$$

$$m(\theta_2 = \frac{\pi}{3}) = \tan\left(\frac{5\pi}{6}\right) = \frac{\tan\theta_2 \left.\frac{dr}{d\theta}\right|_{\theta_2} + r(\theta_2)}{\left.\frac{dr}{d\theta}\right|_{\theta_2} - r(\theta_2)\tan\theta_2}$$

where Θ is the dihedral angle and is used to prevent any misunderstanding by θ as the independent variable in polar coordinates.

$$F(\theta, r, r') = \sqrt{r^2 + \left(\frac{dr}{d\theta}\right)^2}, \quad G(\theta, r, r') = \frac{1}{2}r^2 \quad \Rightarrow$$

$$H(\theta, r, r') = F(\theta, r, r') - \lambda G(\theta, r, r') = \sqrt{r^2 + r'^2} - \frac{\lambda}{2}r^2$$

$$\frac{\partial}{\partial r}\Big(H(\theta, r, r')\Big) - \frac{d}{d\theta}\left(\frac{\partial}{\partial r'}\Big(H(\theta, r, r')\Big)\right) = 0$$

$$\frac{\partial H}{\partial r} = r\left(\frac{1}{\sqrt{r^2 + r'^2}} - \lambda\right)$$

$$\frac{\partial H}{\partial r'} = \frac{r'}{\sqrt{r^2 + r'^2}}$$

$$\frac{d}{d\theta}\left(\frac{\partial H}{\partial r'}\right) = \frac{d}{d\theta}\left(\frac{r'}{\sqrt{r^2 + r'^2}}\right) = \frac{r^2 r'' - r r'^2}{(r^2 + r'^2)^{\frac{3}{2}}} \Rightarrow$$

$$r\left(\frac{1}{\sqrt{r^2 + r'^2}} - \lambda\right) - \frac{r^2 r'' - r r'^2}{(r^2 + r'^2)^{\frac{3}{2}}} = 0 \Rightarrow$$

$$r\left(r^2+r'^2\right)^{\frac{3}{2}}\left(r^2+2r'^2-rr''-\lambda\left(r^2+r'^2\right)^{\frac{3}{2}}\right)=0 \tag{A.23}$$

Considering the above ODE (Eq. A.23), r cannot be zero, and $\left(r^2+r'^2\right)^{3/2}$ cannot be zero, too. The first one means that the solution is zero function and the second one means that the differential arc length $(r^2+r'^2)d\theta$ will be zero everywhere, which is not correct. So the problem in polar coordinates reduces to the following ordinary differential equation with the following boundary conditions and one constraint

$$\tan\left(\frac{\Theta}{2}\right)=\frac{r}{r'}\Big|_{\theta=0}$$

$$\tan\left(\frac{5\pi}{6}\right)=\frac{\sqrt{3}r'+r}{r'-\sqrt{3}r}\Big|_{\theta=\pi/3}$$

$$A=\int_0^{\pi/3}\frac{1}{2}r^2\,d\theta$$

To solve the above nonlinear system of differential equation, we need to discretize equations and boundary conditions to have a system of nonlinear equations. Then we can use the Newton-Raphson method to solve that nonlinear system.

$$r_i'\simeq\frac{r_{i+1}-r_{i-1}}{2\Delta\theta},\quad r_i''\simeq\frac{r_{i+1}-2r_i+r_{i-1}}{\Delta\theta^2}\quad\Rightarrow$$

$$r_i^2+2\left(\frac{r_{i+1}-r_{i-1}}{2\Delta\theta}\right)^2-r_i\left(\frac{r_{i+1}-2r_i+r_{i-1}}{\Delta\theta^2}\right)-\lambda\left(r_i^2+\left(\frac{r_{i+1}-r_{i-1}}{2\Delta\theta}\right)^2\right)^{\frac{3}{2}}=0\quad\Rightarrow$$

$$F:\ r_i^2+\frac{1}{2\Delta\theta^2}\left(r_{i+1}-r_{i-1}\right)^2-\frac{1}{\Delta\theta^2}\left(r_i r_{i+1}-2r_i^2+r_i r_{i-1}\right)-\frac{\lambda}{8\Delta\theta^3}\left(4r_i^2\Delta\theta^2+\left(r_{i+1}-r_{i-1}\right)^2\right)^{\frac{3}{2}}=0 \tag{A.24}$$

$$A=\int_{\theta_1}^{\theta_2}\frac{1}{2}r^2\,d\theta\simeq\sum_{i=1}^{N}\frac{1}{2}r_i^2\Delta\theta\quad\Rightarrow$$

$$G:\ r_1^2+r_2^2+\cdots+r_N^2-\frac{2A}{\Delta\theta}=\sum_{i=1}^{N}r_i^2-\frac{2A}{\Delta\theta}=0 \tag{A.25}$$

As we have the third kind of boundary condition (Robin) in both ends, we define 'ghost' nodes in boundaries. The ghost node before the main first node is called '0' and the otherone is called '$N+1$'. At $i=1$ we have

$$\tan\left(\pi-\frac{\Theta}{2}\right)=\frac{r}{r'}\Big|_{\theta=0}\simeq\frac{r_1}{\frac{r_2-r_0}{2\Delta\theta}}\quad\Rightarrow$$

$$H:\ r_2-r_1\left(\frac{2\Delta\theta}{\tan\left(\pi-\Theta/2\right)}\right)-r_0=0 \tag{A.26}$$

and at $i=N$ we have

$$\tan\left(\frac{5\pi}{6}\right)=\frac{\sqrt{3}r'+r}{r'-\sqrt{3}r}\Big|_{\theta=\pi/3}=\frac{\sqrt{3}\left(r_{N+1}-r_{N-1}\right)+2\sqrt{3}\,r_N\Delta\theta}{r_{N+1}-r_{N-1}-2\sqrt{3}\,r_N\Delta\theta}\quad\Rightarrow$$

$$K:\ \left(\tan\left(5\pi/6\right)-\sqrt{3}\right)r_{N+1}-(2\Delta\theta)\left(\sqrt{3}\tan\left(5\pi/6\right)+1\right)r_N+\left(\sqrt{3}-\tan\left(5\pi/6\right)\right)r_{N-1}=0 \tag{A.27}$$

Newton-Raphson method for solving nonlinear system of equations - A system of n equations in n unknowns $x_1, x_2, \ldots, x_n$ is called nonlinear if one or more of the equations is nonlinear. Any nonlinear $n \times n$ system can be put in the general form

$$\begin{cases} f_1(x_1, x_2, \ldots, x_n) = 0 \quad \Rightarrow \quad f_1(\underset{\sim}{x}) = 0 \\ f_2(x_1, x_2, \ldots, x_n) = 0 \quad \Rightarrow \quad f_2(\underset{\sim}{x}) = 0 \\ \vdots \\ f_n(x_1, x_2, \ldots, x_n) = 0 \quad \Rightarrow \quad f_n(\underset{\sim}{x}) = 0 \end{cases} \quad \Rightarrow \quad \underset{\sim}{F}(\underset{\sim}{x}) = 0 \qquad \text{(A.28)}$$

Suppose that

$$\underset{\sim}{\bar{x}} = [\bar{x}_1, \bar{x}_2, \ldots, \bar{x}_n]^T$$

is a solution of the system represented by Eq. A.28. This means that

$$f_1(\underset{\sim}{\bar{x}}) = f_2(\underset{\sim}{\bar{x}}) = \cdots = f_n(\underset{\sim}{\bar{x}}) = 0.$$

If x approximates $\bar{x}$, then the increment from x to $\bar{x}$ will be denoted by

$$\Delta \underset{\sim}{x} = \underset{\sim}{\bar{x}} - \underset{\sim}{x} = \begin{bmatrix} \bar{x}_1 - x_1 \\ \bar{x}_2 - x_2 \\ \vdots \\ \bar{x}_n - x_n \end{bmatrix} = \begin{bmatrix} \Delta x_1 \\ \Delta x_2 \\ \vdots \\ \Delta x_n \end{bmatrix}$$

where Δx_j is the increment from x_j to $\bar{x}_j$ for $j = 1, 2, \ldots, n$. We need to find the direction and distance to move from x (in n-space) to get to the desired point

$$\bar{x} = x + \Delta x$$

which is mapped into zero by each $f_1, f_2, \ldots, f_n$. Rather than seeking the exact increment Δx that satisfies

$$f_i(x + \Delta x) = 0, \qquad i = 1, 2, \ldots, n$$

we should try to find an approximate increment

$$dx = [dx_1, dx_2, \ldots, dx_n]^T$$

that satisfies the more easily solved system

$$\text{Linear approximation of } f_i(x + dx) = 0, \qquad i = 1, 2, \ldots, n$$

in other words, we can say, dx satisfies the following approximating system

$$\begin{cases} f_1(x) + \dfrac{\partial f_1}{\partial x_1}dx_1 + \dfrac{\partial f_1}{\partial x_2}dx_2 + \cdots + \dfrac{\partial f_1}{\partial x_n}dx_n = 0 \\ f_1(x) + \dfrac{\partial f_2}{\partial x_1}dx_1 + \dfrac{\partial f_2}{\partial x_2}dx_2 + \cdots + \dfrac{\partial f_2}{\partial x_n}dx_n = 0 \\ \vdots \\ f_n(x) + \dfrac{\partial f_n}{\partial x_1}dx_1 + \dfrac{\partial f_n}{\partial x_2}dx_2 + \cdots + \dfrac{\partial f_n}{\partial x_n}dx_n = 0 \end{cases} \tag{A.29}$$

So for having an approximation of exact increment (Δx), dx, we should solve the system of equations presented in A.29, when partial derivatives ($\partial f_i / \partial x_j$) are evaluated at the currently known x. This linear system can be expressed in matrix from as

$$\underbrace{\begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \frac{\partial f_1}{\partial x_2} & \cdots & \frac{\partial f_1}{\partial x_n} \\ \frac{\partial f_2}{\partial x_1} & \frac{\partial f_2}{\partial x_2} & \cdots & \frac{\partial f_2}{\partial x_n} \\ \vdots & & & \\ \frac{\partial f_n}{\partial x_1} & \frac{\partial f_n}{\partial x_2} & \cdots & \frac{\partial f_n}{\partial x_n} \end{bmatrix}}_{\underset{\sim}{F}'(\underset{\sim}{x})=\frac{\partial f_i}{\partial x_j}} \underbrace{\begin{bmatrix} dx_1 \\ dx_2 \\ \vdots \\ dx_n \end{bmatrix}}_{d\underset{\sim}{x}} = - \underbrace{\begin{bmatrix} f_1(x) \\ f_2(x) \\ \vdots \\ f_n(x) \end{bmatrix}}_{\underset{\sim}{F}(\underset{\sim}{x})}$$

The solution of above system, where $\underset{\sim}{F}'(\underset{\sim}{x})$ is nonsingular, is

$$d\underset{\sim}{x} = \underset{\sim}{F}'^{-1}(\underset{\sim}{x}) \times \underset{\sim}{F}(\underset{\sim}{x}).$$

The square matirx $\underset{\sim}{F}'(\underset{\sim}{x})$ is called Jacobian matrix of F at x. We should note that row$_i$ at $\underset{\sim}{F}'$ contains all partial derivatives of f_i whereas col$_j$ of $\underset{\sim}{F}'$ contains all partials with respect to x_j thus

$$\underset{\sim}{F}(\underset{\sim}{x}) = \left(\frac{\partial f_i(x)}{\partial x_j} \right)_{n \times n}$$

If x is close enough to $\bar{x}$ so that the linear approximation

$$f_i(x + \delta x) \approx f_i(x) + \frac{\partial f_i(x)}{\partial x_1} dx_1 + \cdots + \frac{\partial f_i(x)}{\partial x_n} dx_n$$

is accurate for $i = 1, 2, \ldots, n$, then Δx (which satisfies $f_i(x + \Delta x) =$) should be accurately approximated by $d\underset{\sim}{x}$ (which satisfies the system $\underset{\sim}{F}'(\underset{\sim}{x}) d\underset{\sim}{x} = -\underset{\sim}{F}(\underset{\sim}{x})$). Hence

$$\bar{\underset{\sim}{x}} = \underset{\sim}{x} + \Delta\underset{\sim}{x} \approx \underset{\sim}{x} + d\underset{\sim}{x} = x - \underset{\sim}{F}'^{-1}(\underset{\sim}{x}) \underset{\sim}{F}(\underset{\sim}{x})$$

and the iteration can take place as follows

$$\underset{\sim}{x}^{k+1} = \underset{\sim}{x}^{k} + (d\underset{\sim}{x})^{k}$$

$$(d\underset{\sim}{x})^{k} = -\underset{\sim}{F}^{\prime -1}(\underset{\sim}{x}^{k})\underset{\sim}{F}(\underset{\sim}{x}^{k})$$

Now we apply this method to our problem. We should solve the following nonlinear system of equations

$$\begin{cases} F:\ r_i^2 + \dfrac{1}{2\Delta\theta^2}\left(r_{i+1} - r_{i-1}\right)^2 - \dfrac{1}{\Delta\theta^2}\left(r_i r_{i+1} - 2r_i^2 + r_i r_{i-1}\right) \\ \qquad\qquad -\dfrac{\lambda}{8\Delta\theta^3}\left(4r_i^2\Delta\theta^2 + \left(r_{i+1} - r_{i-1}\right)^2\right)^{\frac{3}{2}} = 0 & i = 2, 3, \ldots, N-1 \\ G:\ r_1^2 + r_2^2 + \cdots + r_N^2 - \dfrac{2A}{\Delta\theta} = 0 & i = 1, 2, \ldots, N \\ H:\ r_2 - r_1\left(\dfrac{2\Delta\theta}{\tan\left(\pi - \Theta/2\right)}\right) - r_0 = 0 & i = 1 \\ K:\ \left(\tan\left(5\pi/6\right) - \sqrt{3}\right) r_{N+1} - \left(2\Delta\theta\right)\left(\sqrt{3}\tan\left(5\pi/6\right) + 1\right) r_N \\ \qquad\qquad + \left(\sqrt{3} - \tan\left(5\pi/6\right)\right) r_{N-1} = 0 & i = N \end{cases} \tag{A.30}$$

We should notice that our solution vector has the following form

$$\underset{\sim}{r} = [r_0, r_1, r_2, \ldots, r_N, r_{N+1}, \lambda]^T$$

and the correction vector for the initial guess is represented as below

$$d\underset{\sim}{r}^{k} = \left[dr_0^k, dr_1^k, dr_2^k, \ldots, dr_N^k, dr_{N+1}^k, d\lambda^k\right]^T$$

We are trying to solve $(d\underset{\sim}{r})^k = -\underset{\sim}{J}^{-1}\underset{\sim}{B}$, in which, B, is the right hand side of the linear system of equation and can be shown by

$$\text{RHS: } B = \begin{bmatrix} -H(r_0^k, r_1^k, r_2^k) \\ -F(r_0^k, r_1^k, r_2^k) \\ -F(r_1^k, r_2^k, r_3^k) \\ \vdots \\ -F(r_{N-1}^k, r_N^k, r_{N+1}^k) \\ -K(r_{N-1}^k, r_N^k, r_{N+1}^k) \\ -G(r_1^k, r_2^k, \ldots, r_N^k) \end{bmatrix}$$

Jacobian matrix can be constructed as follow

$$\frac{\partial F}{\partial r_{i-1}} = F_1(i) = \frac{1}{\Delta\theta^2}\left(r_{i-1} - r_i - r_{i+1}\right) + \frac{3\lambda}{8\Delta\theta^3}\left(r_{i+1} - r_{i-1}\right)\left(4r_i^2\Delta\theta^2 + \left(r_{i+1} - r_{i-1}\right)^2\right)^{\frac{1}{2}}$$

$$\frac{\partial F}{\partial r_i} = F_2(i) = 2r_i - \frac{1}{\Delta\theta^2}\left(r_{i+1} - 4r_i + r_{i-1}\right) - \frac{3\lambda r_i}{2\Delta\theta}\left(4r_i^2\Delta\theta^2 + \left(r_{i+1} - r_{i-1}\right)^2\right)^{\frac{1}{2}}$$

$$\frac{\partial F}{\partial r_{i+1}} = F_3(i) = \frac{1}{\Delta\theta^2}\left(r_{i+1} - r_i - r_{i-1}\right) - \frac{3\lambda}{8\Delta\theta^3}\left(r_{i+1} - r_{i-1}\right)\left(4r_i^2\Delta\theta^2 + \left(r_{i+1} - r_{i-1}\right)^2\right)^{\frac{1}{2}}$$

$$\frac{\partial F}{\partial \lambda} = F_\lambda(i) = -\frac{1}{8\Delta\theta^3}\left(4r_i^2\Delta\theta^2 + \left(r_{i+1} - r_{i-1}\right)^2\right)^{\frac{3}{2}}$$

$$\frac{\partial G}{\partial r_i} = G(i) = 2r_i$$

$$\frac{\partial H}{\partial r_0} = H_0 = -1$$

$$\frac{\partial H}{\partial r_1} = H_1 = -\frac{2\Delta\theta}{\tan\left(\pi - \Theta/2\right)}$$

$$\frac{\partial H}{\partial r_2} = H_2 = 1$$

$$\frac{\partial K}{\partial r_{N-1}} = K_{N-1} = \sqrt{3} - \tan\left(5\pi/6\right)$$

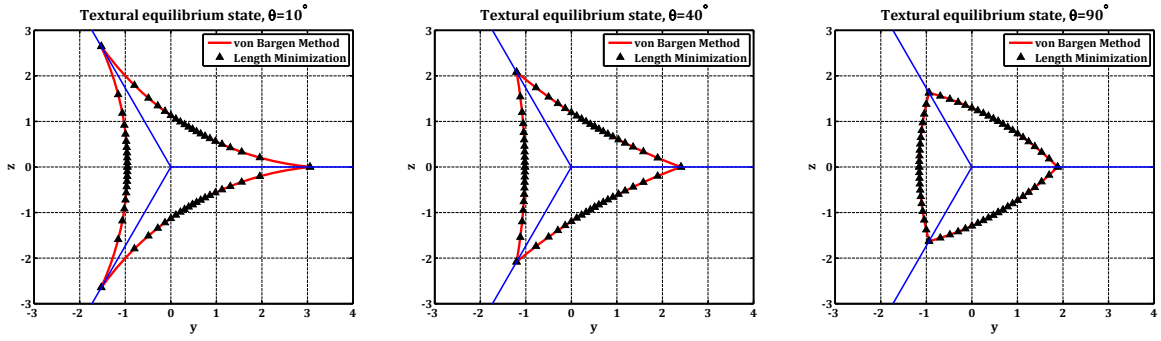
$$\frac{\partial K}{\partial r_N} = K_N = -2\left(\sqrt{3}\tan\left(5\pi/6\right) + 1\right)\Delta\theta$$

$$\frac{\partial K}{\partial r_{N+1}} = K_{N+1} = \tan\left(5\pi/6\right) - \sqrt{3}$$

$$J = \begin{bmatrix}
H_0 & H_1 & H_2 & 0 & 0 & 0 & 0 & 0 & 0 \\
F_1(1) & F_2(1) & F_3(1) & 0 & 0 & 0 & 0 & 0 & F_\lambda(1) \\
0 & F_1(2) & F_2(2) & F_3(2) & 0 & 0 & 0 & 0 & F_\lambda(2) \\
0 & 0 & F_1(3) & F_2(3) & F_3(3) & 0 & 0 & 0 & F_\lambda(3) \\
\vdots & \dots & & \ddots & \ddots & \ddots & & & \vdots \\
\vdots & \dots & & & \ddots & \ddots & \ddots & & \vdots \\
0 & 0 & 0 & 0 & F_1(N-1) & F_2(N-1) & F_3(N-1) & 0 & F_\lambda(N-1) \\
0 & 0 & 0 & 0 & 0 & F_1(N) & F_2(N) & F_3(N) & F_\lambda(N) \\
0 & 0 & 0 & 0 & 0 & K_{N-1} & K_N & K_{N+1} & 0 \\
0 & G(1) & G(2) & \dots & \dots & \dots & G(N) & 0 & 0
\end{bmatrix}$$

so we have Jacobian matrix and RHS vector and we can find $d\underset{\sim}{r}^k$ simply by inversing J and multiplying by B. The iteration process will be continued until $d\underset{\sim}{r}^k$ is so small and negligible or B is close enough to zero. The result of above method is presented below.

The equilibrium topology of a single 2D pore at a triple-junction is calculated using Eq. A.23 subject to constant pore volume (in 2D area) and the dihedral angle condition. Additionally, the von-Bargen's method von Bargen and Waff (1986) is applied to the same problem for comparison and validation. The dihedral angle condition is applied where fluid meets the grain-grain contact. Fig. A.14 shows the results of simulations for both methods and different θ. As shown in Fig. A.14, both methods eventually return identical results. Keeping the fluid volume constant and increasing θ, pore becomes fatter and converges to a circle in an upper limit of θ ($\theta = 180°$, completely non-wetting fluid).



(a) $\theta = 10°$ (b) $\theta = 40°$ (c) $\theta = 90°$

Figure A.14: Effect of θ on the equilibrium shape of a single pore from interfacial area minimization and comparison with von-Bargen method

Fig. A.15 shows the distribution of fluid in a two-dimensional isotropic crystal network at different θ and ϕ. By keeping ϕ constant and increasing θ, the fluid pockets shrink along grain boundaries and percolation probability decreases tremendously. Keeping θ constant and increasing ϕ extends the contact between the fluid pockets and the grain boundary and increases the percolation probability. As can be seen, cases with smaller θ are more likely to make a percolating pore network even at much smaller ϕ.

Fig. A.16 shows the equilibrium fluid-solid structure for an elongated crystal net-

work. We have assumed that solid-liquid and solid-solid surface tensions are constant. We can assume that this anisotropy comes from an anisotropic stress state and fluid is introduced to the system after formation of crystals. As shown in Fig. A.16, keeping ϕ constant and increasing θ results in the shrinkage of fluid pockets and vice versa. This special case shows how a small change in the shape of grains causes considerable anisotropy in fluid topology. The fluid is likely to connect in the x-direction (the shortest path) as ϕ increases. At the moment when these pockets meet in the shorter direction, they shrink and make it less probable to connect in the other direction. There is still an anisotropy in topology when the fluid becomes connected in all directions. The fluid channel throats are thicker in the direction of the shortest percolating path, making it easier for fluid to move in this direction, and this introduces anisotropy in permeability.

Fig. A.17 shows the textural equilibrium in a bowed crystal network. In this case, fluid pockets connect along different grain boundaries in comparison with previous cases. The 2-D results confirm that the topology of fluid in contact with a crystal network is affected by θ and ϕ. Also, the initial structure has profound effects on fluid pocket distribution even for simplified lattices. A slight change in crystal shape can give rise to the anisotropy of permeability and the formation of percolating paths in salt. The method used here needs tracking of fluid phase connectivity and topology, so extending it to random crystal network and/or 3-D crystal lattice is not trivial. Therefore, we need to develop new method which can handle these problems in a smarter way. Although these results are valid in 2-D world, the 3-D results can be surprisingly different. In 3-D, we expect the pore space be connected in very small ϕ ($<1\%$) when $\theta < 60^\circ$. We also expect the anisotropy of pore geometry be different and affected more strongly by θ.

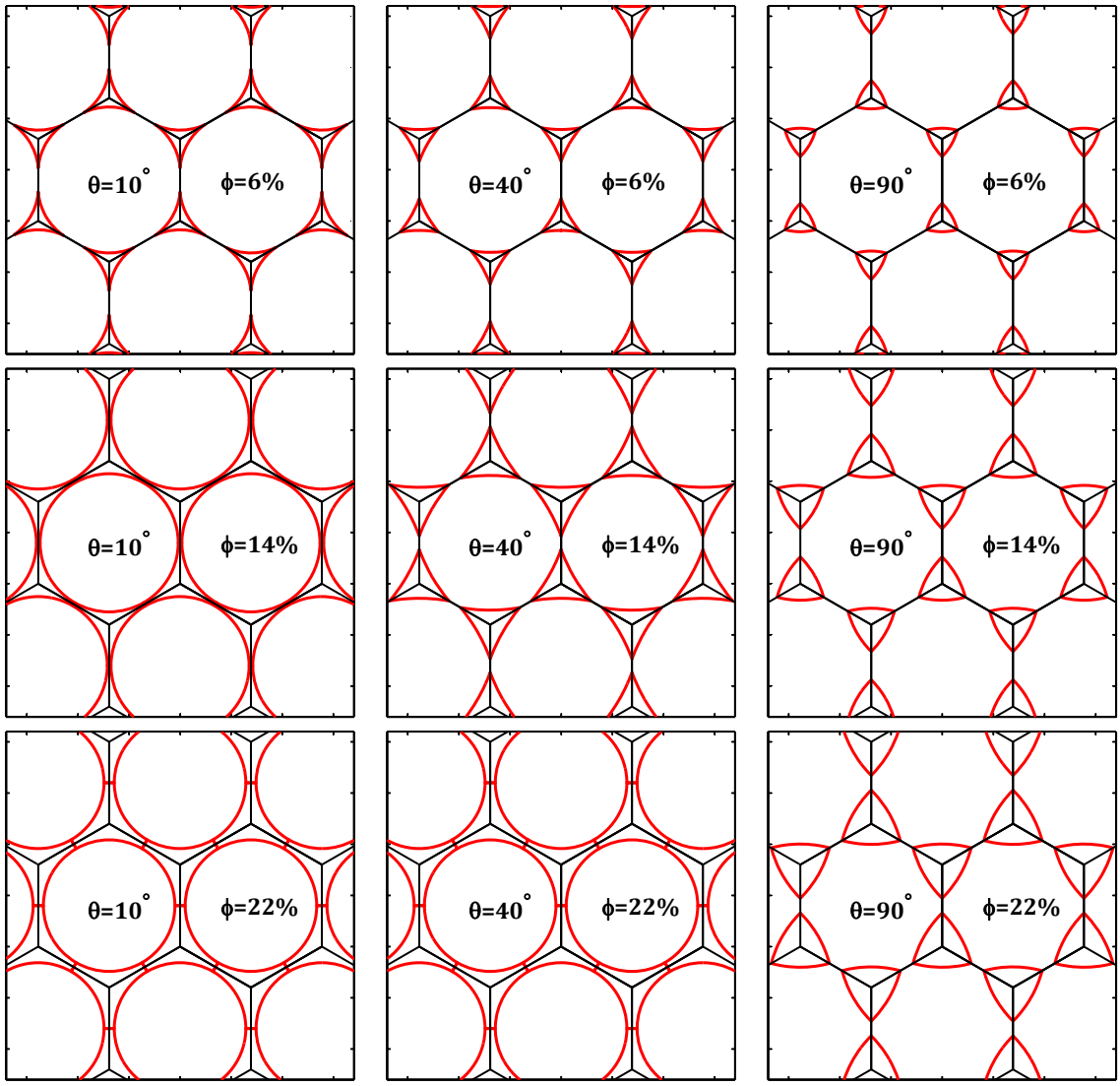


Figure A.15: Effect of ϕ and θ on the topology of fluid in a symmetric crystal lattice

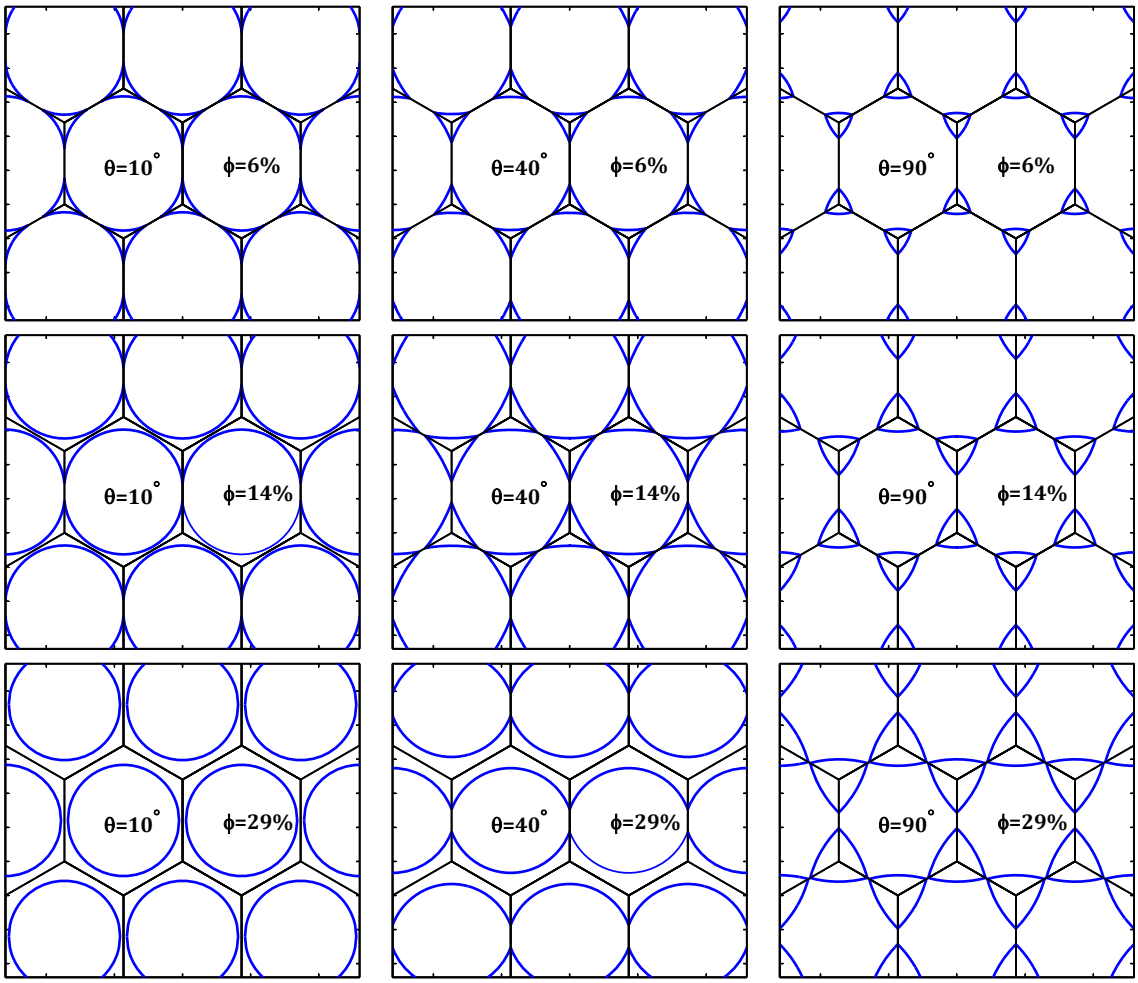


Figure A.16: Effect of ϕ and θ on the topology of fluid in an elongated crystal lattice.

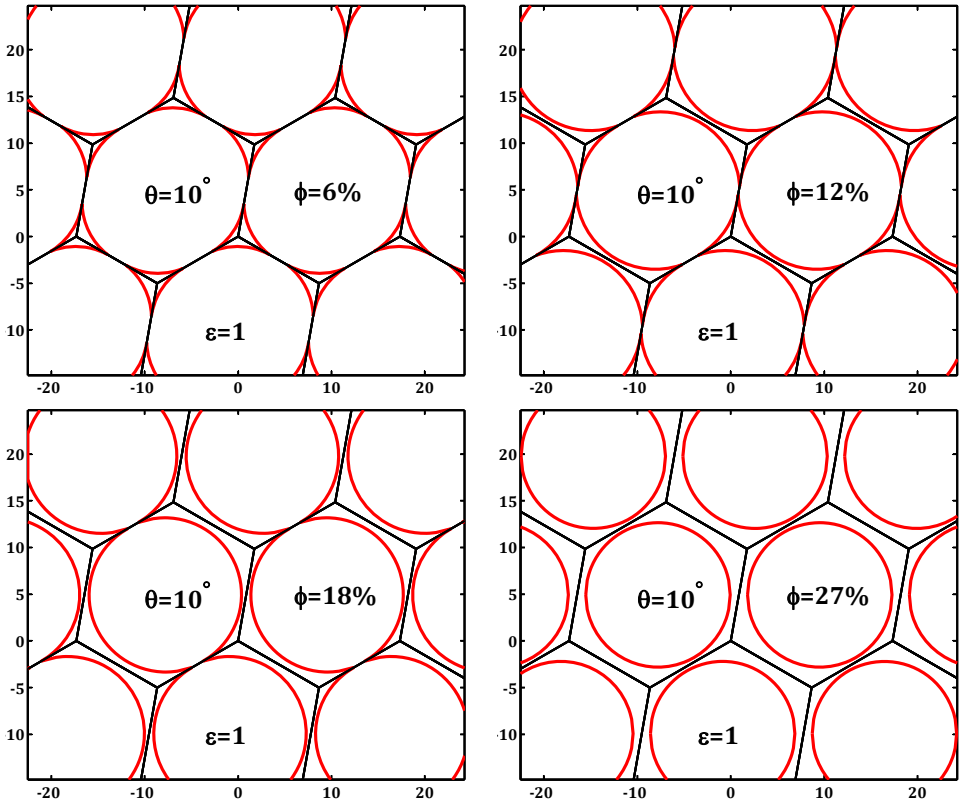


Figure A.17: Effect of ϕ and θ on the topology of fluid in a 2D bowed crystal lattice

Appendix B

Permeability Computation and Results

The permeability of computed pore networks is calculated using Palabos, which is in an open source CFD package based on the Lattice Boltzmann method and implemented in parallel. Palabos is developed based on Lattice Bhatnagar-Gross-Krook (LBGK) model where momenta of colliding particles will redistribute at some constant rate toward an equilibrium distribution. Permeability is calculated by imposing a constant pressure at the inlet, and a lower pressure at the outlet. The flux is compared with Darcy's law and the resulting constant is permeability.

Here we use the pore networks computed using the level-set methodGhanbarzadeh et al. (2015b) to calculate the permeability of the pore space. As expected all pore networks with $\theta \leq 60^{\circ}$ are interconnected and the permeability has a finite non-zero value for all non-zoro melt fractions (Fig. B.1 and B.2). We also consider the hysteresis in pore network topology and computed permeability for cases between the trapping and percolation thresholds when $\theta > 60^{\circ}$. Permeability of the pore network in these cases is initially zero until the melt fraction exceeds the percolation threshold. After this, the permeability follows a smooth path. Decreasing the melt fraction to values below percolation threshold does not disconnect the pore network and permeability does not vanish until trapping threshold is reached (Fig. B.3 and B.4).

The velocity field (magnitude of velocity) in texturally equilibrated pore network in the regular solid shows the melt channels that are aligned to xy-plain do not contribute to flow in z-direction (insert in Fig. B.3*b*). However, in the realistic grains the near horizontal

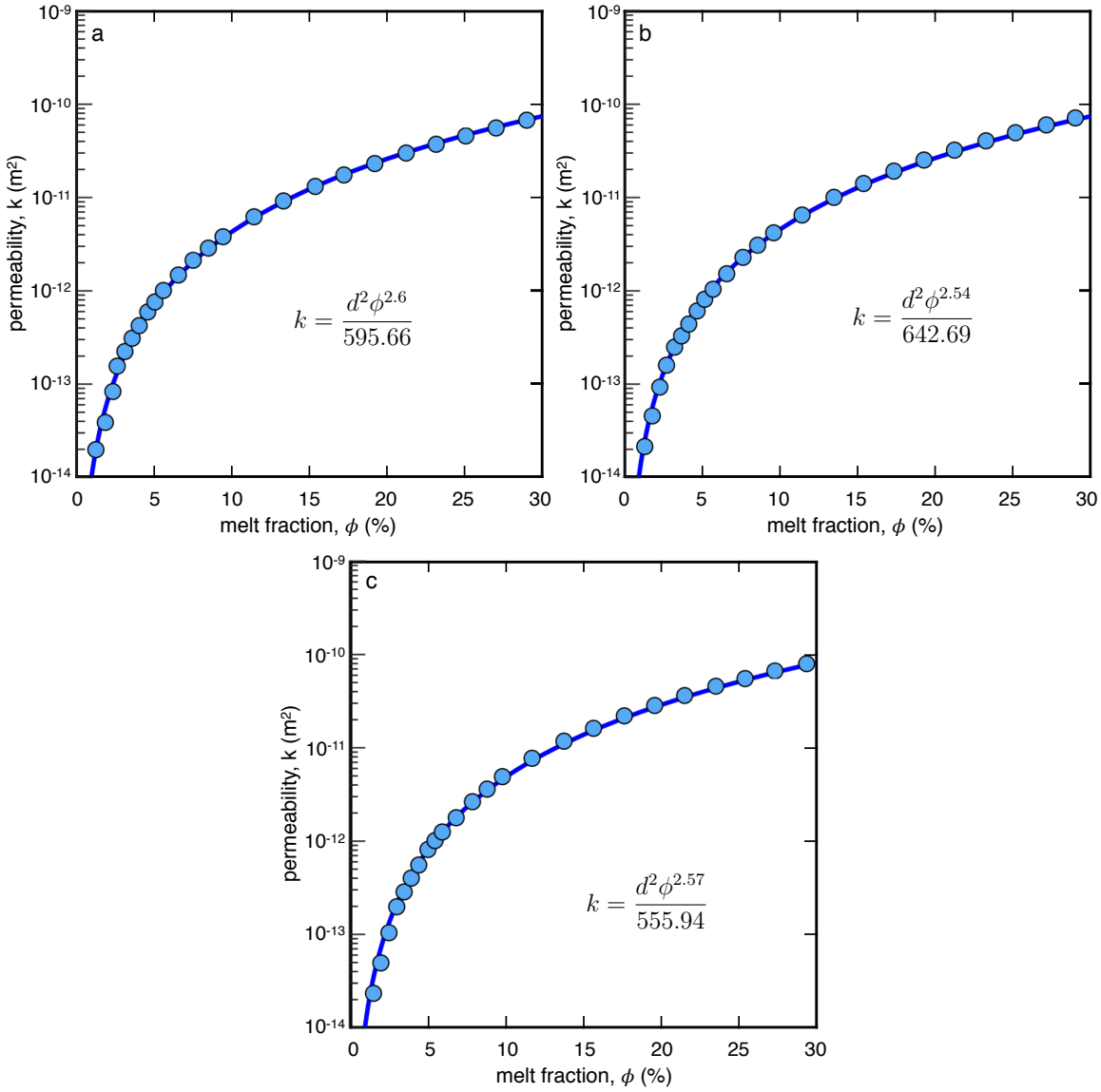


Figure B.1: Permeability for solid comprised of truncated octahedron grains $\boldsymbol{\theta \leq 60^\circ}$. Permeability is shown in SI units in semi-log plots, assuming average grain size of 1mm. Power law fit functions are inserted in figures and are plotted with solid line. Melt network is interconnected for all examined melt fractions. (a) $\theta = 10^\circ$, (b) $\theta = 30^\circ$ and (c) $\theta = 60^\circ$.

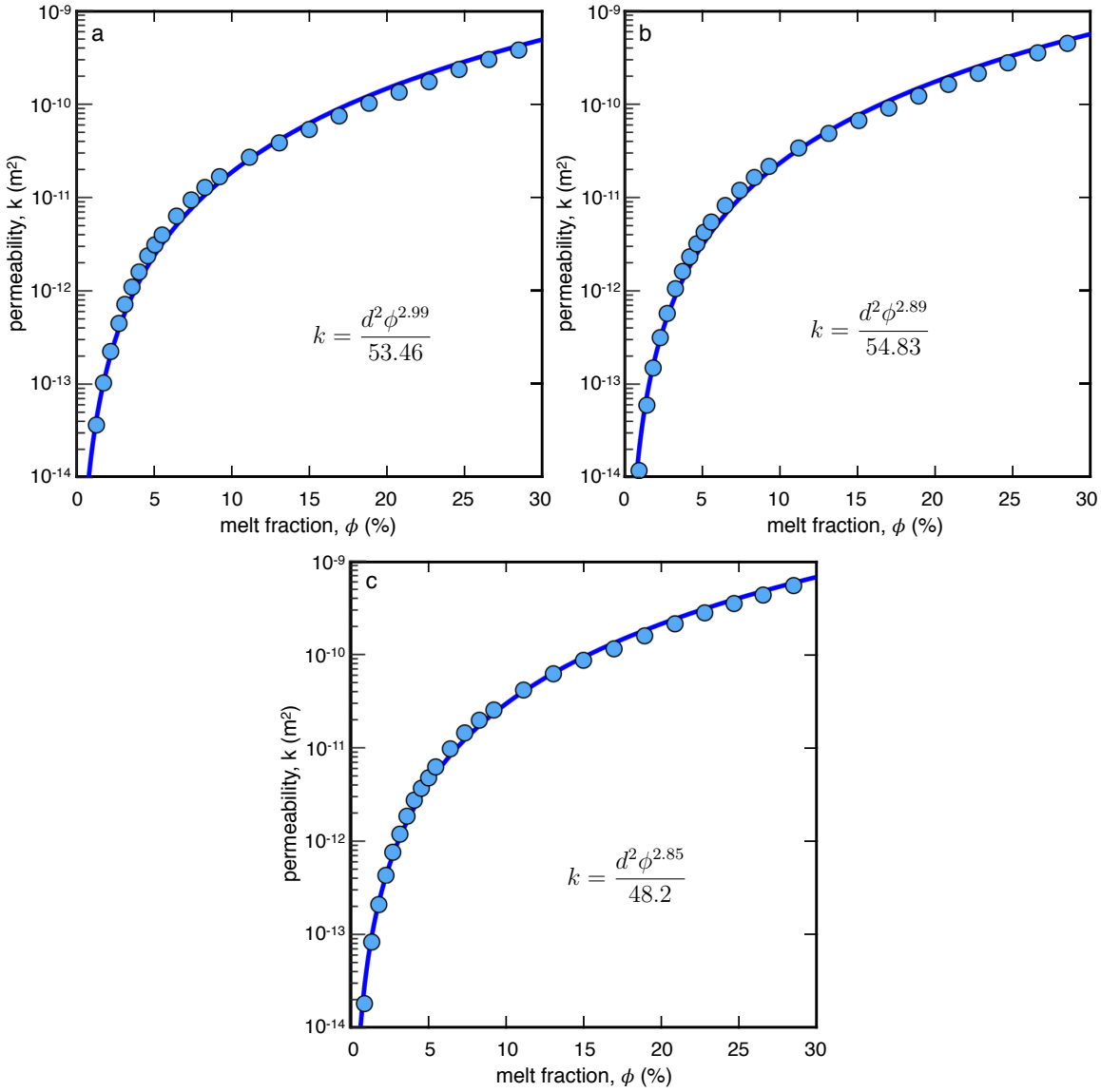


Figure B.2: Permeability for solid comprised of irregular and realistic grains with $\boldsymbol{\theta \leq 60^\circ}$. Permeability is shown in SI units in semi-log plots, assuming average grain size of 1mm. Power law fit functions are inserted in figures and are plotted with solid line. Melt network is interconnected for all examined melt fractions. (a) $\theta = 10^\circ$, (b) $\theta = 30^\circ$ and (c) $\theta = 60^\circ$.

Table B.1: Permeability-melt fraction relationships for different dihedral angles and grain textures.

θ	Truncated octahedron grains	irregular grains
10°	$k = \frac{d^2\phi^{2.6}}{595.66}$	$k = \frac{d^2\phi^{2.99}}{53.46}$
30°	$k = \frac{d^2\phi^{2.54}}{642.69}$	$k = \frac{d^2\phi^{2.89}}{54.83}$
60°	$k = \frac{d^2\phi^{2.57}}{555.94}$	$k = \frac{d^2\phi^{2.85}}{48.2}$
70°	$k = \frac{d^2\phi^{2.56}}{474.24}$	$k = \frac{d^2\phi^{2.79}}{45.87}$
90°	$k = \frac{d^2\phi^{2.65}}{407.38}$	$k = \frac{d^2\phi^{2.53}}{168.35}$
105°	$k = \frac{d^2\phi^{2.57}}{448.74}$	$k = \frac{d^2\phi^{2.36}}{694.44}$
120°	$k = \frac{d^2\phi^{2.87}}{274.79}$	$k = \frac{d^2\phi^{2.04}}{3176.85}$

melt channels also contribute to porous flow due to the irregularity of the pore-network (insert in Fig. B.3*b*). Here we summarize the computed values of permeability of the medium versus melt fraction in both increasing and decreasing porosity paths for regular and irregular grains (Table **??**, Figs. B.1-B.4). The permeability values are converted from lattice units to SI [m^2] units by scaling the average grain size to 1 mm. All the computed data are also available in a spreadsheet as supplementary information in the online version.

In Fig. B.5, all the computed permeability data are plotted in $\phi\theta$ space for both regular and irregular media. For better visualization, the logarithm of the permeability is plotted. Dots show the (ϕ, θ) pairs where the pore networks and permeability have been computed. The white areas show the 'no percolation' zone where melt is trapped and the hatched areas represent the porosity and dihedral angle range at which we expect the connectivity to be a function of the history of the system. Permeability is mainly a function

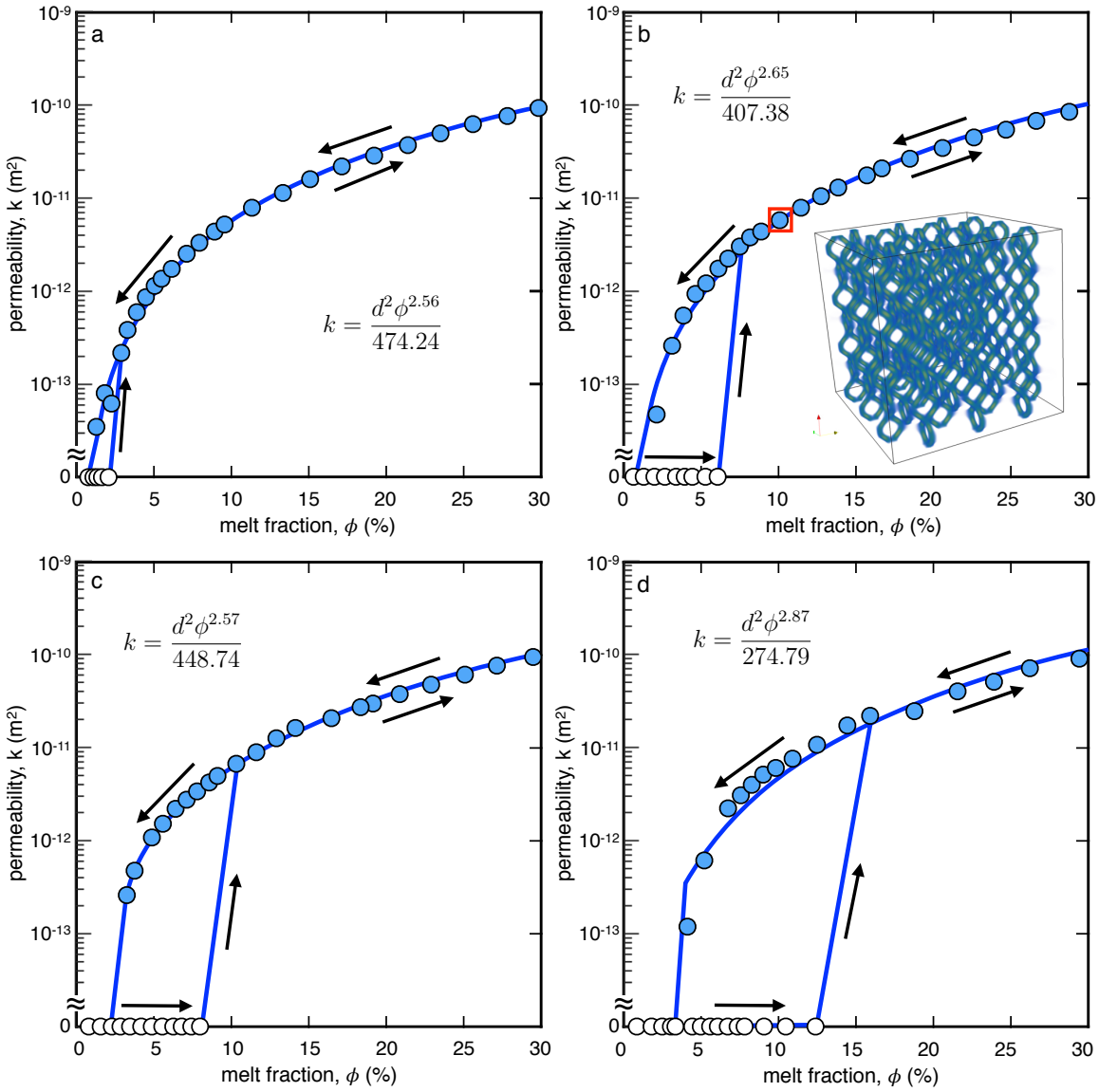


Figure B.3: Permeability for solid comprised of truncated octahedron grains $\boldsymbol{\theta > 60°}$. Permeability is shown in SI units in semi-log plots, assuming average grain size of 1mm. Hysteresis in melt network connectivity introduces a loop in permeability values. Y-axis is cut to account for zero permeability values in disconnected networks. Power law fit functions are inserted in figures and are plotted with solid line. Empty dots denote the disconnected pore space and filled dots indicate a percolating melt network. (a) $\theta = 70°$, (b) $\theta = 90°$, (c) $\theta = 105°$ and (d) $\theta = 120°$.

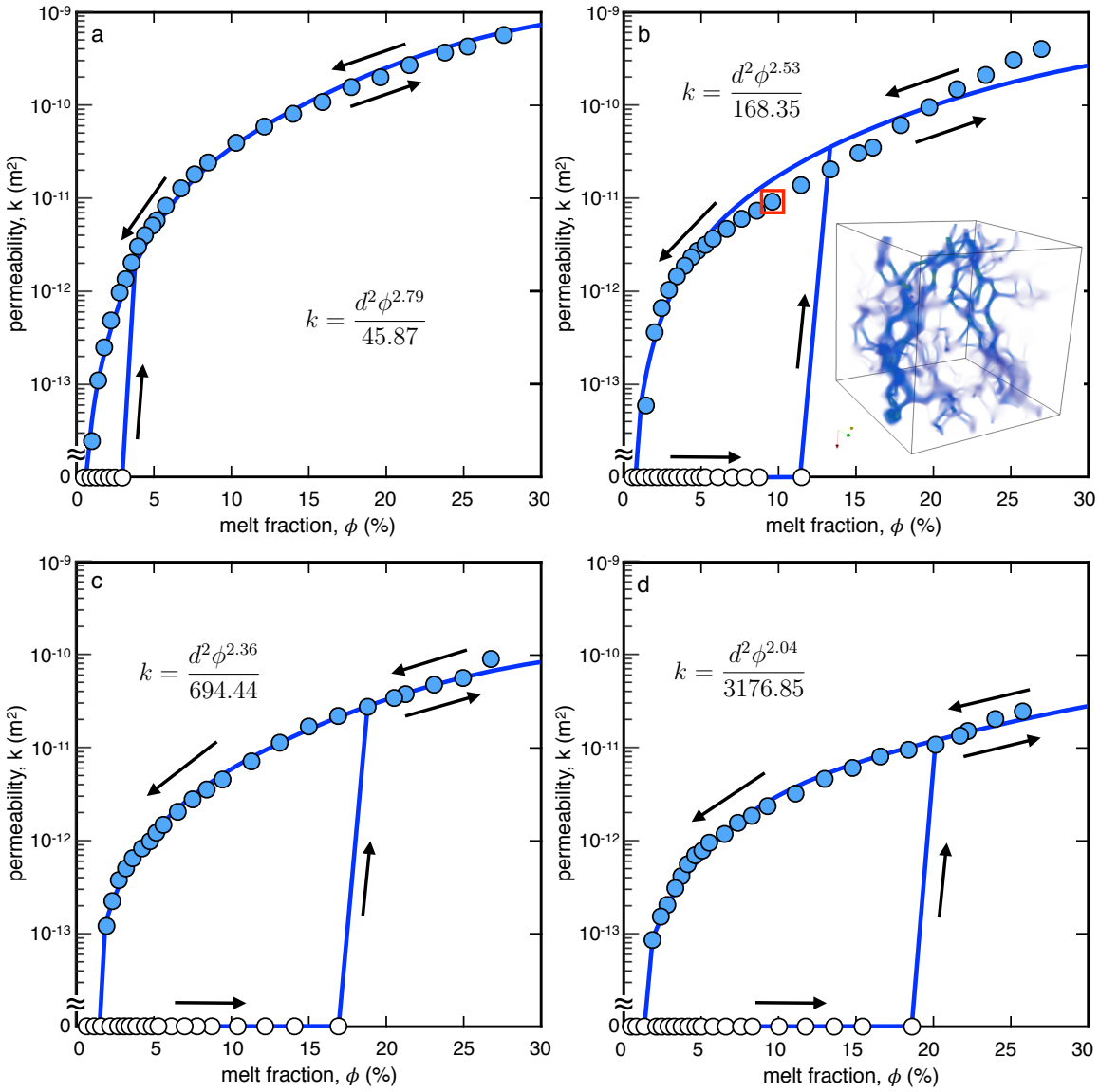


Figure B.4: Permeability for solid comprised of irregular and realistic grains with $\boldsymbol{\theta > 60^\circ}$. Permeability is shown in SI units in semi-log plots, assuming average grain size of 1mm. Hysteresis in melt network connectivity introduces a loop in permeability values. Y-axis is cut to account for zero permeability values in disconnected networks. Power law fit functions are inserted in figures and are plotted with solid line. Empty dots denote the disconnected pore space and filled dots indicate a percolating melt network. (a) $\theta = 70^\circ$, (b) $\theta = 90^\circ$, (c) $\theta = 105^\circ$ and (d) $\theta = 120^\circ$.

of porosity and the dependence of permeability on the dihedral angle is very weak. This implies that although dihedral angle has a first-order control on connectivity of the pore network, once the pore space is connected, the important properties affecting the permeability, i.e. tortuosity of the pore network, are not a strong function of dihedral angle. The permeability attains a maximum at $\theta \approx 70^{\circ}$. For smaller dihedral angles the permeability decreases because the melt spreads further onto the grain boundaries, reducing the effective hydraulic radius of the grain edge channels. For larger dihedral angles the permeability decreases, because the melt accumulates in the grain corners, reducing the size of the grain edge channels.

The computed permeability values are also compared with available data in literature, whether direct experimental measurements or values computed using LBM on synthetic texturally equilibrated rocks (Fig. B.6). The permeability values versus melt fraction for different dihedral angles compare very well for both regular (Fig. B.6*a*) and irregular media (Fig. B.6*b*). All the permeability data, independent of dihedral angle, are collected and a single power law curve is passed through the data and visualized on a semi-log plot (Fig. B.5*c*). Computed permeability values, which are obtained from computed pore networks, fall between data presented in literature. This provides a basic observation for the validity of the computed pore networks and corresponding permeability values.

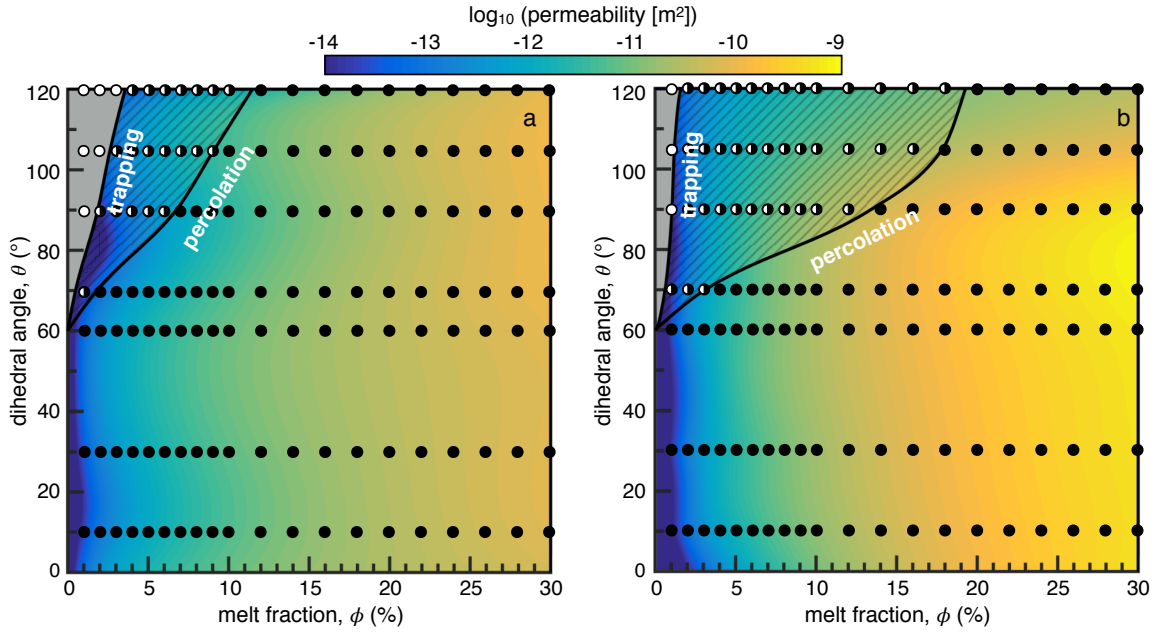


Figure B.5: Percolation-trapping thresholds with permeability. The calculated permeability of the texturally equilibrated melt network, in SI units, is superimposed. All data is scaled to correspond the average grain size of 1 mm. Hatched area between trapping and percolation thresholds indicates the region where melt drainage via porous flow is possible due to hysteresis in melt connectivity once percolation threshold is reached. (a) polycrystalline material comprised of truncated octahedron grains. (b) Polycrystalline solid comprised of realistic irregular grains.

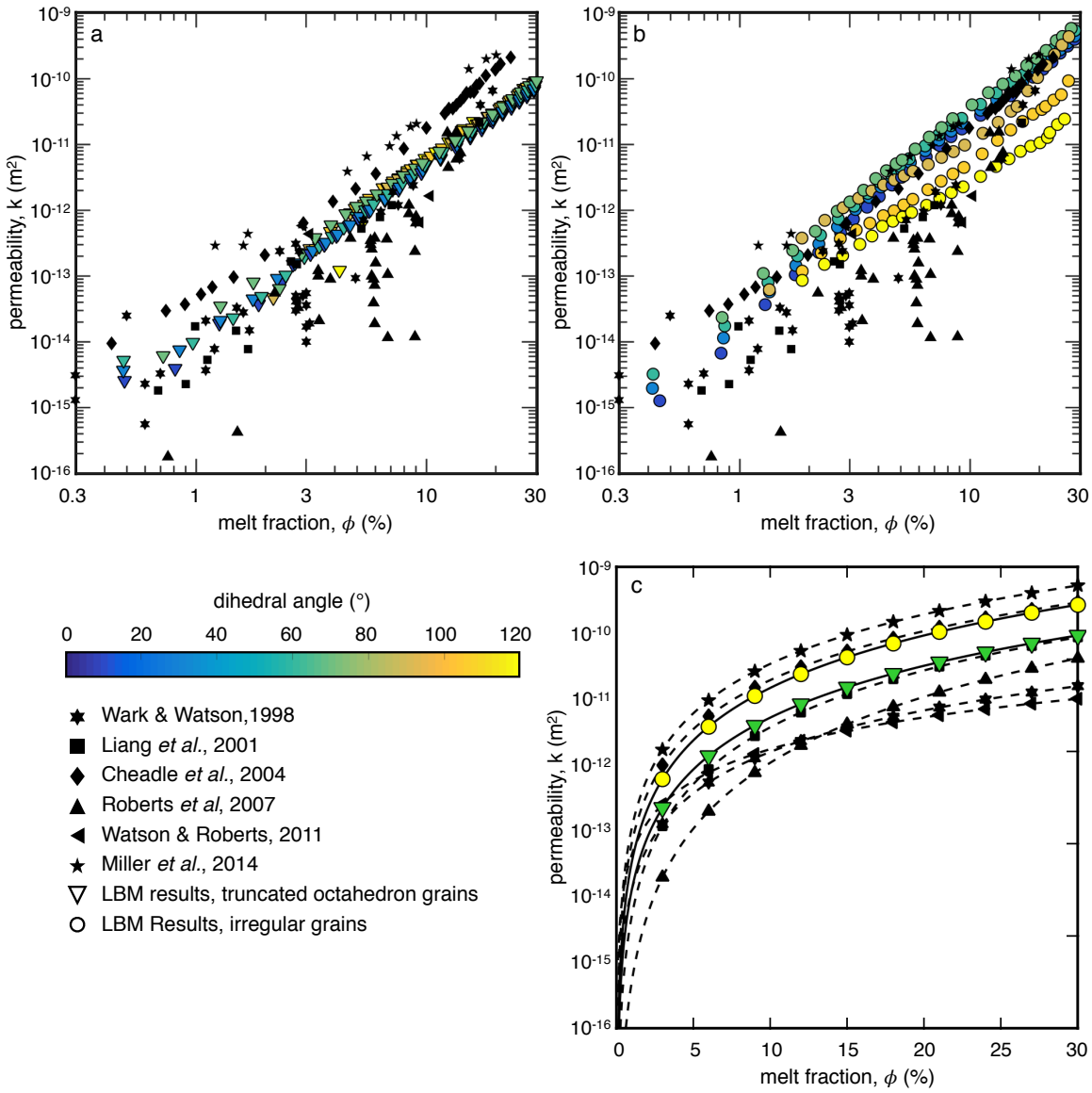


Figure B.6: Comparison of computed permeability with available data. Color map of dihedral angle and sources of data found in literature (black markers) are presented in bottom left. (a) permeability-porosity data on a log-log plot for solid comprised of truncated octahedron grains. The colored markers correspond to LBM results of computed pore networks using level set method. (b) permeability-porosity data on a log-log plot for solid comprised of irregular and realistic grains. The colored markers correspond to LBM results of computed pore networks using level set method. (c) Best power law fit of permeability shown in a and b on a semi-log plot. The LBM results are combined for each solid without considering the relationship to dihedral angle. All data is scaled to correspond the average grain size of 1 mmWark and Watson (1998); Liang et al. (2001); Cheadle et al. (2004); Roberts et al. (2007); Watson and Roberts (2011); Miller et al. (2014).

Appendix C

A Level Set Method for Materials with Texturally Equilibrated Pores

```
%% author: Soheil Ghanbarzadeh
%% Last version date: 06/28/2016
% Description:
% reads data from raw file, convert each to level set. It adjustes the grid
% size by interpolation of the original data set. Sets up the grid for each
% grain and initializes the liquid level set with solid grains. How? with
% curvature and normal motions. See the JCP for more info.

% Input:
% reads the original data set received form Ludwig. We need some info like
% size of scan and so on which are set below.


% Output:
% saves a file with name RGphi0.5The10.mat, which actually is the initial
% data structure that we work with in main.m.
% it has the required fields, including the level set function for solid
% grains and liquid as well as decomposed grid.
% ------------------------------------------------------------------------


% clean up
clear; close; clc;


% Read the data received from Ludwig. See their 2008 paper.
fid = fopen('data_unit16_470x470x201.raw','r');
% file size, obvous from name
nx = 470;
ny = 470;
nz = 201;
% read file
data = fread(fid,nx*ny*nz,'uint16');
% reshape the data to a 3d array
data = reshape(data,[nx ny nz]);
```

```
% close file
fclose(fid);

% ------------------------------------------------------------------------
% create two sets of grid, one corresponds to the original dataset
% received. But we have many grid points in each grain, this makes
% computations very slow. Based on my estimates, we only need 20 grid
% points in each direction in each grain to be able to solve this problem.
% That's why I am resampling the dataset down to half size with g2.
g.extrapolate = 1;
g.dim = 3;
g.min = [-5; -5 ; -5];
g.max = -g.min;
g.dx = 0.05;
g = processGrid(g);

g2.extrapolate = 1;
g2.dim = 3;
g2.min = g.min;
g2.max = g.max;
g2.dx = 0.1;
g2 = processGrid(g2);

% visualize the grains?
show_grains = 0;

% visualize the prepared data as initial condition from the grains?
show_initial_condition = 0;

% How many grains are there?
Num_obj = length(unique(data(:)));

% original data is cylinder. make it cube.
data_cut = data(135:335,135:335,:);
% how many grains are there in the cube?
num_obj_cut = length(unique(data_cut(:)));
% we need to change the label of each grains. why? because now we have
% different number of grain, lets say it was N, no it's n. We need to have
% grain labels in data_cut_new ranging from 1:n.
data_cut_new = zeros(size(data_cut));
obj_counter = 0;
for i = 1 : max(max(max(data(:))))
    % if there is a grain with the lable i
    if ~isempty(data_cut(data_cut == i))
        % replace that grain with the label obj_counter
        obj_counter = obj_counter + 1;
        data_cut_new(data_cut == i) = obj_counter;
    end
end
```

```
% initialize the data structre which contains
LS = struct([]);

% create mesh grids required for resampling the original data to a smaller
% grid (g2). We need almost 20 grid points in each direction for each
% grains so the original scan has a high resolution for our purpose.
[X, Y, Z ] = meshgrid(g.vs{1}, g.vs{2}, g.vs{3});
[Xq, Yq, Zq ] = meshgrid(g2.vs{1}, g2.vs{2}, g2.vs{3});

j = 0;
for i = 1 : obj_counter
    % make a sign distance function from the new grain labels array. For
    % each grain, (data_cut_new==i) is a logical array with ones at the
    % grain location and it needs to be converted to a distance function

    % take care of outside of grain:
    D1 = bwdist(data_cut_new==i,'euclidean');
    % take care of inside of the grain:
    D2 = bwdist(~(data_cut_new==i),'euclidean');
    % combine the two and make it distance function by multiplying in dx.
    % note it is not perfect. why? because in both D1 and D2 the boundary
    % is zero and therefore D has a jump on the grain faces. I reinitialize
    % it to be close to a sign distance function
    D = 0.1 * (D1-D2);
    % let's interpolate the function to the new grid
    D = interp3(X,Y,Z,D,Xq,Yq,Zq,'spline');
    % let's get rid of the small particles!
    if length(D(D<0)) > 100
        % need a new counter that only consideres grains that have more
        % than 100 gridpoints in them or their volume is bigger than
        % 100*dx^3
        j = j + 1;

        % sai would be the grain level set (\psi), see JCP for \psi
        LS(j,1).sai = D;

        % we need to define the inside index, which is a binary array. It
        % will be used in grain decomposition
        LS(j,1).saiIn_Index = (LS(j,1).sai < 0);

        % decompose the computational domain for each level set function. This,
        % hugely, makes the computations efficient.
        [g2,LS] = domainDecomposition(j, g2, LS);
    end
end

% let's clear and replace it with g2.
% as g is a data structure, we cannot just say g = g2.
```

```
clear g; g = g2; clear g2;

%% Visualize
% do you want to see the resulting grain structure?
if show_grains
    % Level to be plotted
    level = 0;
    % Display type for level set
    displayType = 'surface';
    % sometimes it's a good idea to save the handle. then we can delete an
    % object in the graphic dispaly
    h = zeros(length(LS),1);
    fig = figure(1);
    set(fig,'Units', 'Normalized', 'OuterPosition', [0 0 1 1]);
    hold on
    for j = 1 : length(LS)
        h(j) = visualizeLevelSet(LS(j).grid, LS(j).sai, displayType,...
            level, [], rand(1,3));
    end
    axis(g.axis);
    daspect([ 1 1 1 ]);
    camlight ('left');
    lighting gouraud;
    set(gca, 'Visible','off')
    view([-37.5,15])
    drawnow;
end

%% Reinitialize
% I think grains first need to undergo an initialization so they become a
% sign distance function. It would not be perfect of course, but we don't
% live in a perfect world!

for i = 1 : length(LS)
    fprintf('grain %g: ', i);
    LS(i,1).sai = iterative_signDist(LS(i,1).grid, LS(i,1).sai);
end


%--------------------------------------------------------------------------
% Curvature and Normal speed parameters.
aValue = 0.15;
bValue = 0.02;
accuracy = 'medium';

% Choose approximations at appropriate level of accuracy.
switch(accuracy)
    case 'low'
        schemeData.derivFunc = @upwindFirstFirst;
```

```
        integratorFunc = @odeCFL1;
    case 'medium'
        schemeData.derivFunc = @upwindFirstENO2;
        integratorFunc = @odeCFL2;
    case 'high'
        schemeData.derivFunc = @upwindFirstENO3;
        integratorFunc = @odeCFL3;
    case 'veryHigh'
        schemeData.derivFunc = @upwindFirstWENO5;
        integratorFunc = @odeCFL3;
    otherwise
        error('Unknown accuracy level %s', accuracy);
end

%% Curvature motion
% We need to do curvature motion so the grains that are being used
% initialization of liquid level set function become a little bit curved.
% Note it's better to do it very gently, becuase the grain vanishes very
% fast with curvature motion.
% Hope it makes sense. See JCP for more info.

for i = 1 : length(LS)

    data = LS(i,1).sai;
    data0 = data;
    %---------------------------------------------------------------------
    % Integration parameters.
    % Start time.
    t0 = 0;
    % End time.
    tMax = 0.25;

    %---------------------------------------------------------------------
    % Set up motion by mean curvature with constant coefficient.
    schemeFunc = @termCurvature;
    schemeData.grid = LS(i,1).grid;
    schemeData.curvatureFunc = @curvatureSecond;
    schemeData.b = bValue;

    %---------------------------------------------------------------------
    % Set up time approximation scheme.
    integratorOptions = odeCFLset('factorCFL', 0.9, 'stats', 'on');

    tNow = t0;
    tic;
    while(tMax - tNow > 2.2204e-14 * tMax)

        % Reshape data array into column vector for ode solver call.
        y0 = data(:);
```

```
        % How far to step?
        tSpan = [ tNow tMax ];

        % Take a timestep.
        [ t, y ] = feval(integratorFunc, schemeFunc, tSpan, y0,...
            integratorOptions, schemeData);
        tNow = t(end);

        % Get back the correctly shaped data array
        data = reshape(y, LS(i,1).grid.shape);
    end

    data = iterative_signDist(LS(i,1).grid, data);
    LS(i,1).data = data;
    a = toc;
    fprintf('grian %g, curvature flow done in %g seconds\n', i, a);
end

%% Normal motion
% We need to do normal motion on the data set. So that the fluid levet set
% grows a bit and they intersect with each other on a curve not a surface.
% It activates the terms \delta(\phi_i)\delta(\phi_j) on the dihedral edges.
% Hope it makes sense. See JCP for more info.
for i = 1 : length(LS)
    data = LS(i,1).data;

    %-----------------------------------------------------------------------
    % Speed of motion normal to the interface.
    aValue = 0.25;

    %-----------------------------------------------------------------------
    % Integration parameters.
    % Start time.
    t0 = 0;
    % End time.
    tMax = 0.15;

    %-----------------------------------------------------------------------
    % Set up motion in the normal direction (derivative choice is set
    % below).
    schemeFunc = @termNormal_original;
    schemeData.grid = LS(i,1).grid;
    schemeData.speed = aValue;
    accuracy = 'medium';

    % Set up time approximation scheme.
    integratorOptions = odeCFLset('factorCFL', 0.5, 'stats', 'on');
```

```
    tNow = t0;
    tic;
    while(tMax - tNow > 2.2204e-14 * tMax)
        % Reshape data array into column vector for ode solver call.
        y0 = data(:);

        % How far to step?
        tSpan = [ tNow tMax ];

        % Take a timestep.
        [ t, y ] = feval(integratorFunc, schemeFunc, tSpan, y0,...
            integratorOptions, schemeData);
        tNow = t(end);

        % Get back the correctly shaped data array
        data = reshape(y, LS(i,1).grid.shape);
    end

    data = iterative_signDist(LS(i,1).grid, data);
    LS(i,1).data = data;
    LS(i,1).y = data(:);
    a = toc;
    fprintf('grian %g, normal flow done in %g seconds\n', i, a);
end


% We need to define a refined grid for visualization purposes and volume
% calculation. See the JCP for more details
gRef.bdry = @addGhostExtrapolate;
gRef.dim = 3;
gRef.min = g.min;
gRef.max = g.max;
gRef.dx = 0.025;
gRef = processGrid(gRef);

% last thing to consider: we need to find and save the location at which
% the grains fall in the refined grid.
for j = 1 : length(LS)
    LS(j,1).Xref = LS(j,1).grid.vs{1}(1) : gRef.dx :...
        LS(j,1).grid.vs{1}(end);
    LS(j,1).Yref = LS(j,1).grid.vs{2}(1) : gRef.dx :...
        LS(j,1).grid.vs{2}(end);
    LS(j,1).Zref = LS(j,1).grid.vs{3}(1) : gRef.dx :...
        LS(j,1).grid.vs{3}(end);
    [LS(j,1).ii(1),~] = find(abs(gRef.vs{1} - LS(j,1).Xref(1))<1e-10);
    [LS(j,1).ii(2),~] = find(abs(gRef.vs{1} - LS(j,1).Xref(end))<1e-10);
    [LS(j,1).jj(1),~] = find(abs(gRef.vs{2} - LS(j,1).Yref(1))<1e-10);
    [LS(j,1).jj(2),~] = find(abs(gRef.vs{2} - LS(j,1).Yref(end))<1e-10);
    [LS(j,1).kk(1),~] = find(abs(gRef.vs{3} - LS(j,1).Zref(1))<1e-10);
```

```
    [LS(j,1).kk(2),~] = find(abs(gRef.vs{3} - LS(j,1).Zref(end))<1e-10);
end


% you should save the created data to be used in main function. How? just
% save it with some name. Let's say we want to start simulations with theta
% 10 and phi 0.5%. Name formart that I select is this way:
% "RGphi0.5the10.mat", RG stands for Real Grains(!). I know
% here we don't have any porosity (in fact it is zero) but we can read this
% file in main.m and save it with the same name once the simulations are
% done. or once we have set the dihedral angle and porosity. For future
% simulations, we read the file with smaller porosity, increase the
% porosity in sumulations, then save it with new name (new porosity value
% in file name). We can follow the same procedure to decrease the porosity.
% right? So for the initial guess, let's save the LS data structure formed
% in here with a fake porosity, then we run simulations in main.m.

save('RGphi0.5the10.mat','LS','-v7.3');

% do you want to see the resulting grain structure?
if show_initial_condition
    % Level to be plotted
    level = 0;
    % Display type for level set
    displayType = 'surface';
    % sometimes it's a good idea to save the handle. then we can delete an
    % object in the graphic dispaly
    h = zeros(length(LS),1);
    fig = figure(2);
    set(fig,'Units', 'Normalized', 'OuterPosition', [0 0 1 1]);
    hold on
    for j = 1 : length(LS)
        h(j) = visualizeLevelSet(LS(j).grid, LS(j).data, displayType,...
            level, [], rand(1,3));
    end
    axis(g.axis);
    daspect([ 1 1 1 ]);
    camlight ('left');
    lighting gouraud;
    set(gca, 'Visible','off')
    view([-37.5,15])
    drawnow;
end



%% author: Soheil Ghanbarzadeh
%% Last version date: 06/28/2016
% Description:
```

```
% reads data from raw file, convert each to level set. It adjustes the grid
% size by interpolation of the original data set. Sets up the grid for each
% grain and initializes the liquid level set. Here I chose to do the liquid
% initialization with a sphere, you can do it with grains. How? See
% createRG.m for initial guess procedure.

% Input:
% We need some info like elongation factor, grid size (dx = 0.1 works
% best), center of the crystals, normal to faces and a point on each face
% to setup sign distance functions and create grains. See below for more
% details


% Output:
% saves a file with name TOHphi0.5The10.mat, which actually is the initial
% data structure that we work with in main.m.
% it has the required fields, including the level set function for solid
% grains and liquid as well as decomposed grid.
% --------------------------------------------------------------------------



clear; close; clc;

% elongation factor for truncated octahedron grains. See JCP.
f = 1;

% visualize the final grains?
show_grains = 0;

% create grid for domain: we need to specify the type of boundary
% condition, dimension of problem (2-D or 3-D), minimum and maximum
% values of x,y,z  and spatial increment. g.bdry handles a function for
% boundary condition and processGrid function adds required enteries to
% grid based on the information given in first steps. Note we can specify
% different g.min and g.max for x, y and z. To do this, it's enough to
% define g.min as a vector, i.e. g.min = [-1; 0; -2];
% See processGrid.m for more info.

g.extrapolate = 1;
g.dim = 3;
g.min = [-5; -5; -5 * f] - 0.5;
g.max = -g.min;
g.dx = 1 / 10;
g.bdry = @addGhostExtrapolate;
g = processGrid(g);

% we need to create the level set for the solid truncated octahedron grains
% from ground up. Each is created as the min distance between planes. each
```

```
% grain has 14 faces, therefore we have 14 normals. We need a point on each
% face, which are defined further in the code. We also need to move those
% points with the center of the grains, to create different grains in
% different locations. Note they should be space filling. So we need to
% fill some space with the grains.

% if you only need 15 grains, 1 in center (0,0,0) and 14 grains around it,
% to just place with the code. It does not give you a network, as there is
% only one grain in center.
% xcenter = [0 2 -2 0  0 0  0 1 -1  1 -1  1 -1  1 -1];
% ycenter = [0 0  0 2 -2 0  0 1  1 -1 -1  1  1 -1 -1];
% zcenter = f * [0 0  0 0  0 2 -2 1  1  1  1 -1 -1 -1 -1];


% here is the center fot 189 grains. they fill space. how? 5*5*5 = 125 and
% there are 4*4*4 = 64 grain space in between them! they are not cube. so
% you cannot stack them up to fill space. there are empty spaces in
% between. It gives you a nice network between [-4 4] in all directions.
xcenter = [-4   -4  -4  -4  -4  -4  -4  -4  -4  -4  -4  -4  -4  -4  -4 ...
    -4  -4  -4  -4  -4  -4  -4  -4  -4  -4  -3  -3  -3  -3  -3  -3  -3 ...
    -3  -3  -3  -3  -3  -3  -3  -3  -3  -2  -2  -2  -2  -2  -2  -2  -2 ...
    -2  -2  -2  -2  -2  -2  -2  -2  -2  -2  -2  -2  -2  -2  -2  -2  -2 ...
    -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1   0 ...
    0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0 ...
    0   0   0   0   0   0   0   1   1   1   1   1   1   1   1   1   1 ...
    1   1   1   1   1   1   2   2   2   2   2   2   2   2   2   2   2 ...
    2   2   2   2   2   2   2   2   2   2   2   2   2   2   3   3   3 ...
    3   3   3   3   3   3   3   3   3   3   3   3   3   4   4   4   4 ...
    4   4   4   4   4   4   4   4   4   4   4   4   4   4   4   4   4 ...
    4   4   4   4];
ycenter = [-4   -4  -4  -4  -4  -2  -2  -2  -2  -2  0   0   0   0   0 ...
    2   2   2   2   2   4   4   4   4   4   -3  -3  -3  -3  -1  -1  -1 ...
    -1  1   1   1   1   3   3   3   3   -4  -4  -4  -4  -4  -2  -2  -2 ...
    -2  -2  0   0   0   0   0   2   2   2   2   2   4   4   4   4   4 ...
    -3  -3  -3  -3  -1  -1  -1  -1  1   1   1   1   3   3   3   3   -4 ...
    -4  -4  -4  -4  -2  -2  -2  -2  -2  0   0   0   0   0   2   2   2 ...
    2   2   4   4   4   4   4   -3  -3  -3  -3  -1  -1  -1  -1  1   1 ...
    1   1   3   3   3   3   -4  -4  -4  -4  -4  -2  -2  -2  -2  -2  0 ...
    0   0   0   0   2   2   2   2   2   4   4   4   4   4   -3  -3  -3 ...
    -3  -1  -1  -1  -1  1   1   1   1   3   3   3   3   -4  -4  -4  -4 ...
    -4  -2  -2  -2  -2  -2  0   0   0   0   0   2   2   2   2   2   4 ...
    4   4   4   4];
zcenter = f * [-4   -2  0   2   4   -4  -2  0   2   4   -4  -2  0   2   4 ...
    -4  -2  0   2   4   -4  -2  0   2   4   -3  -1  1   3   -3  -1  1 ...
    3   -3  -1  1   3   -3  -1  1   3   -4  -2  0   2   4   -4  -2  0 ...
    2   4   -4  -2  0   2   4   -4  -2  0   2   4   -4  -2  0   2   4 ...
    -3  -1  1   3   -3  -1  1   3   -3  -1  1   3   -3  -1  1   3   -4 ...
    -2  0   2   4   -4  -2  0   2   4   -4  -2  0   2   4   -4  -2  0 ...
    2   4   -4  -2  0   2   4   -3  -1  1   3   -3  -1  1   3   -3  -1 ...
```

```
    1   3   -3  -1  1   3   -4  -2  0   2   4   -4  -2  0   2   4   -4 ...
    -2  0   2   4   -4  -2  0   2   4   -4  -2  0   2   4   -3  -1  1 ...
    3   -3  -1  1   3   -3  -1  1   3   -3  -1  1   3   -4  -2  0   2 ...
    4   -4  -2  0   2   4   -4  -2  0   2   4   -4  -2  0   2   4   -4 ...
    -2  0   2   4];

% initilize the level set data structure.
LS = struct([]);

%% Crystals as Level Set

% normal to plane 1 is defined by cross product of two vectors from 3
% points. The process is same for all others.
PPP1 = [1 0 sqrt(2)/2] .* [1 1 f];
PPP2 = [1 -sqrt(2)/2 0] .* [1 1 f];
PPP3 = [1 0 -sqrt(2)/2] .* [1 1 f];
n1 = cross(PPP1-PPP2,PPP1-PPP3);
n1 = n1/norm(n1,2);

% normal to plane 2 is defined by cross product of two vectors from 3
% points. The process is same for all others.
PPP1 = [-1 0 sqrt(2)/2] .* [1 1 f];
PPP2 = [-1 -sqrt(2)/2 0] .* [1 1 f];
PPP3 = [-1 0 -sqrt(2)/2] .* [1 1 f];
n2 = cross(PPP1-PPP2,PPP1-PPP3);
n2 = -n2/norm(n2,2);

% normal to plane 3 is defined by cross product of two vectors from 3
% points. The process is same for all others.
PPP1 = [0 1 sqrt(2)/2] .* [1 1 f];
PPP2 = [-sqrt(2)/2 1 0] .* [1 1 f];
PPP3 = [0 1 -sqrt(2)/2] .* [1 1 f];
n3 = cross(PPP1-PPP2,PPP1-PPP3);
n3 = -n3/norm(n3,2);

% normal to plane 4 is defined by cross product of two vectors from 3
% points. The process is same for all others.
PPP1 = [0 -1 sqrt(2)/2] .* [1 1 f];
PPP2 = [-sqrt(2)/2 -1 0] .* [1 1 f];
PPP3 = [0 -1 -sqrt(2)/2] .* [1 1 f];
n4 = cross(PPP1-PPP2,PPP1-PPP3);
n4 = n4/norm(n4,2);

% normal to plane 5 is defined by cross product of two vectors from 3
% points. The process is same for all others.
PPP1 = [sqrt(2)/2 0 1] .* [1 1 f];
PPP2 = [-sqrt(2)/2 0 1] .* [1 1 f];
PPP3 = [0 sqrt(2)/2 1] .* [1 1 f];
n5 = cross(PPP1-PPP2,PPP1-PPP3);
```

```
n5 = -n5/norm(n5,2);

% normal to plane 6 is defined by cross product of two vectors from 3
% points. The process is same for all others.
PPP1 = [sqrt(2)/2 0 -1] .* [1 1 f];
PPP2 = [-sqrt(2)/2 0 -1] .* [1 1 f];
PPP3 = [0 -sqrt(2)/2 -1] .* [1 1 f];
n6 = cross(PPP1-PPP2,PPP1-PPP3);
n6 = -n6/norm(n6,2);

% normal to plane 7 is defined by cross product of two vectors from 3
% points. The process is same for all others.
PPP1 = [1 0 sqrt(2)/2] .* [1 1 f];
PPP2 = [0 1 sqrt(2)/2] .* [1 1 f];
PPP3 = [sqrt(2)/2 0 1] .* [1 1 f];
n7 = cross(PPP1-PPP2,PPP1-PPP3);
n7 = n7/norm(n7,2);

% normal to plane 8 is defined by cross product of two vectors from 3
% points. The process is same for all others.
PPP1 = [1 0 -sqrt(2)/2] .* [1 1 f];
PPP2 = [0 1 -sqrt(2)/2] .* [1 1 f];
PPP3 = [sqrt(2)/2 0 -1] .* [1 1 f];
n8 = cross(PPP1-PPP2,PPP1-PPP3);
n8 = -n8/norm(n8,2);

% normal to plane 9 is defined by cross product of two vectors from 3
% points. The process is same for all others.
PPP1 = [-1 0 sqrt(2)/2] .* [1 1 f];
PPP2 = [0 1 sqrt(2)/2] .* [1 1 f];
PPP3 = [-sqrt(2)/2 0 1] .* [1 1 f];
n9 = cross(PPP1-PPP2,PPP1-PPP3);
n9 = n9/norm(n9,2);

% normal to plane 10 is defined by cross product of two vectors from 3
% points. The process is same for all others.
PPP1 = [-1 0 -sqrt(2)/2] .* [1 1 f];
PPP2 = [0 1 -sqrt(2)/2] .* [1 1 f];
PPP3 = [-sqrt(2)/2 0 -1] .* [1 1 f];
n10 = cross(PPP1-PPP2,PPP1-PPP3);
n10 = -n10/norm(n10,2);

% normal to plane 11 is defined by cross product of two vectors from 3
% points. The process is same for all others.
PPP1 = [1 0 sqrt(2)/2] .* [1 1 f];
PPP2 = [0 -1 sqrt(2)/2] .* [1 1 f];
PPP3 = [sqrt(2)/2 0 1] .* [1 1 f];
n11 = cross(PPP1-PPP2,PPP1-PPP3);
n11 = n11/norm(n11,2);
```

```
% normal to plane 12 is defined by cross product of two vectors from 3
% points. The process is same for all others.
PPP1 = [1 0 -sqrt(2)/2] .* [1 1 f];
PPP2 = [0 -1 -sqrt(2)/2] .* [1 1 f];
PPP3 = [sqrt(2)/2 0 -1] .* [1 1 f];
n12 = cross(PPP1-PPP2,PPP1-PPP3);
n12 = -n12/norm(n12,2);

% normal to plane 13 is defined by cross product of two vectors from 3
% points. The process is same for all others.
PPP1 = [-1 0 sqrt(2)/2] .* [1 1 f];
PPP2 = [0 -1 sqrt(2)/2] .* [1 1 f];
PPP3 = [-sqrt(2)/2 0 1] .* [1 1 f];
n13 = cross(PPP1-PPP2,PPP1-PPP3);
n13 = n13/norm(n13,2);

% normal to plane 14 is defined by cross product of two vectors from 3
% points. The process is same for all others.
PPP1 = [-1 0 -sqrt(2)/2] .* [1 1 f];
PPP2 = [0 -1 -sqrt(2)/2] .* [1 1 f];
PPP3 = [-sqrt(2)/2 0 -1] .* [1 1 f];
n14 = cross(PPP1-PPP2,PPP1-PPP3);
n14 = -n14/norm(n14,2);

% 14 points on those planes.
P1 = [2 0 0] .* [1 1 f]/2;
P2 = [-2 0 0] .* [1 1 f]/2;
P3 = [0 2 0] .* [1 1 f]/2;
P4 = [0 -2 0] .* [1 1 f]/2;
P5 = [0 0 2] .* [1 1 f]/2;
P6 = [0 0 -2] .* [1 1 f]/2;
P7 = [1 0 2] .* [1 1 f]/2;
P8 = [1 0 -2] .* [1 1 f]/2;
P9 = [1 0 -2] .* [1 1 f]/2;
P10 = [1 0 2] .* [1 1 f]/2;
P11 = [-1 0 -2] .* [1 1 f]/2;
P12 = [-1 0 2] .* [1 1 f]/2;
P13 = [-1 0 2] .* [1 1 f]/2;
P14 = [-1 0 -2] .* [1 1 f]/2;



for j = 1 : length(xcenter)

    PP1 = P1 + [xcenter(j) ycenter(j) zcenter(j)];
    PP2 = P2 + [xcenter(j) ycenter(j) zcenter(j)];
    PP3 = P3 + [xcenter(j) ycenter(j) zcenter(j)];
    PP4 = P4 + [xcenter(j) ycenter(j) zcenter(j)];
```

```
PP5 = P5 + [xcenter(j) ycenter(j) zcenter(j)];
PP6 = P6 + [xcenter(j) ycenter(j) zcenter(j)];
PP7 = P7 + [xcenter(j) ycenter(j) zcenter(j)];
PP8 = P8 + [xcenter(j) ycenter(j) zcenter(j)];
PP9 = P9 + [xcenter(j) ycenter(j) zcenter(j)];
PP10 = P10 + [xcenter(j) ycenter(j) zcenter(j)];
PP11 = P11 + [xcenter(j) ycenter(j) zcenter(j)];
PP12 = P12 + [xcenter(j) ycenter(j) zcenter(j)];
PP13 = P13 + [xcenter(j) ycenter(j) zcenter(j)];
PP14 = P14 + [xcenter(j) ycenter(j) zcenter(j)];


% find the distance of all of the points in computational domains from
% those plains.
d1 = (n1(1)*g.xs{1} + n1(2)*g.xs{2} + n1(3)*g.xs{3} - n1(1)*PP1(1)...
    - n1(2)*PP1(2) -n1(3)*PP1(3))/sqrt(n1(1)^2 + n1(2)^2 + n1(3)^2);
d2 = (n2(1)*g.xs{1} + n2(2)*g.xs{2} + n2(3)*g.xs{3} - n2(1)*PP2(1)...
    - n2(2)*PP2(2) -n2(3)*PP2(3))/sqrt(n2(1)^2 + n2(2)^2 + n2(3)^2);
d3 = (n3(1)*g.xs{1} + n3(2)*g.xs{2} + n3(3)*g.xs{3} - n3(1)*PP3(1)...
    - n3(2)*PP3(2) -n3(3)*PP3(3))/sqrt(n3(1)^2 + n3(2)^2 + n3(3)^2);
d4 = (n4(1)*g.xs{1} + n4(2)*g.xs{2} + n4(3)*g.xs{3} - n4(1)*PP4(1)...
    - n4(2)*PP4(2) -n4(3)*PP4(3))/sqrt(n4(1)^2 + n4(2)^2 + n4(3)^2);
d5 = (n5(1)*g.xs{1} + n5(2)*g.xs{2} + n5(3)*g.xs{3} - n5(1)*PP5(1)...
    - n5(2)*PP5(2) -n5(3)*PP5(3))/sqrt(n5(1)^2 + n5(2)^2 + n5(3)^2);
d6 = (n6(1)*g.xs{1} + n6(2)*g.xs{2} + n6(3)*g.xs{3} - n6(1)*PP6(1)...
    - n6(2)*PP6(2) -n6(3)*PP6(3))/sqrt(n6(1)^2 + n6(2)^2 + n6(3)^2);
d7 = (n7(1)*g.xs{1} + n7(2)*g.xs{2} + n7(3)*g.xs{3} - n7(1)*PP7(1)...
    - n7(2)*PP7(2) -n7(3)*PP7(3))/sqrt(n7(1)^2 + n7(2)^2 + n7(3)^2);
d8 = (n8(1)*g.xs{1} + n8(2)*g.xs{2} + n8(3)*g.xs{3} - n8(1)*PP8(1)...
    - n8(2)*PP8(2) -n8(3)*PP8(3))/sqrt(n8(1)^2 + n8(2)^2 + n8(3)^2);
d9 = (n9(1)*g.xs{1} + n9(2)*g.xs{2} + n9(3)*g.xs{3} - n9(1)*PP9(1)...
    - n9(2)*PP9(2) -n9(3)*PP9(3))/sqrt(n9(1)^2 + n9(2)^2 + n9(3)^2);
d10 = (n10(1)*g.xs{1} + n10(2)*g.xs{2} + n10(3)*g.xs{3} - ...
    n10(1)*PP10(1) - n10(2)*PP10(2) -n10(3)*PP10(3))/sqrt(n10(1)^2 + ...
    n10(2)^2 + n10(3)^2);
d11 = (n11(1)*g.xs{1} + n11(2)*g.xs{2} + n11(3)*g.xs{3} - ...
    n11(1)*PP11(1) - n11(2)*PP11(2) -n11(3)*PP11(3))/sqrt(n11(1)^2 + ...
    n11(2)^2 + n11(3)^2);
d12 = (n12(1)*g.xs{1} + n12(2)*g.xs{2} + n12(3)*g.xs{3} - ...
    n12(1)*PP12(1) - n12(2)*PP12(2) -n12(3)*PP12(3))/sqrt(n12(1)^2 + ...
    n12(2)^2 + n12(3)^2);
d13 = (n13(1)*g.xs{1} + n13(2)*g.xs{2} + n13(3)*g.xs{3} - ...
    n13(1)*PP13(1) - n13(2)*PP13(2) -n13(3)*PP13(3))/sqrt(n13(1)^2 + ...
    n13(2)^2 + n13(3)^2);
d14 = (n14(1)*g.xs{1} + n14(2)*g.xs{2} + n14(3)*g.xs{3} - ...
    n14(1)*PP14(1) - n14(2)*PP14(2) -n14(3)*PP14(3))/sqrt(n14(1)^2 + ...
    n14(2)^2 + n14(3)^2);

% level set for solid grain, sai, would be the maximum of the value of
```

```
    % all of d1 to d14. this makes the space betweem planes negative
    % distnace and the outside the planes, or outside the grain a positive
    % value.
    LS(j,1).sai = max(max(max(max(max(d1,d2),max(d3,d4)),...
        max(max(d5,d6),max(d7,d8))),max(max(d9,d10),...
        max(d11,d12))),max(d13,d14));

    % we need to define the inside index, which is a binary array. It
    % will be used in grain decomposition
    LS(j,1).saiIn_Index = (LS(j,1).sai < 0);

    % decompose the computational domain for each level set function. This,
    % hugely, makes the computations efficient.
    [g , LS] = domainDecomposition(j, g, LS);
end


% you can start simulations from the grain shapes. right? like what we do
% for the irregular grains. by why do that? we can initilize the liquid
% level set, in this case "data" field in LS from spheres. So initial
% condition is much closer to a constant mean curvature surface. of course
% if you put the theta 90 degrees, then you need to flip the curvature! but
% anyway I live with it this way.

% in order to have a first guess, we need to build an sphere in level set
% language. this sphere better be bigger than the grains in diameter, so
% the spheres of the different grains colide with each other and we can set
% the dihedral angle on grain-grain contacts. That's why I consider the
% radius of sphere 1.07. That's just a number.

radius = 1.07;
for j = 1 : length(LS)
    % Sign distance circle as initial level set
    LS(j,1).data = ...
        (((LS(j,1).grid.xs{1} - xcenter(j))/(radius)) .^ 2 +...
        ((LS(j,1).grid.xs{2} - ycenter(j))/(radius)) .^ 2 +...
        ((LS(j,1).grid.xs{3} - zcenter(j))/(radius*f)) .^ 2) .^ 0.5 - 1;
    % initialize the data for computations.
    LS(j,1).y = LS(j,1).data(:);
end

% Now that we are making LS, let's make the area in which the curvature
% term is active. where? inside each grain, and of course active to a few
% nodes outside it. Let's use Smeared_Heaviside function. See the detials
% there.
for j = 1 : length(LS)
    LS(j,1).ActiveCurvatureInsideSai = Smeared_Heaviside(LS(j,1).grid ,...
        - LS(j,1).sai , 3 * g.dx(1) , - 3 * g.dx(1));
end
```

```
% We need to define a refined grid for visualization purposes and volume
% calculation. See the JCP for more details
gRef.bdry = @addGhostExtrapolate;
gRef.dim = 3;
gRef.min = g.min;
gRef.max = g.max;
gRef.dx = 0.025;
gRef = processGrid(gRef);

% last thing to consider: we need to find and save the location at which
% the grains fall in the refined grid.
for j = 1 : length(LS)
    LS(j,1).Xref = LS(j,1).grid.vs{1}(1) : gRef.dx : ...
        LS(j,1).grid.vs{1}(end);
    LS(j,1).Yref = LS(j,1).grid.vs{2}(1) : gRef.dx : ...
        LS(j,1).grid.vs{2}(end);
    LS(j,1).Zref = LS(j,1).grid.vs{3}(1) : gRef.dx : ...
        LS(j,1).grid.vs{3}(end);
    [LS(j,1).ii(1),~] = find(abs(gRef.vs{1} - LS(j,1).Xref(1))<1e-10);
    [LS(j,1).ii(2),~] = find(abs(gRef.vs{1} - LS(j,1).Xref(end))<1e-10);
    [LS(j,1).jj(1),~] = find(abs(gRef.vs{2} - LS(j,1).Yref(1))<1e-10);
    [LS(j,1).jj(2),~] = find(abs(gRef.vs{2} - LS(j,1).Yref(end))<1e-10);
    [LS(j,1).kk(1),~] = find(abs(gRef.vs{3} - LS(j,1).Zref(1))<1e-10);
    [LS(j,1).kk(2),~] = find(abs(gRef.vs{3} - LS(j,1).Zref(end))<1e-10);
end



% you should save the created data to be used in main function. How? just
% save it with some name. Let's say we want to start simulations with theta
% 10 and phi 0.5%. Name formart that I select is this way:
% "TOHphi0.5the10f1.mat", TOH stands for Truncated OctaHedron. I know
% here we don't have any porosity (in fact it is zero) but we can read this
% file in main.m and save it with the same name once the simulations are
% done. or once we have set the dihedral angle and porosity. For future
% simulations, we read the file with smaller porosity, increase the
% porosity in sumulations, then save it with new name (new porosity value
% in file name). We can follow the same procedure to decrease the porosity.
% right? So for the initial guess, let's save the LS data structure formed
% in here with a fake porosity, then we run simulations in main.m.

save('TOHphi0.5the10f1.mat','LS','-v7.3');

% do you want to see the resulting grain structure?
if show_grains
    % Level to be plotted
    level = 0;
    % Display type for level set
```

```
    displayType = 'surface';
    % sometimes it's a good idea to save the handle. then we can delete an
    % object in the graphic dispaly
    h = zeros(length(LS),1);
    fig = figure(1);
    set(fig,'Units', 'Normalized', 'OuterPosition', [0 0 1 1]);
    hold on
    for j = 1 : length(LS)
        h(j) = visualizeLevelSet(LS(j).grid, LS(j).sai, displayType,...
            level, [], rand(1,3));
    end
    axis(g.axis);
    daspect([ 1 1 1 ]);
    camlight ('left');
    lighting gouraud;
    set(gca, 'Visible','off')
    view([-37.5,15])
    drawnow;
end



function [curvature, grad , gradMag ] = curvGrad(grid, data)

%% author: Soheil Ghanbarzadeh
%% Last version date: 06/28/2016
% Description:
% Computes a curvature, gradient and gradient magnitude of a level set
% function.

% Input:
% grid = Grid structure (see processGrid.m for details).
% data = Data array


% Output:
% curvature = curvature array, same size as data
% grad = 1D cell array containing centered approx to gradient
% gradMag = an array, same size az data, containing the magnitude of the
% gradient.
% -------------------------------------------------------------------------


% get the centered approximation for gradient and Hessian's terms.
[ second, first ] = hessianSecond(grid, data);
grad = first;

% get the magnitude of the gradient
gradMag2 = first{1}.^2;
```

```
for i = 2 : grid.dim
    gradMag2 = gradMag2 + first{i}.^2;
end
gradMag = sqrt(gradMag2);

% calculate the curvature
% see definition of curvature.
curvature = zeros(size(data));
for i = 1 : grid.dim;
    curvature = curvature + second{i,i} .* (gradMag2 - first{i}.^2);
    for j = 1 : i - 1
        curvature = curvature - 2 * first{i} .* first{j} .* second{i,j};
    end
end
nonzero = find(gradMag > 0);
curvature(nonzero) = curvature(nonzero) ./ gradMag(nonzero).^3;



function [grid , LS] = domainDecomposition(j, grid, LS)

%% author: Soheil Ghanbarzadeh
%% Last version date: 06/28/2016
% Description:
% Search for the location of the level set j in the main domain (Omega in
% JCP) and create a sub-domain for it. This way all the calculations for
% the level set j can be done on a small domain, while the coupling terms
% (convective and normal velocities) are trasnlated and caculated on the
% main domain. See setupVelocities.m for more details about coupling terms.

% Input:
% j = number of the grain (or level set function) which domain
% decomposition is being carried on.
% grid = Grid structure (see processGrid.m for details).
% LS = Data structure

% Output:
% grid = Grid structure (see processGrid.m for details).
% LS = Updated data structure, which now contains all the level set
% functions and grid structure for every grain.
% -------------------------------------------------------------------------


% number of grid points around a grain to include in the decompesed region.
% It seems like 5 additional grid points, on each side, works very well.
n = 5;

% initialize two two-dimensional arrays which help determining the extent
% of the level set j.
```

```
AA = zeros(grid.N(1),grid.N(2));
BB = zeros(grid.N(2),grid.N(3));


% collect the saiIm in a 2D matrix. This would be used to see where the
% bounds are in third dimension (width of the grain level set)
for i=1:grid.N(3)
    AA = AA + LS(j,1).saiIn_Index(:,:,i);
end

% collect the saiIm in a 2D matrix. This would be used to see where the
% bounds are in third dimension (width of the grain level set)
for i =1:grid.N(1)
    BB = BB + squeeze(LS(j,1).saiIn_Index(i,:,:));
end

% find the extent of functions in each direction
indexCol = any(AA);
indexRow = any(AA');
indexWid = any(BB);
firstCol = find(indexCol, 1, 'first');
lastCol = find(indexCol, 1, 'last');
firstRow = find(indexRow, 1, 'first');
lastRow = find(indexRow, 1, 'last');
firstWid = find(indexWid, 1, 'first');
lastWid = find(indexWid, 1, 'last');

% assign the location and size of new decompised domains
if firstCol < n+1
    LS(j,1).gridPlaceCol(1) = 1;
else
    LS(j,1).gridPlaceCol(1) = firstCol-n;
end
if lastCol > grid.N(2) - (n+1)
    LS(j,1).gridPlaceCol(2) = grid.N(1);
else
    LS(j,1).gridPlaceCol(2) = lastCol+n;
end

if firstRow < n+1
    LS(j,1).gridPlaceRow(1) = 1;
else
    LS(j,1).gridPlaceRow(1) = firstRow-n;
end

if lastRow > grid.N(2) - (n+1)
    LS(j,1).gridPlaceRow(2) = grid.N(2);
else
    LS(j,1).gridPlaceRow(2) = lastRow+n;
```

```
end

if firstWid < n+1
    LS(j,1).gridPlaceWid(1) = 1;
else
    LS(j,1).gridPlaceWid(1) = firstWid-n;
end

if lastWid > grid.N(3) - (n+1)
    LS(j,1).gridPlaceWid(2) = grid.N(3);
else
    LS(j,1).gridPlaceWid(2) = lastWid+n;
end

if firstWid >= n+1 && lastWid<= (grid.N(3) - (n+1))
    LS(j,1).gridPlaceWid = [firstWid-n , lastWid+n];
end

% Size of the new decomposed region.
LS(j,1).gridShape = [LS(j,1).gridPlaceRow(2) - LS(j,1).gridPlaceRow(1) + 1,...
    LS(j,1).gridPlaceCol(2) - LS(j,1).gridPlaceCol(1) + 1,...
    LS(j,1).gridPlaceWid(2) - LS(j,1).gridPlaceWid(1) + 1];

% save the location of the decomposed region (omega) in the original grid
% point (Omega)
LS(j,1).row = LS(j,1).gridPlaceRow(1):LS(j,1).gridPlaceRow(2);
LS(j,1).col = LS(j,1).gridPlaceCol(1):LS(j,1).gridPlaceCol(2);
LS(j,1).wid = LS(j,1).gridPlaceWid(1):LS(j,1).gridPlaceWid(2);

% define required inputs for creating the new grid in the decomposed
% region.
LS(j,1).grid.dim = grid.dim;
min = [grid.vs{1,1}(LS(j,1).gridPlaceRow(1)) ; ...
    grid.vs{2,1}(LS(j,1).gridPlaceCol(1)) ; ...
    grid.vs{3,1}(LS(j,1).gridPlaceWid(1))];
max = [grid.vs{1,1}(LS(j,1).gridPlaceRow(2)) ; ...
    grid.vs{2,1}(LS(j,1).gridPlaceCol(2)) ; ...
    grid.vs{3,1}(LS(j,1).gridPlaceWid(2))];
LS(j,1).grid.min = min;
LS(j,1).grid.dx = grid.dx;
LS(j,1).grid.bdry = grid.bdry;
LS(j,1).grid.max = max;

% save new grid for each decomposed level set
LS(j,1).grid = processGrid(LS(j,1).grid);

% replace the sai function with one only defined in the decomposed region
LS(j,1).sai = LS(j,1).sai([LS(j,1).row], [LS(j,1).col], [LS(j,1).wid]);
```

```
% replace the saiIn function with one only defined in the decomposed region
LS(j,1).saiIn_Index = LS(j,1).sai<0;



function [kappabar]= get_meanCurv (grid,LS)
%% author: Soheil Ghanbarzadeh
%% Last version date: 06/28/2016
% Description:
% Computes the mean curvature of solid-liquid interface using equation
% available in JCP paper.

% Input:
% grid = Grid structure (see processGrid.m for details).
% LS = Data structure, which contains all the level set functions,
% grid structure for every grain, all the required parameters that are
% called or used in other functions.


% Output:
% kappabar = mean curvature of the solid-liquid interface.
% ------------------------------------------------------------------------

AA = zeros(grid.shape);
BB = zeros(grid.shape);

if grid.dim == 2
    for j = 1 : length(LS)
        A = LS(j,1).curvature .* isNearInterface(LS(j,1).data, 0 ,1)...
            .* LS(j,1).saiIn_Index .* LS(j,1).gradMag .* grid.dx(1);
        B = isNearInterface(LS(j,1).data, 0 ,1) .* LS(j,1).saiIn_Index ...
            .* LS(j,1).gradMag .* grid.dx(1);
        AA([LS(j,1).row], [LS(j,1).col])...
            = A + AA([LS(j,1).row], [LS(j,1).col]);
        BB([LS(j,1).row], [LS(j,1).col])...
            = B + BB([LS(j,1).row], [LS(j,1).col]);
    end
else
    for j = 1 : length(LS)
        A = LS(j,1).curvature .* isNearInterface(LS(j,1).data, 0 ,1)...
            .* LS(j,1).saiIn_Index .* LS(j,1).gradMag .* grid.dx(1);
        B = isNearInterface(LS(j,1).data, 0 ,1) .* LS(j,1).saiIn_Index ...
            .* LS(j,1).gradMag .* grid.dx(1);
        AA([LS(j,1).row], [LS(j,1).col], [LS(j,1).wid])...
            = A + AA([LS(j,1).row], [LS(j,1).col], [LS(j,1).wid]);
        BB([LS(j,1).row], [LS(j,1).col], [LS(j,1).wid])...
            = B + BB([LS(j,1).row], [LS(j,1).col], [LS(j,1).wid]);
    end
end
```

```
% find the region between -4 and 4, in all direction
XX(1,1) = find(abs(grid.vs{1} - (-4))<1e-10);
XX(1,2) = find(abs(grid.vs{1} - (4))<1e-10);
XX(2,1) = find(abs(grid.vs{2} - (-4))<1e-10);
XX(2,2) = find(abs(grid.vs{2} - (4))<1e-10);
XX(3,1) = find(abs(grid.vs{3} - (-4 * grid.ep))<1e-10);
XX(3,2) = find(abs(grid.vs{3} - (4 * grid.ep))<1e-10);

% cut the arrays, to correspond to the region in
AAcut = AA(XX(1,1):XX(1,2),XX(2,1):XX(2,2),XX(3,1):XX(3,2));
BBcut = BB(XX(1,1):XX(1,2),XX(2,1):XX(2,2),XX(3,1):XX(3,2));
kappabar = sum(AAcut(:)) / sum(BBcut(:));
```

```
function [LS] = heaviside_param(grid,LS)
%% author: Soheil Ghanbarzadeh
%% Last version date: 06/28/2016
% Description:
% Computes all the required parameters and arrays required in the code. The
% function goes through all the level set functions for grains and
% calculates smeared Heaviside and delta functions, active grid points for
% surface diffusion term, curvature, gradient, gradient magnitude and sign
% function. All the computed parameters are saved as field in LS data
% structure and are used in other functions.

% Input:
% grid = Grid structure (see processGrid.m for details).
% LS = Data structure, which contains all the level set functions,
% grid structure for every grain, all the required parameters that are
% called or used in other functions.


% Output:
% LS = updated data structure, which contains all the level set functions,
% grid structure for every grain, all the required parameters that are
% called or used in other functions.
% ------------------------------------------------------------------------


for j=1:length(LS)

    % define the value of epsilon as a layer around the interface
    epsilon=1.5*grid.dx(1);

    % calculate smooth Heaviside function
    % see below
    LS(j,1).smoothHOutside = Smeared_Heaviside( LS(j,1).grid,...
```

```
    LS(j,1).data , epsilon , -epsilon);

% calulate the smooth delta function. 2D and 3D are different due to
% scaling problems.

LS(j,1).smoothdelta = zeros(size(LS(j,1).data));
HHH = zeros(size(LS(j,1).data(:)));
ind_plain = 0;
if grid.dim ==2
    if g.beta < 2 * pi/3
        epsilon = 3 * 0.05;
        ind_plain = (LS(j,1).data(:) <=  epsilon) &...
            (LS(j,1).data(:) >=  -epsilon);
        HHH(ind_plain) = 1/(1 * 0.05) * (1 +...
            (cos(pi*LS(j,1).data(ind_plain)/epsilon)));
    else
        epsilon = 3 * 0.05;
        ind_plain = (LS(j,1).data(:) <=  epsilon) &...
            (LS(j,1).data(:) >=  -epsilon);
        HHH(ind_plain) = 1/(2 * 0.05) * (1 + ...
            (cos(pi*LS(j,1).data(ind_plain)/epsilon)));
    end
else
    if grid.beta < 2 * pi/3
        epsilon = 3 * grid.dx(1);
        ind_plain = (LS(j,1).data(:) <=  epsilon) &...
            (LS(j,1).data(:) >=  -epsilon);
        HHH(ind_plain) = 1/(1 * 0.05) * (1 +...
            (cos(pi*LS(j,1).data(ind_plain)/epsilon)));
    else
        epsilon = 3 * 0.05;
        ind_plain = (LS(j,1).data(:) <=  epsilon) &...
            (LS(j,1).data(:) >=  -epsilon);
        HHH(ind_plain) = 1/(2 * 0.05) * (1 +...
            (cos(pi*LS(j,1).data(ind_plain)/epsilon)));
    end

end
LS(j,1).smoothdelta =  reshape((HHH) , LS(j,1).grid.shape);

% now compute the active places for surface diffusion motion.
% it only should occur close to interface, otherwise it would be very
% unstable.
HHH = zeros(size(LS(j,1).data(:)));
HHH(ind_plain) = 1;
LS(j,1).Active =  reshape((HHH) , LS(j,1).grid.shape);

% Get the values of curvature, gradient, and grad magnitude
[ LS(j,1).curvature, LS(j,1).grad , LS(j,1).gradMag ] =...
```

```
        curvGrad(LS(j,1).grid, LS(j,1).data);

    % Compute the sign function. It is defined in Sethian book
    LS(j,1).sign = LS(j,1).data ./ sqrt( LS(j,1).data .^2 +...
        grid.dx(1)^2 .* LS(j,1).gradMag.^2 );
end



function data = iterative_signDist(g, data)
%% author: Soheil Ghanbarzadeh
%% Last version date: 06/28/2016
% Description:
% Computes a curvature, gradient and gradient magnitude of a level set
% function.

% Input:
% grid = Grid structure (see processGrid.m for details).
% data = Data array


% Output:
% curvature = curvature array, same size as data
% grad = 1D cell array containing centered approx to gradient
% gradMag = an array, same size az data, containing the magnitude of the
% gradient.
% -------------------------------------------------------------------------

reinitGridCells = inf;
errorMax = 1e-6;
accuracy = 'medium';
g.bdry = {@addGhostExtrapolate; @addGhostExtrapolate; @addGhostExtrapolate};
tMax = min(reinitGridCells * max(g.dx), norm(g.max - g.min));
tic;
data = signedDistanceIterative(g, data, accuracy, tMax, errorMax);
a = toc;
fprintf('reinitialzed in %g seconds\n', a);



%% author: Soheil Ghanbarzadeh
%% Last version date: 06/28/2016

%% FLUID - SOLID TEXTURAL EQUILIBRIUM USING LEVEL SET METHOD
% Description:
% reads the previous set of simulation for a given dihedral angle and
% porosity and runs the simulations toward the new conditions. It sets up
% the parameters and functions required for the level set method for
% materials with texturally equilibrtaed pores. So it moves the
```

```
% solid-liquid interface toward a constant mean curvature, with constrains
% of dihedral angle and porosity. For more details see my JCP paper.

% A big portion of executed functions are in original or changed format of
% the toolbox that belongs to Ian M. Mitchell (mitchell@cs.ubc.ca). All
% required functions, whether changed or not, are included in this folder.

% I only tried and verified the change in porosity. change in dihedral
% angle is not verified and tested. So let's say I start the simulations
% with theta 30 and porosity 3%. I can set the new porosity to 4% or 2%. It
% would be better if the steps in porosities are small increments. This
% makes sure that we have quasi-steady behavior and time to equilibrate.

% I made recent changes to the function, so it does not check if the mean
% curvautre is almost constant or not. You barely can get a constant mean
% curvature as there are many terms fighting with each other and moving the
% interface in many direction. Besides, the is almost no correct way to get
% the mean curvature when you have many grains. If lucky, your curvature
% would be almost constant close to interface, but not on the interface
% itselt. The problem becomes worse when you have irregular grains. Then it
% doens't really converge. The story is same for dihedral angle. So good
% luck getting constant mean curvature or setting exact dihedral angle. but
% you can easily get a reasonbly correct and extremely beautiful results
% with this tool box. The shape of the pore network seems o.k. when you
% change the dihedral angle and porosity and that's what we need for now.
% The level set method is honestly a graphical method, not for exact
% computations. It can be exact if you have a nice and easy problem, use
% small grids and run it for infinite time.

% Therefore, I replaced the convergence conditions with just time. With try
% and error, it seems like the time between 0.0001 to 0.0003 is enough for
% convergence and gives you a reasonble results. If you are trying to run
% the simulations for first time, do it for 0.0004 and then in the next
% subseqent simulations, for higher or lower porosity 0.0001 seems like
% fine.


% Input:
% reads the file that contains the LS made in createRG or createTOH. It
% then goes through setting up the operators, functions and required data
% to run the simulations. The implementation is almost straight forward.


% Output:
% saves a file with name format of RGphiXTheY.mat, which is basically the
% results of simulations with new conditions, phi or theta. The output file
% has the required fields, including the level set function for solid
% grains and liquid as well as decomposed grid for the next set of
% simulations.
```

```
% --------------------------------------------------------------------------------




%% SETTING UP CODE & PARAMETERS:
% Clear memory, screen and figures
clear; close all; clc;

% Make sure we have all the functions needed
addpath(genpath('/Users/Soheil/Google Drive/LSM Toolbox'));

%% Set up problem
% elongation factor (anisotropy factor).
% it makes thing a bit hard, but don't change it for real grains (RG).
% f should be 1 in that case.
f = 1;

% dihedral angle
TheTa = 60;

% in (%) percent, it must be a row vector with at least two entry. the
% first one is the initial condition which will be loaded from somewhere
% that is already added to the directory.
porosity = [1 2];



%% Grid
%
% To construct grid for domain, we need to specify the type of boundary
% condition, dimension of problem (2-D or 3-D), minimum and maximum
% values of x,y,z  and spatial increment. g.bdry handles a function for
% boundary condition and processGrid function adds required enteries to
% grid based on the information given in first steps. Note we can specify
% different g.min and g.max for x, y and z. To do this, it's enough to
% define g.min as a vector, i.e. g.min = [-1; 0; -2];

% Extrapolate boundary condition
g.bdry = @addGhostExtrapolate;
% Dimension
g.dim = 3;
% for TOH instead of 5.5 (see create TOH)
% for RG use 5 (see create RG)
% % Minimum value of domain
g.min = [-5; -5; -5 * f];
% Maximum value of domain
```

```
g.max = -g.min;
% grid size. with try and error this is the best choice.
g.dx = 1 / 10;
% Magic happens here! Constructing grid. See the function for more details
g = processGrid(g);



%% Refined grid.
% We need to define a refined grid for visualization purposes and volume
% calculation. See the JCP for more details
gRef.bdry = @addGhostExtrapolate;
gRef.dim = 3;
gRef.min = g.min;
gRef.max = g.max;
gRef.dx = 0.025;
gRef = processGrid(gRef);



%% Time Integration
% This part we set the integration parameters. t0 and tMax are clear.
% Plotsteps determines the number of interval we need to come back from
% integratorFunc to the main code, plot the results and then go back again
% into integratorFunc. Here we also set the plotting parameters. Drawplot
% is a boolean paramets which determines whethere or not plot the
% intermediate plots during simulation. This could be time consuming as the
% number of level sets and crystals increases. Any way, after simulation is
% finished, final and intermediate results would be plotted.

% Start time.
t0 = 0;
% End time.
tMax = 0.0001;
% Number of intermediate plots to produce.
plotSteps = 2;
% Intermediate plotting time.
tPlot = (tMax - t0) / (plotSteps - 1);
% Time integration tolerance.
small = 100 * eps;
% Plotting the results during time integration?
drawplot = 1;
% Level to be plotted
level = 0;
% Display type for level set
displayType = 'surface';

%% Spatial and Temporal Accuracy
%
```

```
% Here we can choose approximations at appropriate level of accuracy. Same
% accuracy is used by every component of motion including reinitialization.
% Note that using high and veryHigh accuracy doesn't make any sense as we
% have curvature flow with ~O(\Delta x^2) accuracy. As the time step is
% very very small, \Delta x^4, it also doesn't make sense to use high order
% methods in time, as well.

% Set accuracy of spatial and time derivatives.
accuracy = 'low';
% Set up time approximation scheme.
integratorOptions =...
    odeCFLset('factorCFL', 0.5, 'stats', 'on');

switch(accuracy)
    case 'low'
        % Second order upwind scheme in space
        derivFunc = @upwindFirstENO2;
        % First order Runge Kutta scheme in time
        integratorFunc = @odeCFL1;
    case 'medium'
        % Second order upwind scheme in space
        derivFunc = @upwindFirstENO2;
        % Second order Runge Kutta scheme in time
        integratorFunc = @odeCFL2;
    case 'high'
        % Third order upwind scheme in space
        derivFunc = @upwindFirstENO3;
        % Third order Runge Kutta scheme in time
        integratorFunc = @odeCFL3;
    case 'veryHigh'
        % Fifth order upwind scheme in space
        derivFunc = @upwindFirstWENO5;
        % Third order Runge Kutta scheme in time
        integratorFunc = @odeCFL3;
    otherwise
        error('Unknown accuracy level %s', accuracy);
end

%% Dihedral Angle
% Here we can set the dihedral angle condition to be applied for grains in
% contact. g.contactAngle is a boolean parameter to set that. TheTa is the
% desired dihedral angle between to level sets and g.beta is the angle
% between corresponing normals of level sets.

% Angle between level sets gradient. It would be better that it goes into
% grid so that it would be passed every where.
g.beta = pi - TheTa * pi/180;
g.ep = f;
```

```
%% porosity loop

% loop that goes through all the simulations. It will be repeated n-1 time
% where n is the length of porosity. We read the first number in
% porosity, read its file name, start the simulation with its file
% and save a new file with the porosity in second member of TargetPorosity.
% This will be repeated until with have done all the target porosities.
for pc = 2:length(porosity)

    %% Importing initial condition
    %
    % Here we import the initial geometry and all other required info about
    % the grid and crystal configuration.

    % load the initial condition, which is the pc-1 member in porosity
    % array. You need to change the name for RG, it's in the line below.
%     filename = ['TOHphi' num2str(porosity(pc-1)) 'the' num2str(TheTa)...
%         'f' num2str(f) '.mat'];
    filename = ['RGphi' num2str(porosity(pc-1)) 'the' num2str(TheTa)...
        '.mat'];
    load(filename);

    %% Potting First Guess and initial crystal configuration
    %
    % If drwaplot is true, so the results would be shown periodically during
    % simulation. This parts prepare the figure to be plotted containing
    % crystals and level sets.

    % we need to plot some part of the domain. Below I am plotting the area
    % between -3 and 3 in all directions. for that, I first need to know
    % where in the refined grid I am. Array 'index' has the extent of the
    % data in original refined gird points. NOTE new dx should be dividable
    % to old dx (here 0.1). Otherwise you cannot find the extent.
    index(1,1) = find(abs(gRef.vs{1} - (-3))<1e-10);
    index(1,2) = find(abs(gRef.vs{1} - 3)<1e-10);
    index(2,1) = find(abs(gRef.vs{2} - (-3))<1e-10);
    index(2,2) = find(abs(gRef.vs{2} - 3)<1e-10);
    index(3,1) = find(abs(gRef.vs{3} - (-3 * g.ep))<1e-10);
    index(3,2) = find(abs(gRef.vs{3} - 3 * g.ep)<1e-10);


    axis_cut = [gRef.vs{1}(index(1,1)) gRef.vs{1}(index(1,2)) ...
        gRef.vs{2}(index(2,1)) gRef.vs{2}(index(2,2)) ...
        gRef.vs{3}(index(3,1)) gRef.vs{3}(index(3,2))];

    % set up the grid for the part we want to visualize
    gcut.extrapolate = 1;
    gcut.dim = 3;
    gcut.min = [axis_cut(1); axis_cut(3); axis_cut(5)];
```

```
gcut.max = [axis_cut(2); axis_cut(4); axis_cut(6)];
gcut.dx = gRef.dx(1);
gcut = processGrid(gcut);

% plot if draw plot is on
if drawplot
    % set the figure to figure 1
    fig = figure(1);
    % make the window large
    set(fig,'Units', 'Normalized', 'OuterPosition', [0 0 1 1]);
    [CCRef, ~] = RealtimeRef(g, gRef, LS);
    % cut the refined level set function
    CCcut = CCRef(index(1,1) : index(1,2), ...
        index(2,1) : index(2,2), ...
        index(3,1) : index(3,2));
    % use visualizeLevelSet function to see the pore space.
    h = visualizeLevelSet(gcut, CCcut , 'surface', 0,'', [1 0 0]);
    % Adjusting axis bounds
    axis(axis_cut);
    % hide the axis
    set(gca, 'Visible','off')
    % Adjusting aspect ratio
    daspect([ 1 1 1 ]);
    % set lighting for the graphic environment
    camlight ('left');
    lighting gouraud;
    drawnow;
end


%% MOTION TERMS: Laplacian of Curvature Flow
% Speed of diffusive motion normal to the interface.
bValue = 0.1;
curvFunc = @termLapCurvature;
curvData.grid = g;
curvData.curvatureFunc = @curvatureSecond;
curvData.b = bValue;


%% MOTION TERMS: Normal Velocity Flow
% set up the normal motion. velocities for this motion are determined
% in setupVelocities.m
normalFunc = @termNormal;
normalData.grid = g;
normalData.derivFunc = derivFunc;
normalData.tagetPor = porosity(pc);


%% MOTION TERMS: Convective Flow
```

```
% set up the convective flow. velocities for this motion are determined
% in setupVelocities.m
convFunc = @termConvection;
convData.grid = g;
convData.derivFunc = derivFunc;


%% Combine components of motion.
% we need to combine all the motions
schemeFunc = @termSum;
schemeData.innerFunc = {curvFunc; normalFunc; convFunc};
schemeData.innerData = {curvData; normalData; convData};
schemeData.grid = g;
schemeData.gRef = gRef;
schemeData.accuracy = accuracy;

%% Reinitilization parameters.
% do you want to einitialize?
schemeData.reinitflag = 1;
% how often do you want to reinitialize. how many time steps to take to
% enter the reinitialization process? I would say 2 or 3 but here I set
% it to large number to save some time in demoing the software
schemeData.u = 5;


%% SETTING UP MAIN TIME LOOP AND COUNTERS
% we need to reset the time before the main time loop
tNow = t0;
tic

%% MAIN TIME LOOP
% Loop until tMax (subject to a little roundoff) reached.
while(tMax - tNow > small * tMax)

    % How far to step?
    tSpan = [ tNow, min(tMax, tNow + tPlot) ];

    % Take the timesteps to complete tSpan.
    [ t , LS, CCRef ] = feval(integratorFunc, schemeFunc, tSpan, LS,...
        integratorOptions, schemeData);

    tNow = t(end);

    if drawplot
        figure(1)
        delete(h);
        CCcut = CCRef(index(1,1) : index(1,2),...
            index(2,1) : index(2,2),...
            index(3,1) : index(3,2));
```

```
            h = visualizeLevelSet(gcut, CCcut , 'surface', 0, '', [1 0 0]);
            set(gca, 'Visible','off')
            axis(axis_cut);
            daspect([ 1 1 1 ]);
            drawnow;
        end
    end

    %% FINALIZING SIMULATION, SAVING RESULTS, SENDING OUT NOTIFICATION

    % new file name, with the current porosity
    filename = ['RGphi' num2str(porosity(pc)) 'the' num2str(TheTa)...
        'f' num2str(f) '.mat'];
    save(filename, 'LS', '-v7.3');
    endTime = toc;
    fprintf(['\n Total execution time for the current porosity is %g'...
        'seconds\n'], endTime);
end



function [CCRef, por] = realtimeRef(grid, grid_Ref, LS)

%% author: Soheil Ghanbarzadeh
%% Last version date: 06/28/2016
% Description:
% Refines the level set functions (phi) and then turns them into Phi, see
% JCP for the detail. The final solid-liquid interface is defines as zero
% level set of the Phi (capitalized phi).

% Input:
% grid = Grid structure (see processGrid.m for details).
% grid_Ref = Refined grid structure (see processGrid.m for details).
% LS = Data structure, which contains all the level set functions,
% grid structure for every grain, all the required parameters that are
% called or used in other functions.


% Output:
% grid_Ref = updated refine grid structure.
% CCRef = equivalent of Phi (cap phi)
% por = porosity at the current time. It is obtinaed after refinement. See
% JCP paper for more details.
% -------------------------------------------------------------------------

if grid.dim == 2
    % initialize CC. It needs to be large, in this case 100 as we are
    % finding minimum.
    CC = 100*ones(grid_Ref.shape);
```

```
    % go through all the level set function
    for j = 1 : length(LS)
        % create refined grid for each level set function
        [Xq,Yq] = ndgrid(LS(j,1).Xref, LS(j,1).Yref);

        % fit the spline function
        intPolFunc = griddedInterpolant(LS(j,1).grid.xs{:},...
            LS(j,1).data, 'spline');

        % evaluate the fitted function in refined grid points
        DataRef = intPolFunc(Xq, Yq);
        if j == 1
            LS(1,1).outsideRef = DataRef > 0;
        end

        % select the region related to level set j. we should save the
        % current value in the region level set j exists. therefore a
        % CCselect array is selected from CC.
        CCselect = CC(LS(j,1).ii(1):LS(j,1).ii(2),...
            LS(j,1).jj(1):LS(j,1).jj(2));

        % now we should fild the minimum between CC (here CCselect) and
        % DataRef which is refined level set function.
        CC(LS(j,1).ii(1):LS(j,1).ii(2), LS(j,1).jj(1):LS(j,1).jj(2))...
            = min(CCselect, DataRef);
    end
    CCRef = CC;
    A = LS(1,1).outsideRef .* LS(1,1).insideSaiRef;
    por = length(A(A==1))/length(A(LS(1,1).insideSaiRef == 1));
end


if grid.dim == 3
    % initialize CC. It needs to be large, in this case 100 as we are
    % finding minimum.
    CC = 100*ones(grid_Ref.shape);

    % go through all the level set function
    for j = 1 : length(LS)
        % create refined grid for each level set function
        [Xq,Yq,Zq] = ndgrid(LS(j,1).Xref, LS(j,1).Yref, LS(j,1).Zref);

        % fit the spline function
        intPolFunc = griddedInterpolant(LS(j,1).grid.xs{:}, LS(j,1).data,...
            'spline');

        % evaluate the fitted function in refined grid points
        DataRef = intPolFunc(Xq, Yq, Zq);
```

```
        % select the region related to level set j. we should save the
        % current value in the region level set j exists. therefore a
        % CCselect array is selected from CC.
        CCselect = CC(LS(j,1).ii(1):LS(j,1).ii(2),...
            LS(j,1).jj(1):LS(j,1).jj(2), LS(j,1).kk(1):LS(j,1).kk(2));

        % now we should fild the minimum between CC (here CCselect) and
        % DataRef which is refined level set function.
        CC(LS(j,1).ii(1):LS(j,1).ii(2), LS(j,1).jj(1):LS(j,1).jj(2),...
            LS(j,1).kk(1):LS(j,1).kk(2)) = min(CCselect, DataRef);
    end


    CCRef = CC;

    % find the region between -4 and 4, in all direction
    XX(1,1) = find(abs(grid_Ref.vs{1} - (-4))<1e-10);
    XX(1,2) = find(abs(grid_Ref.vs{1} - (4))<1e-10);
    XX(2,1) = find(abs(grid_Ref.vs{2} - (-4))<1e-10);
    XX(2,2) = find(abs(grid_Ref.vs{2} - (4))<1e-10);
    XX(3,1) = find(abs(grid_Ref.vs{3} - (-4 * grid.ep))<1e-10);
    XX(3,2) = find(abs(grid_Ref.vs{3} - (4 * grid.ep))<1e-10);

    % cut the CC array, to correspond to the region in
    CCcut = CC(XX(1,1):XX(1,2),XX(2,1):XX(2,2),XX(3,1):XX(3,2));

    % compute porosity
    por = length(CCcut(CCcut > 0))/length(CCcut(:));
end



function [ LS, g] = reinit_fun(LS, g, accuracy)
%% author: Soheil Ghanbarzadeh
%% Last version date: 06/28/2016

% Description:
% The function is called every few time steps in ode solver, and goes
% through all the level set functions in the LS data structure and
% reinitializes their values. For more information look at the termReinit
% function. The goal is to get a sign distance function, at least very
% close to the interface, for all the level set functions in LS.


% Input:
% LS = Level Set data structure.
% g = grid structure
% accuracy = spatial and temporal accuracy
```

```
% Output:
% LS = Updated the data structure, with reinitialized level set functions.
% g = grid remains unchanged.



% Set up time approximation scheme.
integratorOptions = odeCFLset('factorCFL', 0.5, 'stats', 'off');

% Choose approximations at appropriate level of accuracy.
switch(accuracy)
    case 'low'
        derivFunc = @upwindFirstENO2;
        integratorFunc = @odeCFL1_init;
    otherwise
        error('Unknown accuracy level %s', accuracy);
end


singleStep = 0;
if(singleStep)
    integratorOptions = odeCFLset(integratorOptions, 'singleStep', 'on');
end

% Set up spatial approximation scheme.
schemeFunc = @termReinit;

schemeData.derivFunc = derivFunc;

% Use the subcell fix by default.
schemeData.subcell_fix_order = 0;
schemeData.reinitflag = 0;


%----------------------------------------------------------------------------
% Loop until tMax (subject to a little roundoff).
for j = 1 : length(LS)
    tPlot = 0.02;                % Period at which plot should be produced.
    t0 = 0;                      % Start time.
    tNow = t0;
    s1 = cputime;
    fprintf('\t \t Reinitializing Level Set Number :  %d ... ',j);


    % set the max time to 0.7. It does not matter to do more than that
    % time. because the only imporntant place is near the interface and by
    % that time, the changes have already propagated from the interface.
```

```
    while tNow < 1
        schemeData.initial = LS(j,1).data;
        schemeData.grid = LS(j,1).grid;

        % Reshape data array into column vector for ode solver call.
        y0 =  LS(j,1).data(:);

        % How far to step?
        tSpan = [ tNow,  tNow + tPlot];

        % Take a timestep.
        [ t , y ] = feval(integratorFunc, schemeFunc, tSpan, y0,...
            integratorOptions, schemeData);
        tNow = t(end);

        % reshape the data to a matrix
        LS(j,1).data = reshape(y, LS(j,1).gridShape);

    end

    % display the time it took to reinit the level set function j
    n1 = cputime;
    fprintf(' DONE in tMax = %g!!! (%g seconds) \n',tNow, n1-s1);
end



function [grid , LS] = setupVelocities(grid, LS, MeanCurvature, dPor)
%% author: Soheil Ghanbarzadeh
%% Last version date: 06/28/2016
% Description:
% Computes the normal and convective velocities. For equations see the JCP
% paper. It saves the velocities as fields in LS structure.

% Input:
% grid = Grid structure (see processGrid.m for details).
% LS = Data structure, which contains all the level set functions,
% grid structure for every grain, all the required parameters that are
% called or used in other functions.
% MeanCurvature = mean curvature of the solid liquid interface.
% deltaV = relative difference between target porosity and the current
% porosity


% Output:
% grid = Grid structure (see processGrid.m for details).
% LS = updated data structure, which contains all the level set functions,
% grid structure for every grain, all the required parameters that are
% called or used in other functions.
```

```
% ------------------------------------------------------------------------

% create a list of level set functions. name of the functions are their
% number.
list = 1 : length(LS);

% scale the dPor. See the JCP paper for details.
dPorscaled = 5*(dPor) * (exp(abs(dPor)))/(0.05^(grid.dim-1));

if grid.dim == 2
    for j = 1 : length(LS)

        % convective speed has three components: x, y and z. Need a cell
        % type entry to save them all.
        speedConv = cell(grid.dim,1);
        SPC = cell(grid.dim,1);

        % remove the level set j from the list.
        lista = setdiff(list,j);

        % Initialize the normal and convective level sets with zero.
        for k = 1: grid.dim
            SPC{k,1} = zeros(grid.shape);
        end
        SPN = zeros(grid.shape);

        % Go through the remaining level set functions. all other than
        % level set j.
        for i = 1 : length(LS) - 1

            % update to SPN related to level set i
            speedNorm = LS(lista(i),1).smoothdelta .* ...
                LS(lista(i),1).gradMag .* LS(lista(i),1).sign;
            SPN([LS(lista(i),1).row], [LS(lista(i),1).col])...
                = speedNorm + SPN([LS(lista(i),1).row],...
                [LS(lista(i),1).col]);

            % go through all dimensions for convective velocity
            for k = 1:grid.dim
                speedConv{k,1} = LS(lista(i),1).smoothdelta .*...
                    LS(lista(i),1).grad{k,1} .* LS(lista(i),1).sign;
                SPC{k,1}([LS(lista(i),1).row], [LS(lista(i),1).col])...
                    = speedConv{k,1} + SPC{k,1}([LS(lista(i),1).row],...
                    [LS(lista(i),1).col]);
            end

        end

        % update the convective velocities in all directions.
```

```
        for k = 1:grid.dim
            LS(j,1).ConvectionSpeed{k,1} = (1/(1)) .* LS(j,1).smoothdelta ...
                .* SPC{k,1}([LS(j,1).row], [LS(j,1).col]);
        end

        % include all the required terms in normal velocity, including cos
        % of pp-theta (beta), delta functions, mean carvature and also
        % scaled normal velocity term from dPor.
        LS(j,1).Normalspeed = (-cos(grid.beta)/(1)) .* LS(j,1).smoothdelta...
            .* SPN([LS(j,1).row], [LS(j,1).col])...
            + dPorscaled  + MeanCurvature * LS(j,1).Active .* ...
            LS(j,1).ActiveCurvatureInsideSai;
        LS(j,1).NormOnlycont = (-cos(grid.beta)) * LS(j,1).smoothdelta .*...
            SPN([LS(j,1).row], [LS(j,1).col]);
    end
else
    for j = 1 : length(LS)

        % convective speed has three components: x, y and z. Need a cell
        % type entry to save them all.
        speedConv = cell(grid.dim,1);
        SPC = cell(grid.dim,1);

        % Initialize the normal and convective level sets with zero.
        for k = 1: grid.dim
            SPC{k,1} = zeros(grid.shape);
        end
        SPN = zeros(grid.shape);

        % remove the level set j from the list.
        lista = setdiff(list,j);

        % Go through the remaining level set functions. all other than
        % level set j.
        for i = 1 : length(LS) - 1
            % update to SPN related to level set i
            speedNorm = LS(lista(i),1).smoothdelta .* ...
                LS(lista(i),1).gradMag .* LS(lista(i),1).sign;
            SPN([LS(lista(i),1).row], [LS(lista(i),1).col],...
                [LS(lista(i),1).wid])...
                = speedNorm + SPN([LS(lista(i),1).row],...
                [LS(lista(i),1).col], [LS(lista(i),1).wid]);

            % go through all dimensions for convective velocity
            for k = 1 : grid.dim
                speedConv{k,1} = LS(lista(i),1).smoothdelta .* ...
                    LS(lista(i),1).grad{k,1} .* LS(lista(i),1).sign;
                SPC{k,1}([LS(lista(i),1).row], [LS(lista(i),1).col],...
                    [LS(lista(i),1).wid])...
```

```
                    = speedConv{k,1} + SPC{k,1}([LS(lista(i),1).row],...
                    [LS(lista(i),1).col], [LS(lista(i),1).wid]);
            end
        end

        % update the convective velocities in all directions.
        for k = 1 : grid.dim
            LS(j,1).ConvectionSpeed{k,1} = (1/(1)) .* LS(j,1).smoothdelta...
                .* SPC{k,1}([LS(j,1).row], [LS(j,1).col], [LS(j,1).wid]);
        end

        % include all the required terms in normal velocity, including cos
        % of pp-theta (beta), delta functions, mean carvature and also
        % scaled normal velocity term from dPor.
        LS(j,1).Normalspeed = (-cos(grid.beta)/(1)) .* LS(j,1).smoothdelta...
            .* SPN([LS(j,1).row], [LS(j,1).col], [LS(j,1).wid])...
            + dPorscaled + MeanCurvature * LS(j,1).Active .* ...
            LS(j,1).ActiveCurvatureInsideSai;
    end
end



function [H] = smeared_Heaviside(grid, data , upper , lower)

%% author: Soheil Ghanbarzadeh
%% Last version date: 06/28/2016
% Description:
% Computes the smeared Heaviside function for a level set array. the upper
% and lower limits, in terms of epsion, are defined by user.

% Input:
% grid = Grid structure (see processGrid.m for details).
% data = Data array
% upper = upper limit for epsilon.
% lower = lower limit for epsilon.


% Output:
% H = array same size as data, containing smeared Heaviside function with
% provided upper and lower limits.
% -------------------------------------------------------------------------


H = zeros(size(data(:)));
ind_plain = data(:) < lower;
H(ind_plain) = 0;
ind_plain = data(:) > upper;
H(ind_plain) = 1;
```

```
ind_plain = (data(:) <=  upper) & (data(:) >= lower);
H(ind_plain) = 0.5 + (data(ind_plain) - (upper+lower)/2) ...
    /(2*abs(upper-lower))...
    + (1/(2*pi)) * (sin(pi*(data(ind_plain) - (upper+lower)/2) ...
    /abs(upper-lower)));
H = reshape(H , grid.shape);



clear; close; clc;

%% author: Soheil Ghanbarzadeh
%% Last version date: 06/28/2016

%% FLUID - SOLID TEXTURAL EQUILIBRIUM USING LEVEL SET METHOD
% Description:
% reads the previous set of simulation for a given dihedral angle and
% porosity and runs the simulations toward the new conditions. It sets up
% the parameters and functions required for the level set method for
% materials with texturally equilibrtaed pores. So it moves the
% solid-liquid interface toward a constant mean curvature, with constrains
% of dihedral angle and porosity. For more details see my JCP paper.

% Input:
% reads the file that contains the LS made in createRG or createTOH. It
% then goes through setting up the operators, functions and required data
% to run the simulations. The implementation is almost straight forward.


% Output:
% saves a file with name format of RGphiXTheY.mat, which is basically the
% results of simulations with new conditions, phi or theta. The output file
% has the required fields, including the level set function for solid
% grains and liquid as well as decomposed grid for the next set of
% simulations.
% ------------------------------------------------------------------------

% set the new grid size for visualization
dxnew = 0.025;
% what is dihedral angle? it can be array to go through multiple final
% simulation data and plot them all.
TheTa = [30];
% whay is the porosity? it can be array to go through multiple final
% simulation data and plot them all.
porosity = [0.5];
% elongation factor, it only goes into the name of file.
f = 1;

%% Grid
```

```
% Extrapolate boundary condition
g.bdry = @addGhostExtrapolate;
% Dimension
g.dim = 3;
% for TOH instead of 5.5 (see create TOH)
% for RG use 5 (see create RG)
% % Minimum value of domain
g.min = [-5.5; -5.5; -5.5 * f];
% Maximum value of domain
g.max = -g.min;
% grid size. with try and error this is the best choice.
g.dx = 1 / 10;
g.ep = f;
% Magic happens here! Constructing grid. See the function for more details
g = processGrid(g);


%% Refined grid.
% We need to define a refined grid for visualization purposes and volume
% calculation. See the JCP for more details
gRef.bdry = @addGhostExtrapolate;
gRef.dim = 3;
gRef.min = g.min;
gRef.max = g.max;
gRef.dx = dxnew;
gRef = processGrid(gRef);


for tc = 1 : length(TheTa)
    for pc = 1 : length(porosity)
        filename = ['TOHphi' num2str(porosity(pc)) 'the' num2str(TheTa)...
            'f' num2str(f) '.mat'];
%         filename = ['RGphi' num2str(porosity(pc-1)) 'the' num2str(TheTa)...
%             '.mat'];
        load(filename);

        % We need to re-define the extension of each grain in new grid
        for j = 1 : length(LS)
            LS(j,1).Xref = LS(j,1).grid.vs{1}(1) : gRef.dx :...
                LS(j,1).grid.vs{1}(end);
            LS(j,1).Yref = LS(j,1).grid.vs{2}(1) : gRef.dx :...
                LS(j,1).grid.vs{2}(end);
            LS(j,1).Zref = LS(j,1).grid.vs{3}(1) : gRef.dx :...
            LS(j,1).grid.vs{3}(end);
            [LS(j,1).ii(1),~] = find(abs(gRef.vs{1} - ...
                LS(j,1).Xref(1))<1e-10);
            [LS(j,1).ii(2),~] = find(abs(gRef.vs{1} - ...
                LS(j,1).Xref(end))<1e-10);
            [LS(j,1).jj(1),~] = find(abs(gRef.vs{2} - ...
```

```
        LS(j,1).Yref(1))<1e-10);
    [LS(j,1).jj(2),~] = find(abs(gRef.vs{2} - ...
        LS(j,1).Yref(end))<1e-10);
    [LS(j,1).kk(1),~] = find(abs(gRef.vs{3} - ...
        LS(j,1).Zref(1))<1e-10);
    [LS(j,1).kk(2),~] = find(abs(gRef.vs{3} - ...
        LS(j,1).Zref(end))<1e-10);
end

% we need to plot some part of the domain. Below I am plotting the
% area between -3 and 3 in all directions. for that, I first need
% to know where in the refined grid I am. Array 'index' has the
% extent of the data in original refined gird points. NOTE new dx
% should be dividable to old dx (here 0.1). Otherwise you cannot
% find the extent.
index(1,1) = find(abs(gRef.vs{1} - (-3))<1e-10);
index(1,2) = find(abs(gRef.vs{1} - 3)<1e-10);
index(2,1) = find(abs(gRef.vs{2} - (-3))<1e-10);
index(2,2) = find(abs(gRef.vs{2} - 3)<1e-10);
index(3,1) = find(abs(gRef.vs{3} - (-3 * f))<1e-10);
index(3,2) = find(abs(gRef.vs{3} - 3 * f)<1e-10);


axis_cut = [gRef.vs{1}(index(1,1)) gRef.vs{1}(index(1,2)) ...
    gRef.vs{2}(index(2,1)) gRef.vs{2}(index(2,2)) ...
    gRef.vs{3}(index(3,1)) gRef.vs{3}(index(3,2))];

% set up the grid for the part we want to visualize
gcut.extrapolate = 1;
gcut.dim = 3;
gcut.min = [axis_cut(1); axis_cut(3); axis_cut(5)];
gcut.max = [axis_cut(2); axis_cut(4); axis_cut(6)];
gcut.dx = gRef.dx(1);
gcut = processGrid(gcut);


% set the figure to figure 1
fig = figure(1);
% make the window large
set(fig,'Units', 'Normalized', 'OuterPosition', [0 0 1 1]);
[CCRef, ~] = RealtimeRef(g, gRef, LS);
% cut the refined level set function
CCcut = CCRef(index(1,1) : index(1,2), ...
    index(2,1) : index(2,2), ...
    index(3,1) : index(3,2));
% use visualizeLevelSet function to see the pore space.
h = visualizeLevelSet(gcut, CCcut , 'surface', 0,'', [1 0 0]);
% Adjusting axis bounds
axis(gcut.axis);
```

```
% hide the axis
set(gca, 'Visible','off')
% Adjusting aspect ratio
daspect([ 1 1 1 ]);
% set lighting for the graphic environment
camlight('left')
lighting phong
% set the view and angle of camera
view([-37.5,15])
% force the plot
drawnow;

%% draw abox around the graphic object.
% MATLAB really socks when you set the box to on.
x1 = linspace(gcut.xs{1}(1),gcut.xs{1}(end),21);
y1= gcut.xs{2}(1) .* ones(size(x1));
z1 = gcut.xs{3}(1) .* ones(size(x1));
plot3(x1,y1,z1,'k','linewidth',3)

x2 = linspace(gcut.xs{1}(1),gcut.xs{1}(end),21);
y2= gcut.xs{2}(end) .* ones(size(x1));
z2 = gcut.xs{3}(1) .* ones(size(x1));
plot3(x2,y2,z2,'k','linewidth',3)

x3 = linspace(gcut.xs{1}(1),gcut.xs{1}(end),21);
y3= gcut.xs{2}(1) .* ones(size(x1));
z3 = gcut.xs{3}(end) .* ones(size(x1));
plot3(x3,y3,z3,'k','linewidth',3)

x4 = linspace(gcut.xs{1}(1),gcut.xs{1}(end),21);
y4= gcut.xs{2}(end) .* ones(size(x1));
z4 = gcut.xs{3}(end) .* ones(size(x1));
plot3(x4,y4,z4,'k','linewidth',3)

x5 = gcut.xs{1}(1) .* ones(size(x1));
y5= linspace(gcut.xs{2}(1),gcut.xs{2}(end),21);
z5 = gcut.xs{3}(1) .* ones(size(x1));
plot3(x5,y5,z5,'k','linewidth',3)

x6 = gcut.xs{1}(end) .* ones(size(x1));
y6= linspace(gcut.xs{2}(1),gcut.xs{2}(end),21);
z6 = gcut.xs{3}(1) .* ones(size(x1));
plot3(x6,y6,z6,'k','linewidth',3)

x7 = gcut.xs{1}(1) .* ones(size(x1));
y7= linspace(gcut.xs{2}(1),gcut.xs{2}(end),21);
z7 = gcut.xs{3}(end) .* ones(size(x1));
plot3(x7,y7,z7,'k','linewidth',3)
```

```
        x8 = gcut.xs{1}(end) .* ones(size(x1));
        y8= linspace(gcut.xs{2}(1),gcut.xs{2}(end),21);
        z8 = gcut.xs{3}(end) .* ones(size(x1));
        plot3(x8,y8,z8,'k','linewidth',3)

        x9 = gcut.xs{1}(1) .* ones(size(x1));
        y9= gcut.xs{2}(1) .* ones(size(x1));
        z9 = linspace(gcut.xs{3}(1),gcut.xs{3}(end),21);
        plot3(x9,y9,z9,'k','linewidth',3)

        x10 = gcut.xs{1}(end) .* ones(size(x1));
        y10= gcut.xs{2}(1) .* ones(size(x1));
        z10 = linspace(gcut.xs{3}(1),gcut.xs{3}(end),21);
        plot3(x10,y10,z10,'k','linewidth',3)

        x11 = gcut.xs{1}(1) .* ones(size(x1));
        y11= gcut.xs{2}(end) .* ones(size(x1));
        z11 = linspace(gcut.xs{3}(1),gcut.xs{3}(end),21);
        plot3(x11,y11,z11,'k','linewidth',3)

        x12 = gcut.xs{1}(end) .* ones(size(x1));
        y12= gcut.xs{2}(end) .* ones(size(x1));
        z12 = linspace(gcut.xs{3}(1),gcut.xs{3}(end),21);
        plot3(x12,y12,z12,'k','linewidth',3)

    end
end



function [ second, first ] = hessianForth(grid, data)

%% author: Soheil Ghanbarzadeh
%% Last version date: 06/28/2016
% modified from and insired by hessianSecond.m function by Ian Mitchell in
% the level set toolbox. See ref. for more info.

% Description:
% Computes a forth order centered difference approximation to the Hessian
% (the forth order mixed spatial derivative of the data).

            % Input:
            % grid = Grid structure (see processGrid.m for details).
            % data = Data array


            % Output:
            % second = 2D cell array containing centered approx to
            % Hessian's terms.
```

```
            % To save space, only lower left half of Hessian is given
            % (since mixed partials are derivative order independent).
                % second{i,j} = d^2 data / dx_i dx_j   if j < i
                %             = d^2 data / dx_i^2      if j = i
                %             =[]                      if j > i
            % first = 1D cell array containing centered approx to gradient
            % (incidentally computed while finding second).
                % first{i}    = d data / dx_i

%--------------------------------------------------------------------------
dxInv = 1 ./ grid.dx;

% How big is the stencil?
stencil = 2;

% Add ghost cells to every dimension.
data = addGhostAllDims(grid, data, stencil);

%--------------------------------------------------------------------------
% We need indices to the real data.
indReal = cell(grid.dim, 1);
for i = 1 : grid.dim
    indReal{i} = 1 + stencil : grid.N(i) + stencil;
end

% Also indices to the whole data set (including ghost cells).
indAll = cell(grid.dim, 1);
for i = 1 : grid.dim
    indAll{i} = 1 : grid.N(i) + 2 * stencil;
end

%--------------------------------------------------------------------------
% Centered first partials (gradient approximation).
first = cell(grid.dim, 1);
for i = 1 : grid.dim
    % leave the ghost cells on other dimensions intact
    % (for mixed partials below)
    indices1 = indAll;
    indices2 = indAll;
    indices3 = indAll;
    indices4 = indAll;
    indices1{i} = indReal{i} + 1;
    indices2{i} = indReal{i} - 1;
    indices3{i} = indReal{i} + 2;
    indices4{i} = indReal{i} - 2;
    first{i} = (1/12) * dxInv(i) * ( - data(indices3{:}) + 8 * data(indices1{:})...
        - 8 * data(indices2{:}) + data(indices4{:}));
end
```

```
%--------------------------------------------------------------------------
% Centered second partials (Hessian approximation).
%   We will only fill the lower half of second,
%   since mixed partials' derivative ordering doesn't matter.
second = cell(grid.dim, grid.dim);

for i = 1 : grid.dim
    % First, the pure second partials.
    %   Get rid of ghost cells on other dimensions.
    indices1 = indReal;
    indices2 = indReal;
    indices3 = indReal;
    indices4 = indReal;
    indices1{i} = indices1{i} + 1;
    indices2{i} = indices2{i} - 1;
    indices3{i} = indices3{i} + 2;
    indices4{i} = indices4{i} - 2;
    second{i,i} = (1/12) * dxInv(i).^2 * ( -data(indices3{:}) +....
        16 * data(indices1{:}) - 30 * data(indReal{:}) ...
        + 16 * data(indices2{:}) - data(indices4{:}));

    % Now the mixed partials.
    for j = 1 : i - 1
        % Get rid of ghost cells in dimensions without derivatives.
        indices1 = indReal;
        indices2 = indReal;
        indices3 = indReal;
        indices4 = indReal;
        % In already differentiated dimension, we have no ghost cells.
        indices1{i} = 1 : grid.N(i);
        indices2{i} = 1 : grid.N(i);
        indices3{i} = 1 : grid.N(i);
        indices4{i} = 1 : grid.N(i);
        % Now take a centered difference in second direction.
        indices1{j} = indReal{j} + 1;
        indices2{j} = indReal{j} - 1;
        indices3{j} = indReal{j} + 2;
        indices4{j} = indReal{j} - 2;

        second{i,j} = (1/12) * dxInv(j) * ( - first{i}(indices3{:}) + ...
            8 * first{i}(indices1{:})...
            - 8 * first{i}(indices2{:}) + first{i}(indices4{:}));
    end
end

%--------------------------------------------------------------------------
% If the user wants the gradient approximation,
%   strip unnecessary ghost cells from first partials.
```

```
for i = 1 : grid.dim
    indices1 = indReal;
    % In already differentiated dimension, we have no ghost cells.
    indices1{i} = 1 : grid.N(i);
    first{i} = first{i}(indices1{:});
end



function [ t, y, schemeData ] = ...
    odeCFL1_init(schemeFunc, tspan, y0, options, schemeData)

%% author: Soheil Ghanbarzadeh
%% author of original function: Ian Mitchell, @ level set toolbox
%% Last version date: 06/28/2016
% modified from odeCFL1.m function by Ian Mitchell in the level set
% toolbox. See the origninal function, toolbox and related paper (Mitchell
% 2008) for more information.

% Description:
% Integrates a system forward in time by CFL constrained timesteps
% using a first order forward Euler scheme (which happens to be the first
% order TVD RK scheme).


% Input:
% schemeFunc = Function handle to a CFL constrained ODE system. See
% original function by Ian Mitchell for more information.
% tspan = Range of time over which to integrate (see below).
% y0 = Initial condition vector
% options = An option structure generated by odeCFLset
% schemeData = Structure passed through to schemeFunc. See original
% function by Ian Mitchell for more information.


% Output:
% t = Output time(s) (see below).
% y = Output state (see below).
% schemeData = Output version of schemeData. See original function by Ian
% Mitchell for more information.


%---------------------------------------------------------------------------
% How close (relative) do we need to be to the final time?
small = 100 * eps;

%---------------------------------------------------------------------------
% Make sure we have the default options settings
if((nargin < 4) | isempty(options))
```

```
    options = odeCFLset;
end

%---------------------------------------------------------------------------
% Number of timesteps to be returned.
numT = length(tspan);

%---------------------------------------------------------------------------
% If we were asked to integrate forward to a final time.
if(numT == 2)

    % - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
    % Is this a vector level set integration?
    if(iscell(y0))
        numY = length(y0);

        % We need a cell vector form of schemeFunc.
        if(iscell(schemeFunc))
            schemeFuncCell = schemeFunc;
        else
            [ schemeFuncCell{1:numY} ] = deal(schemeFunc);
        end

    else
        % Set numY, but be careful: ((numY == 1) & iscell(y0)) is possible.
        numY = 1;

        % We need a cell vector form of schemeFunc.
        schemeFuncCell = { schemeFunc };

    end

    % - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
    t = tspan(1);
    steps = 0;
    startTime = cputime;
    stepBound = zeros(numY, 1);
    ydot = cell(numY, 1);
    y = y0;

    % - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
    while(tspan(2) - t >= small * abs(tspan(2)))

        % Approximate the derivative and CFL restriction.
        for i = 1 : numY
            [ ydot{i}, stepBound(i), schemeData ] = ...
                feval(schemeFuncCell{i}, t, y, schemeData);

            % If this is a vector level set, rotate the lists of vector
```

```
    % arguments.
    if(iscell(y))
        y = y([ 2:end, 1 ]);
    end

    if(iscell(schemeData))
        schemeData = schemeData([ 2:end, 1 ]);
    end
end

% - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
% Determine CFL bound on timestep, but not beyond the final time.
%   For vector level sets, use the most restrictive stepBound.
deltaT = min([ options.factorCFL * stepBound; ...
    tspan(2) - t; options.maxStep ]);

% If there is a terminal event function registered, we need
%   to maintain the info from the last timestep.
if(~isempty(options.terminalEvent))
    yOld = y;
    tOld = t;
end

% Update time.
t = t + deltaT;

% Update level set functions.
if(iscell(y))
    for i = 1 : numY
        y{i} = y{i} + deltaT * ydot{i};
    end
else
    y = y + deltaT * ydot{1};
end

steps = steps + 1;

% - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
% If there is one or more post-timestep routines, call them.
if(~isempty(options.postTimestep))
    [ y, schemeData ] = odeCFLcallPostTimestep(t, y, schemeData,...
        options);
end

% If we are in single step mode, then do not repeat.
if(strcmp(options.singleStep, 'on'))
    break;
end
```

```
        % If there is a terminal event function, establish initial sign
        %   of terminal event vector.
        if(~isempty(options.terminalEvent))

            [ eventValue, schemeData ] = ...
                feval(options.terminalEvent, t, y, tOld, yOld, schemeData);

            if((steps > 1) && any(sign(eventValue) ~= sign(eventValueOld)))
                break;
            else
                eventValueOld = eventValue;
            end
        end

    end

    endTime = cputime;

    if(strcmp(options.stats, 'on'))
        fprintf('\t%d steps in %g seconds from %g to %g\n', ...
            steps, endTime - startTime, tspan(1), t);
    end

    %---------------------------------------------------------------------
elseif(numT > 2)
    % If we were asked for the solution at multiple timesteps.
    [ t, y, schemeData ] = ...
        odeCFLmultipleSteps(@odeCFL1, schemeFunc, tspan, y0, options,...
        schemeData);

    %---------------------------------------------------------------------
else
    % Malformed time span.
    error('tspan must contain at least two entries');
end



function [ t, LS, Phi_ref ] = ...
    odeCFL1(schemeFunc, tspan, LS, options, schemeData)

%% author: Soheil Ghanbarzadeh
%% author of original function: Ian Mitchell, @ level set toolbox
%% Last version date: 06/28/2016
% modified from odeCFL1.m function by Ian Mitchell in the level set
% toolbox. See the origninal function, toolbox and related paper (Mitchell
% 2008) for more information.

% Description:
```

```
% Integrates a system forward in time by CFL constrained timesteps
% using a first order forward Euler scheme, which happens to be the first
% order TVD RK scheme.


% Input:
% schemeFunc = Function handle to a CFL constrained ODE system. See
% original function by Ian Mitchell for more information.
% tspan = Range of time over which to integrate. See original function by
% Ian Mitchell for more information.
% LS = Level Set data structure which contains initial condition vector
% options = An option structure generated by odeCFLset
% schemeData = A structure. See original function by Ian
% Mitchell for more information.

% Output:
% t = Output time
% LS = Output state of LS.
% Phi_ref = Refined final level set (Cap Phi in JCP paper)

%---------------------------------------------------------------------------

% get some constants, set the counters to zero.

% how often to reinitilize the level set functions.
u = schemeData.u;
% initialize or not?
flag_init=schemeData.reinitflag;

% get the gird structures.
g = schemeData.grid;
gRef = schemeData.gRef;


% Set the counters to zero
k = 0;
kk = 0;

% how many time step to take and calculate the velocity terms. Updating
% velocity terms usually takes a long time, specially for CCH and elongated
% grains. so you might want to update them every 2-3 time step.
n_vterms = 1;

%---------------------------------------------------------------------------
% How close (relative) do we need to be to the final time?
small = 100 * eps;

%---------------------------------------------------------------------------
% Make sure we have the default options settings
```

```
if((nargin < 4) || isempty(options))
    options = odeCFLset;
end

%---------------------------------------------------------------------------
% Number of timesteps to be returned.
numT = length(tspan);

%---------------------------------------------------------------------------
% If we were asked to integrate forward to a final time.
if(numT == 2)

    % - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
    % Is this a vector level set integration?
    schemeFuncCell =  schemeFunc ;
    t = tspan(1);

    % - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
    while(tspan(2) - t >= small * abs(tspan(2)))
        STEPB = zeros(length(LS),1);

        % update the outside index of each elvel set.
        for j = 1 : length(LS)
            LS(j,1).dataOut_Index = (LS(j,1).data > 0);
        end
        % update the Heaviside functions:  smeared Heaviside and delta
        % functions, active grid points for surface diffusion term,
        % curvature, gradient, gradient magnitude and sign functions
        [LS] = heaviside_param(g, LS);

        % refine the grids and calculate the porosity on the new grid
        [Phi_ref, por] = RealtimeRef(g, gRef, LS);

        % compute the mean curvature and vairance in mean curvature near
        % the interface
        [MeanCurvature] = get_meanCurv(g, LS);

        % calculate the relative difference between target and current
        % porosities.
        dPor = (100 * por - schemeData.innerData{2,1}.tagetPor)/...
            schemeData.innerData{2,1}.tagetPor;

        % updater the counter
        kk = kk + 1;

        % uopdate the velocity terms, if needed.
        if kk == 1 || mod(kk,n_vterms)==0
            [g , LS] = setupVelocities(g, LS, MeanCurvature , dPor);
        else
```

```
    deltaVscaled = 5*(dPor) * (exp(abs(dPor)))/(0.05^(g.dim-1));
    for j = 1 : length(LS)
        LS(j,1).Normalspeed = LS(j,1).Normalspeed + ...
            deltaVscaled + MeanCurvature * LS(j,1).Active...
            .* LS(j,1).ActiveCurvatureInsideSai + ...
            (-5*(DV) * (exp(abs(DV)))/(0.05^(g.dim-1))) ...
            - mC * LS(j,1).Active .* ...
            LS(j,1).ActiveCurvatureInsideSai;
    end
end

% use the parameters from previos timestep.
DV = dPor; mC = MeanCurvature;

% Approximate the derivative and CFL restriction.
[ LS, ~ , schemeData ] = ...
    feval(schemeFuncCell, t, LS, schemeData);

% take the time step bounds out of LS data structure
for j = 1 : length(LS)
    STEPB(j,1) = LS(j,1).stepBound;
end

% - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
% Determine CFL bound on timestep, but not beyond the final time.
%   For vector level sets, use the most restrictive stepBound.
deltaT = min([ options.factorCFL * STEPB(:,1); ...
    tspan(2) - t; options.maxStep ]);

% If there is a terminal event function registered, we need
%   to maintain the info from the last timestep.
if(~isempty(options.terminalEvent))
    yOld = y;
    tOld = t;
end

% Update time.
t = t + deltaT;

% Update level set functions.
for j = 1 : length(LS)
    LS(j,1).y = LS(j,1).y + deltaT * LS(j,1).ydot;
end

for j = 1 : length(LS)
    LS(j,1).data = reshape(LS(j,1).y, LS(j,1).gridShape);
end
```

```
        % - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
        % If there is one or more post-timestep routines, call them.
        if(~isempty(options.postTimestep))
            [ y, schemeData ] = odeCFLcallPostTimestep(t, y, schemeData, options);
        end

        % If we are in single step mode, then do not repeat.
        if(strcmp(options.singleStep, 'on'))
            break;
        end

        % If there is a terminal event function, establish initial sign
        %   of terminal event vector.
        if(~isempty(options.terminalEvent))

            [ eventValue, schemeData ] = ...
                feval(options.terminalEvent, t, y, tOld, yOld, schemeData);

            if((steps > 1) && any(sign(eventValue) ~= sign(eventValueOld)))
                break;
            else
                eventValueOld = eventValue;
            end
        end

        fprintf('\t deltaV = %g %%, MeanCurv = %g\n', ...
                    dPor*100, MeanCurvature);
        % Reinitialize? if yes, follow below.
        if flag_init
            % update the counter for how many time steps before
            % reinitializing
            k=k+1;

            % if time to reinitilize, do it then
            % always reinitilize the first time you came into this
            % function.
            if k == 1 || mod(k,u)==0
                s1 = cputime;
                % go to reinit function
                [ LS, g] = reinit_fun(LS, g , 'low' );
                n1 = cputime;
                fprintf('\t Done re-initializing in %g seconds \n',n1-s1);

            end
        end
    end

    %----------------------------------------------------------------------
elseif(numT > 2)
```

```
    % If we were asked for the solution at multiple timesteps.
    [ t, y, schemeData ] = ...
        odeCFLmultipleSteps(@odeCFL1, schemeFunc, tspan, y0, options, schemeData);

    %---------------------------------------------------------------------------
else
    % Malformed time span.
    error('tspan must contain at least two entries');
end



function [ ydot, stepBound, schemeData ] = termConvection(t, LS, schemeData)


%% author: Soheil Ghanbarzadeh
%% author of original function: Ian Mitchell, @ level set toolbox
%% Last version date: 06/28/2016
% modified from termConvection.m function by Ian Mitchell in the level set
% toolbox. See the origninal function, toolbox and related paper (Mitchell
% 2008) for more information.

% Description:
% Computes an approximation of motion by a constant velocity field V(x,t)
% for a Hamilton-Jacobi PDE (often called convective or advective flow).
% The PDE is:
%
%            D_t \phi = -V(x,t) \dot \grad \phi.
%
% Based on methods outlined in O&F, chapter 3.  The more conservative CFL
% condition (3.10) is used.


% Input:
% t = Time at beginning of timestep.
% LS = Level Set data structure.
% schemeData = A structure. See original function by Ian
% Mitchell for more information.


% Output:
% ydot = Change in the data array, in vector form.
% stepBound = CFL bound on timestep for stability.
% schemeData = A structure.

%---------------------------------------------------------------------------
% For vector level sets, ignore all the other elements.
if(iscell(schemeData))
    thisSchemeData = schemeData{1};
```

```
else
    thisSchemeData = schemeData;
end

%-------------------------------------------------------------------------
% check the provided structure fields.
checkStructureFields(thisSchemeData, 'grid', 'derivFunc');

grid = thisSchemeData.grid;

% check if we have multiple level sets in structure data LS.
% cPhi is the current level set function (i) that the calculations are
% being done on.

if isstruct(LS)
    y = LS(grid.cPhi,1).y;
else
    y = LS;
end

%-------------------------------------------------------------------------
if(iscell(y))
    data = reshape(y{1}, grid.shape);
else
    data = reshape(y, LS(grid.cPhi,1).gridShape);
end
% the normal and convective speeds are already calculated.
thisSchemeData.velocity = LS(grid.cPhi,1).ConvectionSpeed;

%-------------------------------------------------------------------------
% Get velocity field.
if(isa(thisSchemeData.velocity, 'cell'))
    velocity = thisSchemeData.velocity;

elseif(isa(thisSchemeData.velocity, 'function_handle'))

    if(iscell(y))

        if(isfield(thisSchemeData, 'passVLS') && thisSchemeData.passVLS)
            % Pass the vector level set information through.
            numY = length(y);
            vectorData = cell(numY, 1);
            for i = 1 : numY
                if(iscell(schemeData))
                    vectorData{i} = reshape(y{i}, schemeData{i}.grid.shape);
                else
                    vectorData{i} = reshape(y{i}, schemeData.grid.shape);
                end
            end
```

```
            velocity = feval(thisSchemeData.velocity, t, vectorData, schemeData);

        else
            % Ignore any vector level set.
            velocity = feval(thisSchemeData.velocity, t, data, thisSchemeData);
        end

    else
        % There is no vector level set.
        velocity = feval(thisSchemeData.velocity, t, LS, thisSchemeData);
    end

else
    error('schemeData.velocity must be a cell vector or a function handle');

end

%-----------------------------------------------------------------------------
% Approximate the convective term dimension by dimension.
delta = zeros(size(data));
stepBoundInv = 0;
for i = 1 : grid.dim

    % Get upwinded derivative approximations.
    [ derivL, derivR ] = feval(thisSchemeData.derivFunc,...
        LS(grid.cPhi,1).grid, data, i);

    % Figure out upwind direction.
    v = velocity{i};
    flowL = (v < 0);
    flowR = (v > 0);

    % Approximate convective term with upwinded derivatives
    % (where v == 0 derivative doesn't matter).
    deriv = derivL .* flowR + derivR .* flowL;

    % Dot product requires sum over dimensions.
    delta = delta + deriv .* v;

    % CFL condition.  Note that this is conservative; we really should do
    % the summation over the entire grid and then take the maximum,
    % rather than maximizing for each dimension and then summing.
    stepBoundInv = stepBoundInv + max(abs(v(:))) / grid.dx(i);
end

%-----------------------------------------------------------------------------
stepBound = 1 / stepBoundInv;

% Reshape output into vector format and negate for RHS of ODE.
```

```
ydot = -delta(:);



function [ ydot, stepBound, schemeData ] = termLapCurvature(t, LS, schemeData)

%% author: Soheil Ghanbarzadeh
%% author of original function: Ian Mitchell, @ level set toolbox
%% Last version date: 06/28/2016
% modified from termCurvature.m function by Ian Mitchell in the level set
% toolbox. See the origninal function, toolbox and related paper (Mitchell
% 2008) for more information.

% Description:
% Computes an approximation of diffusive motion by laplacian of curvature
% for a Hamilton-Jacobi PDE.  This is a fourth order equation:
%
%                 D_t \phi - b(x,t) \nabla \kappa(x) \| \grad \phi \| = 0.
%
% where \kappa(x) is the mean curvature.


% Input:
% t = Time at beginning of timestep.
% LS = Level Set data structure.
% schemeData = A structure. See original function by Ian
% Mitchell for more information.


% Output:
% ydot = Change in the data array, in vector form.
% stepBound = CFL bound on timestep for stability.
% schemeData = A structure.


%---------------------------------------------------------------------------
if(iscell(schemeData))
    thisSchemeData = schemeData{1};
else
    thisSchemeData = schemeData;
end

checkStructureFields(thisSchemeData, 'grid', 'b', 'curvatureFunc');

grid = thisSchemeData.grid;

% check if we have multiple level sets in structure data LS.
% cPhi is the current level set function (i) that the calculations are
% being done on.
```

```
if isstruct(LS)
    y = LS(grid.cPhi,1).y;
else
    y = LS;
end
%--------------------------------------------------------------------------
if(iscell(y))
    data = reshape(y{1}, grid.shape);
else
    data = reshape(y, LS(grid.cPhi,1).gridShape);
end


%--------------------------------------------------------------------------
% Get multiplier
if(isa(thisSchemeData.b, 'double'))
    b = thisSchemeData.b;


elseif(isa(thisSchemeData.b, 'function_handle'))

    if(iscell(y))
        % If there is a vector level set.

        if(isfield(thisSchemeData, 'passVLS') && thisSchemeData.passVLS)
            % Pass the vector level set information through.
            numY = length(y);
            dataV = cell(numY, 1);
            for i = 1 : numY
                if(iscell(schemeData))
                    dataV{i} = reshape(y{i}, schemeData{i}.grid.shape);
                else
                    dataV{i} = reshape(y{i}, schemeData.grid.shape);
                end
            end
            b = feval(thisSchemeData.b, t, dataV, schemeData);

        else
            % Ignore any vector level set.
            b = feval(thisSchemeData.b, t, data, thisSchemeData);
        end

    else
        % There is no vector level set.
        b = feval(thisSchemeData.b, t, data, thisSchemeData);
    end
```

```
else
    error('schemeData.b must be a scalar, array or function handle');

end

%--------------------------------------------------------------------------
% According to O&F equation (4.5).

% calculate curvature, Laplacina of curvature in places where curvature
% term should be active (only near the interface).
% if not considering Active area, it would not converge.
Active = LS(grid.cPhi,1).Active;
curvature = LS(grid.cPhi,1).curvature;
curvature = -1 * laplacianSecond(LS(grid.cPhi,1).grid, curvature) .* Active;

% calculate gradient for current level set function
gradMag = LS(grid.cPhi,1).gradMag;
% calculate the change in value
delta =  - b .*  curvature .* gradMag .*...
    LS(grid.cPhi,1).ActiveCurvatureInsideSai;
%--------------------------------------------------------------------------
% According to O&F equation (4.7).
if grid.dim == 2
    stepBound = 1 / (20 * max(b(:)) * sum(grid.dx .^ -4));
else
    stepBound = 1 / (20 * max(b(:)) * sum(grid.dx .^ -4));
end


if  numel(stepBound)==0
    stepBound = inf;
end

% Reshape output into vector format and negate for RHS of ODE.
ydot = -delta(:);



function [ ydot, stepBound, schemeData ] = termNormal(t, LS , schemeData)

%% author: Soheil Ghanbarzadeh
%% author of original function: Ian Mitchell, @ level set toolbox
%% Last version date: 06/28/2016
% modified from termNormal.m function by Ian Mitchell in the level set
% toolbox. See the origninal function, toolbox and related paper (Mitchell
% 2008) for more information.

% Description:
% Computes an approximation of motion of the interface at speed a(x,t) in
```

```
% the normal direction.  The PDE is:
%
%             D_t \phi + a(x,t) \| \grad \phi \| = 0.



% Input:
% t = Time at beginning of timestep.
% LS = Level Set data structure.
% schemeData = A structure. See original function by Ian
% Mitchell for more information.


% Output:
% ydot = Change in the data array, in vector form.
% stepBound = CFL bound on timestep for stability.
% schemeData = A structure.

%---------------------------------------------------------------------------
if(iscell(schemeData))
    thisSchemeData = schemeData{1};
else
    thisSchemeData = schemeData;
end

checkStructureFields(thisSchemeData, 'grid', 'derivFunc');

grid = thisSchemeData.grid;

% check if we have multiple level sets in structure data LS.
% cPhi is the current level set function (i) that the calculations are
% being done on.
if isstruct(LS)
    y = LS(grid.cPhi,1).y;
else
    y = LS;
end

% the normal and convective speeds are already calculated.
thisSchemeData.speed = LS(grid.cPhi,1).Normalspeed;
%---------------------------------------------------------------------------
% For most cases, we are interested in the first implicit surface function.
if(iscell(y))
    data = reshape(y{1}, grid.shape);
else
    data = reshape(y, LS(grid.cPhi,1).gridShape);
end

%---------------------------------------------------------------------------
```

```
% Get speed field.
if(isa(thisSchemeData.speed, 'double'))
    speed = thisSchemeData.speed;

elseif(isa(thisSchemeData.speed, 'function_handle'))

    if(iscell(y))
        % If there is a vector level set.

        if(isfield(thisSchemeData, 'passVLS') && thisSchemeData.passVLS)
            % Pass the vector level set information through.
            numY = length(y);
            dataV = cell(numY, 1);
            for i = 1 : numY
                if(iscell(schemeData))
                    dataV{i} = reshape(y{i}, schemeData{i}.grid.shape);
                else
                    dataV{i} = reshape(y{i}, schemeData.grid.shape);
                end
            end
            speed = feval(thisSchemeData.speed, t, dataV, schemeData);

        else
            % Ignore any vector level set.
            speed = feval(thisSchemeData.speed, t, data, thisSchemeData);
        end

    else
        % There is no vector level set.
        speed = feval(thisSchemeData.speed, t, LS, thisSchemeData);
    end

else
    error('schemeData.speed must be a scalar, array or function handle');
end

%-----------------------------------------------------------------------------
% In the end, all we care about is the magnitude of the gradient.
magnitude = zeros(size(data));

% In this case, keep track of stepBound for each node until the very
%   end (since we need to divide by the appropriate gradient magnitude).
stepBoundInv = zeros(size(data));

% Determine the upwind direction dimension by dimension
for i = 1 : grid.dim

    % Get upwinded derivative approximations.
    [ derivL, derivR ] = feval(thisSchemeData.derivFunc,...
```

```
        LS(grid.cPhi,1).grid, data, i);

    % Effective velocity in this dimension (scaled by \|\grad \phi\|).
    prodL = speed .* derivL;
    prodR = speed .* derivR;
    magL = abs(prodL);
    magR = abs(prodR);

    % Determine the upwind direction.
    %   Either both sides agree in sign (take direction in which they agree),
    %   or characteristics are converging (take larger magnitude direction).
    flowL = ((prodL >= 0) & (prodR >= 0)) | ...
        ((prodL >= 0) & (prodR <= 0) & (magL >= magR));
    flowR = ((prodL <= 0) & (prodR <= 0)) | ...
        ((prodL >= 0) & (prodR <= 0) & (magL < magR));

    % For diverging characteristics, take gradient = 0
    %   (so we don't actually need to calculate this term).
    %flow0 = ((prodL <= 0) & (prodR >= 0));

    % Now we know the upwind direction, add its contribution to
    % \|\grad \phi\|.
    magnitude = magnitude + derivL.^2 .* flowL + derivR.^2 .* flowR;

    % CFL condition: sum of effective velocities from O&F (6.2).
    effectiveVelocity = magL .* flowL + magR .* flowR;
    dxInv = 1 / grid.dx(i);
    stepBoundInv = stepBoundInv + dxInv * effectiveVelocity;
end

%-------------------------------------------------------------------------
% Finally, calculate speed * \|\grad \phi\|
magnitude = sqrt(magnitude);


delta = speed .* magnitude;

% Find the most restrictive timestep bound.
nonZero = find(magnitude > 0);
stepBoundInvNonZero = stepBoundInv(nonZero) ./ magnitude(nonZero);
stepBound = 1 / max(stepBoundInvNonZero(:));

if  numel(stepBound)==0
    stepBound = inf;
end

% Reshape output into vector format and negate for RHS of ODE.
ydot = -delta(:);
```

```
function [ ydot, stepBound, schemeData ] = termReinit(t, LS, schemeData)

%% author: Soheil Ghanbarzadeh
%% author of original function: Ian Mitchell, @ level set toolbox
%% Last version date: 06/28/2016
% modified from termReinit.m function by Ian Mitchell in the level set
% toolbox. See the origninal function, toolbox and related paper (Mitchell
% 2008) for more information.

% Description:
% Computes a Godunov approximation to motion by the reinitialization
% equation.  While the reinitialization equation is a general nonlinear HJ
% PDE, such a Godunov approximation is the least dissipative monotone
% approximation (less dissipative than Roe-Fix or Lax-Friedrichs).  The
% reinitialization equation is
%
%            D_t \phi = -sign(\phi_0)(\|\grad \phi\| - 1).
%
% where phi_0 is the initial conditions.  Solving the reinitialization
% equation turns an implicit surface function into a signed distance
% function.  It is iterative, and often slower than a fast marching method;
% however, it can use high order approximations and can start directly from
% the implicit surface function without needing to explicitly find the
% implicit surface (although the subcell fix discussed below does in some
% sense find the surface).


% Input:
% t = Time at beginning of timestep.
% LS = Level Set data structure.
% schemeData = A structure. See original function by Ian
% Mitchell for more information.


% Output:
% ydot = Change in the data array, in vector form.
% stepBound = CFL bound on timestep for stability.
% schemeData = A structure.


%--------------------------------------------------------------------------
% The subcell fix has some options.

% Use the robust signed distance function (17) or the simple one (13)?
% The simple one often fails due to divide by zero errors, so be careful.
robust_subcell = 1;

% Small positive parameter that appears in the robust version.  In
% fact, we will use this as a relative value with respect to grid.dx
```

```
robust_small_epsilon = 1e6 * eps;

%-------------------------------------------------------------------------
% For vector level sets, ignore all the other elements.
if iscell(schemeData)
    thisSchemeData = schemeData{1};
else
    thisSchemeData = schemeData;
end

% Check for required fields.
checkStructureFields(thisSchemeData, 'grid', 'derivFunc');

grid = thisSchemeData.grid;

% check if we have multiple level sets in structure data LS.
% cPhi is the current level set function (i) that the calculations are
% being done on.
if isstruct(LS)
    y = LS(grid.cPhi,1).y;
else
    y = LS;
end



%-------------------------------------------------------------------------
if iscell(y)
    data = reshape(y{1}, grid.shape);
else
    data = reshape(y, grid.shape);
end

%-------------------------------------------------------------------------
if isfield(thisSchemeData, 'subcell_fix_order')
    switch(thisSchemeData.subcell_fix_order)
        case 0
            apply_subcell_fix = 0;

        case 1
            apply_subcell_fix = 1;
            subcell_fix_order = 1;

        otherwise
            error('Reinit subcell fix order of accuracy %d not supported', ...
                thisSchemeData.subcell_fix_order);
    end
else
    % Default behavior is to apply the simplest subcell fix.
```

```
    apply_subcell_fix = 1;
    subcell_fix_order = 1;
end

%---------------------------------------------------------------------------
if apply_subcell_fix
    % The sign function is only used far from the interface, so we do
    % not need to smooth it.
    S = sign(thisSchemeData.initial);
else
    % Smearing factor for the smooth approximation of the sign function.
    dx = max(grid.dx);
    sgnFactor = dx.^2;

    % Sign function (smeared) identifies on which side of surface each node
    % lies.
    S = smearedSign(grid, thisSchemeData.initial, sgnFactor);

end

%---------------------------------------------------------------------------
% Compute Godunov derivative approximation for each dimension.  This
% code is used for the PDE far from the interface, or for all nodes if
% the subcell fix is not applied.
deriv = cell(grid.dim, 1);
%flow = cell(grid.dim, 1);
for i = 1 : grid.dim
    [ derivL, derivR ] = feval(thisSchemeData.derivFunc, grid, data, i);

    % For Gudunov's method, check characteristic directions
    %   according to left and right derivative approximations.

    % Both directions agree that flow is to the left.
    flowL = ((S .* derivR <= 0) & (S .* derivL <= 0));

    % Both directions agree that flow is to the right.
    flowR = ((S .* derivR >= 0) & (S .* derivL >= 0));

    % Diverging flow; entropy condition requires choosing deriv = 0
    %   (so we don't actually have to calculate this term).
    %flow0 = ((S .* derivR >  0) & (S .* derivL <  0));

    % Converging flow, need to check which direction arrives first.
    flows = ((S .* derivR <  0) & (S .* derivL >  0));
    if(any(flows(:)))
        conv = find(flows);
        s = zeros(size(flows));
        s(conv) = S(conv) .* (abs(derivR(conv)) - abs(derivL(conv))) ...
            ./ (derivR(conv) - derivL(conv));
```

```
        % If s == 0, both directions arrive at the same time.
        %   Assuming continuity, both will produce same result, so pick one.
        flowL(conv) = flowL(conv) | (s(conv) < 0);
        flowR(conv) = flowR(conv) | (s(conv) >= 0);
    end

    deriv{i} = derivL .* flowR + derivR .* flowL;
    %flow{i} = flowR - flowL;
end

%-----------------------------------------------------------------------------
% Compute magnitude of gradient.
mag = zeros(size(grid.xs{1}));
for i = 1 : grid.dim;
    mag = mag + deriv{i}.^2;
end
mag = max(sqrt(mag), eps);

%-----------------------------------------------------------------------------
% Start with constant term in the reinitialization equation.
delta = -S;

% Compute change in function and bound on step size.
stepBoundInv = 0;
for i = 1 : grid.dim

    % Effective velocity field (for timestep bounding).
    v = S .* deriv{i} ./ mag;

    % Update just like a velocity field.
    delta = delta + v .* deriv{i};

    % CFL condition using effective velocity.
    stepBoundInv = stepBoundInv + max(abs(v(:))) / grid.dx(i);

end

%-----------------------------------------------------------------------------
if apply_subcell_fix

    switch(subcell_fix_order)
        case 1
            % Most of the effort below -- specifically computation of the
            % distance to the interface D -- depends only on
            % thisSchemeData.initial, so recomputation could be avoided if
            % there were some easy way to
            % memoize the results between timesteps.  It could be done by
            % modifying schemeData, but that has a rather high overhead and
```

```
% could lead to bugs if the user fiddles with schemeData.
% So for now, we recompute at each timestep.

% Set up some index cell vectors.  No ghost cells will be used,
% since nodes near the edge of the computational domain should
% not be near the interface.  Where necessary, we will modify
% the stencil near the edge of the domain.
indexL = cell(grid.dim, 1);
for d = 1 : grid.dim
    indexL{d} = 1 : grid.N(d);
end
indexR = indexL;

% Compute denominator in (13) or (16) or (23).  Note that we
% have moved the delta x term into this denominator to treat
% the case when delta x is not the same in each dimension.
denom = zeros(size(data));
for d = 1 : grid.dim

    dx_inv = 1 ./ grid.dx(d);

    % Long difference used in (13) and (23).  For the nodes
    % near the edge of the computational domain, we will just
    % use short differences.
    indexL{d} = [ 1, 1 : grid.N(d) - 1 ];
    indexR{d} = [ 2 : grid.N(d), grid.N(d) ];
    diff2 = (0.5 * dx_inv * ...
        (thisSchemeData.initial(indexR{:}) ...
        - thisSchemeData.initial(indexL{:}))) .^ 2;

    if robust_subcell
        % Need the short differences.
        indexL{d} = 1 : grid.N(d) - 1;
        indexR{d} = 2 : grid.N(d);
        short_diff2 = (dx_inv * ...
            (thisSchemeData.initial(indexR{:}) ...
            - thisSchemeData.initial(indexL{:}))) .^ 2;

        % All the various terms of (17).
        diff2(indexL{:}) = max(diff2(indexL{:}), short_diff2);
        diff2(indexR{:}) = max(diff2(indexR{:}), short_diff2);
        diff2 = max(diff2, robust_small_epsilon .^ 2);
    end

    % Include this dimension's contribution to the distance.
    denom = denom + diff2;

    % Reset the index vectors.
    indexL{d} = 1 : grid.N(d);
```

```
                indexR{d} = 1 : grid.N(d);

            end
            denom = sqrt(denom);

            % Complete (13) or (16) or (23).  Note that delta x was already
            % included in the denominator calculation above, so it does not
            % appear.
            D = thisSchemeData.initial ./ denom;

            % We do need to know which nodes are near the interface.
            near = isNearInterface(thisSchemeData.initial);

            % Adjust the update.  The delta x that appears in (15) or (22)
            % comes from the smoothing in (14), so we choose the maximum
            % delta x in this case (guarantees sufficient smoothing no
            % matter what the direction of the interface).  For grids with
            % different delta x, this choice may require more
            % reinitialization steps to achieve desired results.
            delta = (delta .* (~near) ...
                + (S .* abs(data) - D) / max(grid.dx) .* near);

            % We will not adjust the CFL step bound.  By Russo & Smereka,
            % the adjusted update has a bound of 1, and the regular scheme
            % above should already have that same upper bound.

        otherwise
            error('Reinit subcell fix order of accuracy %d not supported', ...
                subcell_fix_order);
    end
end

%-----------------------------------------------------------------------------
stepBound = 1 / stepBoundInv;

% Reshape output into vector format and negate for RHS of ODE.
ydot = -delta(:);



%-----------------------------------------------------------------------------
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%-----------------------------------------------------------------------------
function s = smearedSign(grid, data, sgnFactor)
% s = smearedSign(grid, data)
%
% Helper function to generated a smeared signum function.
%
% This version (with sgnFactor = dx.^2) is (7.5) in O&F chapter 7.4.
```

```
s = data ./ sqrt(data.^2 + sgnFactor);



function [ LS, stepBound, schemeData ] = termSum(t, LS, schemeData)

%% author: Soheil Ghanbarzadeh
%% author of original function: Ian Mitchell, @ level set toolbox
%% Last version date: 06/28/2016
% modified from termSum.m function by Ian Mitchell in the level set toolbox.
% See the origninal function, toolbox and related paper (Mitchell
% 2008) for more information.

% Description:
% This function independently evaluates a collection of HJ term
% approximations and returns their elementwise sum.


            % Input:
            % t = Time at beginning of timestep.
            % LS = Level Set data structure.
            % schemeData = A structure. See original function by Ian
            % Mitchell for more information.


            % Output:
            % LS = Updated data strucure. Affacted fields are 'ydot',
            % 'stepBound' and 'stepB'.
            % stepBound = CFL bound on timestep for stability.
            % schemeData = A structure.

%-----------------------------------------------------------------------------
% For vector level sets, get the first element.
if(iscell(schemeData))
    thisSchemeData = schemeData{1};
else
    thisSchemeData = schemeData;
end
checkStructureFields(thisSchemeData, 'innerFunc', 'innerData');

%-----------------------------------------------------------------------------
% Check that innerFunc and innerData are the same size cell vectors.
if(~iscell(thisSchemeData.innerFunc) || ~iscell(thisSchemeData.innerData))
    error('schemeData.innerFunc and schemeData.innerData %s', ...
        'must be cell vectors');
end

numSchemes = length(thisSchemeData.innerFunc(:));
```

```
if(numSchemes ~= length(thisSchemeData.innerData(:)))
    error('schemeData.innerFunc and schemeData.innerData must be %s', ...
        'the same length');
end

%--------------------------------------------------------------------------
% Calculate sum of updates (inverse sum of stepBounds).


for j = 1 : length(LS)
    stepBoundInv = 0;
    ydot = 0;
    stepB = zeros(numSchemes,1);
    for i = 1 : numSchemes

        % Extract the appropriate inner data structure.
        if(iscell(schemeData))
            innerData = schemeData;
            innerData{1} = schemeData{1}.innerData{i};
        else
            innerData = schemeData.innerData{i};
        end

        innerData.grid.cPhi = j;

        % Compute this component of the update.
        [ updateI, stepBoundI, ~ ] = ...
            feval(thisSchemeData.innerFunc{i}, t, LS , innerData);
        ydot = ydot + updateI;

        stepB(i) = stepBoundI;
        stepBoundInv = stepBoundInv + 1 / stepBoundI(1);
        % Store any modifications of the inner data structure.
        if(iscell(schemeData))
            schemeData{1}.innerData{i} = innerData{1};
        else
            schemeData.innerData{i} = innerData;
        end
    end

    %--------------------------------------------------------------------------
    % Final timestep bound.
    if(stepBoundInv == 0)
        stepBound = inf;
    else
        stepBound = 1 / stepBoundInv;
    end
```

```
    LS(j,1).stepB = stepB;
    LS(j,1).ydot = ydot;
    LS(j,1).stepBound = stepBound;
end



function h = visualizeLevelSet(g, data, display_type, level, ...
    title_string, LSColormap)

%% author: Soheil Ghanbarzadeh
%% author of original function: Ian Mitchell, @ level set toolbox
%% Last version date: 06/28/2016
% modified from visualizeLevelSet.m function by Ian Mitchell in the level
% set toolbox. See the origninal function, toolbox and related paper
% (Mitchell 2008) for more information.
% The main difference in here is that a color can be called in input for
% the interface which is being displayed. Unnecessary lighting and views
% are also removed.

% Description:
% Displays a variety of level set visualizations in dimensions 1 to 3.
% The current figure and axis is used.
% See original function by Ian Mitchell for more information.


% Input:
% g = Grid structure.
% data = Array storing the implicit surface function.
% display_type = String specifying the type of visualization. See original
% function by Ian Mitchell for more information.
% level = Double.  Which isosurface to display.  Defaults to 0.
% title_string = Optional string to place in the figure title.
% LSColormap = color chosen by user for display of 2d and 3d level sets

% Output:
% h = Handle to the graphics object created.

%---------------------------------------------------------------------------
if(nargin < 4)
    level = 0;
end

if((strcmp(display_type, 'contour') || strcmp(display_type,...
        'contourslice')) && (numel(level) == 1))
    % Scalar input to contour plot should be repeated.
    level = [ level level ];
end
```

```
if(~isempty(level))
    if((all(data(:) < min(level(:)))) || (all(data(:) > max(level(:)))))
        warning('No implicitly defined surface exists'); %#ok<WNTAG>
    end
end

%-------------------------------------------------------------------------
switch(g.dim)

    %-------------------------------------------------------------------------
    case 1
        switch(display_type)
            case 'plot'
                if(g.N < 20)
                    % For very coarse grids, we can identify the individual
                    % nodes.
                    h = plot(g.xs{1}, data, 'b-+');
                else
                    h = plot(g.xs{1}, data, 'b-');
                end
            otherwise
                error(['Unknown display type %s for %d dimensional'...
                    'system'], display_type, g.dim);
        end

        %-------------------------------------------------------------------
    case 2
        % In 2D, the visualization routines seem to be happy to use ndgrid.
        switch(display_type)
            case 'contour'
                [ ~, h ] = contour(g.xs{1}, g.xs{2}, data, level,...
                    'LineColor', LSColormap,'LineWidth',2);
                axis square;  axis manual;
            case 'surf'
                h = surf(g.xs{1}, g.xs{2}, data , 'linestyle' , 'none');
            otherwise
                error(['Unknown display type %s for %d dimensional'...
                    'system'], display_type, g.dim);
        end

        %-------------------------------------------------------------------
    case 3
        % Stupid Matlab's stupid meshgrid vs ndgrid incompatibility really
        % shows up in 3D -- many of the 3D visualization routines only work
        % for meshgrid produced grids. Therefore, we need to massage the
        % grid and data to make it work.
        [ mesh_xs, mesh_data ] = gridnd2mesh(g, data);

        switch(display_type)
```

```
            case 'surface'
                h = patch(isosurface(mesh_xs{:}, mesh_data, level));
                isonormals(mesh_xs{:}, mesh_data, h);
                set(h, 'FaceColor', LSColormap, 'EdgeColor', 'none' ,...
                    'linestyle' , 'none');

            case 'slice'
                % For lack of a better idea of where to put the slices,
                % we'll just slice through the middle of the grid.  For
                % final visualizations, users should write their own code
                % with a better choice of slice planes.
                avgx = mean(g.vs{1});
                avgy = mean(g.vs{2});
                avgz = mean(g.vs{3});
                h = slice(mesh_xs{:}, mesh_data, avgx, avgy, avgz);

            case 'contourslice'
                avgx = mean(g.vs{1});
                avgy = mean(g.vs{2});
                avgz = mean(g.vs{3});
                h = contourslice(mesh_xs{:}, mesh_data, [ avgx ],...
                    [ avgy ], [ avgz ], level); %#ok<NBRAK>

                %set(h, 'EdgeColor', 'black');

            case 'wireframe'
                % Because isosurface works on ndgrid, we don't need to use
                % the converted grid and data.
                h = patch(isosurface(g.xs{:}, data, level));
                set(h, 'FaceColor', 'none', 'EdgeColor', 'black');
                view(3)

            otherwise
                error(['Unknown display type %s for %d dimensional'...
                    'system'], display_type, g.dim);
        end

        %-----------------------------------------------------------------
    otherwise
        warning('Unable to display data in dimension %d', g.dim);

end

%-------------------------------------------------------------------------
if(nargin >= 5)
    title(title_string);
end
drawnow;
```

```
function dataOut = addGhostAllDims(grid, dataIn, width)
% addGhostAllDims: Create ghost cells along all grid boundaries.
%
% dataOut = addGhostAllDims(grid, dataIn, width)
%
% Creates ghost cells to manage the boundary conditions for the array dataIn.
%
% This function adds the same number of ghost cells in every dimension
% according to the boundary conditions specified in the grid.
%
% Notice that the indexing is shifted by the ghost cell width in output array.
% So in 2D, the first data in the original array will be at
% dataOut(width+1,width+1) == dataIn(1,1)
%
% Parameters:
% grid  Grid structure (see processGrid.m for details).
% dataIn    Input data array.
% width Number of ghost cells to add on each side (default = 1).
%
% dataOut   Output data array.
%
%
% Copyright 2004 Ian M. Mitchell (mitchell@cs.ubc.ca).
% This software is used, copied and distributed under the licensing
% agreement contained in the file LICENSE in the top directory of
% the distribution.
%
% Ian Mitchell, 6/3/03

dataOut = dataIn;

% add ghost cells
for i = 1 : grid.dim
    dataOut = feval(grid.bdry{i}, dataOut, i, width, grid.bdryData{i});
end



function dataOut = addGhostExtrapolate(dataIn, dim, width, ghostData)
% addGhostExtrapolate: add ghost cells, values extrapolated from bdry nodes.
%
% dataOut = addGhostExtrapolate(dataIn, dim, width, ghostData)
%
% Creates ghost cells to manage the boundary conditions for the array dataIn.
%
% This m-file fills the ghost cells with data linearly extrapolated
% from the grid edge, where the sign of the slope is chosen to make sure the
% extrapolation goes away from or towards the zero level set.
%
```

```
% For implicit surfaces, the extrapolation will typically be away from zero
% (the extrapolation should not imply the presence of an implicit surface
% beyond the array bounds).
%
% Notice that the indexing is shifted by the ghost cell width in output array.
% So in 2D with dim == 1, the first data in the original array will be at
% dataOut(width+1,1) == dataIn(1,1)
%
% parameters:
% dataIn    Input data array.
% dim       Dimension in which to add ghost cells.
% width Number of ghost cells to add on each side (default = 1).
% ghostData A structure (see below).
%
% dataOut   Output data array.
%
% ghostData is a structure containing data specific to this type of
% ghost cell.  For this function it contains the field(s)
%
% .towardZero Boolean indicating whether sign of extrapolation should
% be towards or away from the zero level set (default = 0).
%
%
% Copyright 2004 Ian M. Mitchell (mitchell@cs.ubc.ca).
% This software is used, copied and distributed under the licensing
% agreement contained in the file LICENSE in the top directory of
% the distribution.
%
% Ian Mitchell, 5/12/03
% modified to allow choice of dimension, Ian Mitchell, 5/27/03
% modified to allow ghostData input structure & renamed, Ian Mitchell, 1/13/04

if(nargin < 3)
    width = 1;
end

if((width < 0) || (width > size(dataIn, dim)))
    error('Illegal width parameter');
end

if((nargin == 4) && isstruct(ghostData))
    if(ghostData.towardZero)
        slopeMultiplier = -1;
    else
        slopeMultiplier = +1;
    end
else
    slopeMultiplier = +1;
end
```

```
% create cell array with array size
dims = ndims(dataIn);
sizeIn = size(dataIn);
indicesOut = cell(dims, 1);
for i = 1 : dims
    indicesOut{i} = 1:sizeIn(i);
end
indicesIn = indicesOut;

% create appropriately sized output array
sizeOut = sizeIn;
sizeOut(dim) = sizeOut(dim) + 2 * width;
dataOut = zeros(sizeOut);

% fill output array with input data
indicesOut{dim} = width + 1 : sizeOut(dim) - width;
dataOut(indicesOut{:}) = dataIn;

% compute slopes
indicesOut{dim} = 1;
indicesIn{dim} = 2;
slopeBot = dataIn(indicesOut{:}) - dataIn(indicesIn{:});

indicesOut{dim} = sizeIn(dim);
indicesIn{dim} = sizeIn(dim) - 1;
slopeTop = dataIn(indicesOut{:}) - dataIn(indicesIn{:});

% adjust slope sign to correspond with sign of data at array edge
indicesIn{dim} = 1;
slopeBot = slopeMultiplier * abs(slopeBot) .* sign(dataIn(indicesIn{:}));
indicesIn{dim} = sizeIn(dim);
slopeTop = slopeMultiplier * abs(slopeTop) .* sign(dataIn(indicesIn{:}));

% now extrapolate
for i = 1 : width
    indicesOut{dim} = i;
    indicesIn{dim} = 1;
    dataOut(indicesOut{:}) = (dataIn(indicesIn{:}) + ...
        (width - i + 1) * slopeBot);

    indicesOut{dim} = sizeOut(dim) - i + 1;
    indicesIn{dim} = sizeIn(dim);
    dataOut(indicesOut{:}) = (dataIn(indicesIn{:}) + ...
        (width - i + 1) * slopeTop);
end
```

```
function checkStructureFields(structure, varargin)
% checkStructureFields: check that a structure contains certain fields
%
% checkStructureFields(structure, 'field1', 'field2', ...)
%
% Generates an error if:
% 1) Structure input is not actually a structure.
% 2) Any of the field names is not present in the structure.
%
% Parameters:
% structure    The structure in which to check for fields.
% 'field*'     Strings specifying the field names that the structure
% should contain.
%
%
% Copyright 2004 Ian M. Mitchell (mitchell@cs.ubc.ca).
% This software is used, copied and distributed under the licensing
% agreement contained in the file LICENSE in the top directory of
% the distribution.
%
% Ian Mitchell, 2/11/04

if(isstruct(structure))
    for i = 1 : nargin - 1
        if(~isfield(structure, varargin{i}))
            error('Missing field %s in structure %s',...
                varargin{i}, inputname(1));
        end
    end
else
    error('%s is not a structure', inputname(1))
end



function [ curvature, gradMag ] = curvatureSecond(grid, data)
% curvatureSecond: second order centered difference approx of the curvature.
%
%   [ curvature, gradMag ] = curvatureSecond(grid, data)
%
% Computes a second order centered difference approximation to the curvature.
%
%       \kappa = divergence(\grad \phi / | \grad \phi |)
%
% See O&F section 1.4 for more details.  In particular, this routine
%   implements equation 1.8 for calculating \kappa.
%
% parameters:
%   grid    Grid structure (see processGrid.m for details).
```

```
%   data        Data array.
%
%   curvature   Curvature approximation (same size as data).
%   gradMag Magnitude of gradient |\grad \phi|
%                 Incidentally calculated while finding curvature,
%                 also second order centered difference.

% Copyright 2004 Ian M. Mitchell (mitchell@cs.ubc.ca).
% This software is used, copied and distributed under the licensing
%   agreement contained in the file LICENSE in the top directory of
%   the distribution.
%
% Ian Mitchell, 6/3/03

%---------------------------------------------------------------------------
% Get the first and second derivative terms.
[ second, first ] = hessianSecond(grid, data);

%---------------------------------------------------------------------------
% Compute gradient magnitude.
gradMag2 = first{1}.^2;
for i = 2 : grid.dim
  gradMag2 = gradMag2 + first{i}.^2;
end
gradMag = sqrt(gradMag2);

%---------------------------------------------------------------------------
curvature = zeros(size(data));
for i = 1 : grid.dim;
  curvature = curvature + second{i,i} .* (gradMag2 - first{i}.^2);
  for j = 1 : i - 1
    curvature = curvature - 2 * first{i} .* first{j} .* second{i,j};
  end
end

% Be careful not to stir the wrath of "Divide by Zero".
%  Note that gradMag == 0 implies curvature == 0 already, since all the
%  terms in the curvature approximation involve at least one first dervative.
nonzero = find(gradMag > 0);
curvature(nonzero) = curvature(nonzero) ./ gradMag(nonzero).^3;



function [ mesh_xs, varargout ] = gridnd2mesh(grid, varargin)
% gridnd2mesh: converts an ndgrid to a meshgrid, as well as associated data
%
%  [ mesh_xs, mesh_data1, mesh_data2 ... ] = ...
%                                  gridnd2mesh(grid, nd_data1, nd_data2, ...)
%
```

```
% The grid.xs member (which specifies the location of each node in the grid)
% is generated in ToolboxLS by processGrid using a call to ndgrid.  Such
% grids are incompatible with those generated by calls to meshgrid.  This
% routine converts from an ndgrid to a meshgrid.  The output of this routine
% is useful for 3D visualization calls such as slice, contourslice and
% isonormals, as well as interp2 and interp3.  Note however that all of the
% 2D visualization routines (eg contour, surf, mesh, ...), the 3D
% visualization routine isosurface, and the general dimensional interpn work
% just fine with grids generated by ndgrid, so they should be perferred to
% the conversion performed by this routine where possible.
%
% Input Parameters:
%
%   grid: A standard Toolbox grid structure.  It is the grid.xs member of
%   this structure which is converted into the mesh_xs output.
%
%   nd_data: Zero or more arrays of size grid.shape.  The same permutation
%   is performed on this arrays as is performed on the arrays defining the
%   node locations in the grid.xs cell vector.  Optional.
%
% Output Parameters:
%
%   mesh_xs: A cell vector whose elements would be the output of a call
%   to meshgrid for the set of nodes in the grid structure.
%
%   mesh_data: One array for each input array nd_data, each containing the
%   corresponding permuted data.

% Further comments: Matlab has two methods for generating the node locations
% in Cartesian grids: meshgrid and ndgrid.  Because Matlab's indexing is
% based on indexing into an array (rows/vertical, columns/horizontal) it
% does not agree with the traditional method of indexing into plots
% (x/horizontal, y/vertical).  Many Matlab visualization routines therefore
% assume that they need to swap the first two dimensions, and meshgrid
% builds a grid based on this assumption.  In constrast, ndgrid builds a
% grid without this implicit swap.
%
% This swapping procedure easily leads to inconsistencies when you are
% working with data external to Matlab, especially with higher dimensional
% data.  Therefore ToolboxLS chooses to use ndgrid exclusively.  So far, the
% only problem caused by this choice appears to be the fact that several 3D
% visualization routines require an input grid generated by meshgrid.  Users
% of the Toolbox may also have created their own routines which make this
% implicit dimension swap.
%
% Consequently, this routine is provided to perform the correct permutations
% to transform a grid from ndgrid (and its associated data) into a grid
% equivalent to that produced by meshgrid (and the same transform on the
% data).  Note that this permutation is a relatively expensive process, so
```

```
% it should not be invoked inside inner loops.

% Copyright 2007 Ian M. Mitchell (mitchell@cs.ubc.ca).
% This software is used, copied and distributed under the licensing
%   agreement contained in the file LICENSE in the top directory of
%   the distribution.
%
% Ian Mitchell, 5/17/07

% No need to do anything to a 1D grid.
if(grid.dim == 1)
    mesh_xs = grid.xs{1};
    varargout = varargin;
    return;
end

% Permutation to convert ndgrid to meshgrid.
perm = [ 2, 1, 3 : grid.dim ];

% Permute the node location cell vector.
mesh_xs = cell(grid.dim, 1);
for d = 1 : grid.dim
    mesh_xs{d} = permute(grid.xs{d}, perm);
end

% No point in permuting more input arguments than there are output
% arguments.
n_data_out = min(nargin - 1, nargout - 1);
varargout = cell(n_data_out, 1);

% Permute the data arrays.
for i = 1 : n_data_out
    varargout{i} = permute(varargin{i}, perm);
end



function [ second, first ] = hessianSecond(grid, data)
% hessianSecond: second order centered difference approx of the Hessian.
%
%   [ second, first ] = hessianSecond(grid, data)
%
% Computes a second order centered difference approximation to the Hessian
%   (the second order mixed spatial derivative of the data).
%
% parameters:
%   grid    Grid structure (see processGrid.m for details).
%   data        Data array.
%
```

```
%   second      2D cell array containing centered approx to Hessian's terms.
%                 To save space, only lower left half of Hessian is given
%                 (since mixed partials are derivative order independent).
%                     second{i,j} = d^2 data / dx_i dx_j   if j < i
%                                 = d^2 data / dx_i^2      if j = i
%                                 = []                     if j > i
%   first       1D cell array containing centered approx to gradient
%                 (incidentally computed while finding second).
%                     first{i}    = d data / dx_i
%
% Every nonempty element of second (and first) is the same size as
%   the data array.
%
% Why is a gradient approximation provided?
%   A gradient approximation is part of the process of computing the mixed
%   partial terms in the Hessian, so returning its value requires
%   little extra computation.  Note that this gradient is a second order
%   centered difference approximation, so it is inappropriate for use in
%   the convection term of a PDE (upwinding should be used for such terms).

% Copyright 2004 Ian M. Mitchell (mitchell@cs.ubc.ca).
% This software is used, copied and distributed under the licensing
%   agreement contained in the file LICENSE in the top directory of
%   the distribution.
%
% Ian Mitchell, 6/3/03

%---------------------------------------------------------------------------
dxInv = 1 ./ grid.dx;

% How big is the stencil?
stencil = 2;

% Add ghost cells to every dimension.
data = addGhostAllDims(grid, data, stencil);

%---------------------------------------------------------------------------
% We need indices to the real data.
indReal = cell(grid.dim, 1);
for i = 1 : grid.dim
    indReal{i} = 1 + stencil : grid.N(i) + stencil;
end

% Also indices to the whole data set (including ghost cells).
indAll = cell(grid.dim, 1);
for i = 1 : grid.dim
    indAll{i} = 1 : grid.N(i) + 2 * stencil;
end
```

```
% data = addGhostAllDims(grid, data, stencil);

%--------------------------------------------------------------------------
% Centered first partials (gradient approximation).
first = cell(grid.dim, 1);
for i = 1 : grid.dim
    % leave the ghost cells on other dimensions intact (for mixed partials below)
    indices1 = indAll;
    indices2 = indAll;
    indices1{i} = indReal{i} + 1;
    indices2{i} = indReal{i} - 1;
    first{i} = 0.5 * dxInv(i) * (data(indices1{:}) - data(indices2{:}));
end

%--------------------------------------------------------------------------
% Centered second partials (Hessian approximation).
% We will only fill the lower half of second,
% since mixed partials' derivative ordering doesn't matter.
second = cell(grid.dim, grid.dim);

for i = 1 : grid.dim
    % First, the pure second partials.
    % Get rid of ghost cells on other dimensions.
    indices1 = indReal;
    indices2 = indReal;
    indices1{i} = indices1{i} + 1;
    indices2{i} = indices2{i} - 1;
    second{i,i} = dxInv(i).^2 * (data(indices1{:}) - 2 * data(indReal{:}) ...
        + data(indices2{:}));

    % Now the mixed partials.
    for j = 1 : i - 1
        % Get rid of ghost cells in dimensions without derivatives.
        indices1 = indReal;
        indices2 = indReal;
        % In already differentiated dimension, we have no ghost cells.
        indices1{i} = 1 : grid.N(i);
        indices2{i} = 1 : grid.N(i);
        % Now take a centered difference in second direction.
        indices1{j} = indReal{j} + 1;
        indices2{j} = indReal{j} - 1;
        second{i,j} =  0.5 * dxInv(j) * (first{i}(indices1{:}) ...
            - first{i}(indices2{:}));
    end
end

%--------------------------------------------------------------------------
% If the user wants the gradient approximation,
%   strip unnecessary ghost cells from first partials.
```

```
for i = 1 : grid.dim
    indices1 = indReal;
    % In already differentiated dimension, we have no ghost cells.
    indices1{i} = 1 : grid.N(i);
    first{i} = first{i}(indices1{:});
end



function near = isNearInterface(data, interface_level, strict_opposite)
% isNearInterface: true if a node has a neighbor across the interface.
%
%   near = isNearInterface(data, interface_level, strict_opposite)
%
% For each node in the data array, determines whether that node has any
% neighbors (left/right in each dimension) which lie on the other side of
% the interface.  If the interface is the zero level set, then nodes are
% next to the interface if any of their neighbors have the opposite sign.
% Nodes lying precisely on the interface are "near the interface," but
% whether their neighbors are also near depends on the input argument
% strict_opposite.
%
% Input Parameters:
%
%   data: Array of values for each node in the grid.  Note that the grid
%   itself is not necessary for this calculation.
%
%   interface_level: Scalar.  Value which represents the "interface."
%   Optional.  Default = 0.
%
%   strict_opposite: Boolean.  Should "neighbor on the opposite side"
%   include neighbors lying precisely on the interface?  Optional.  Default
%   is 0 (a neighbor lying on the interface is "opposite" for all nodes that
%   are not on the interface).  Note that nodes which lie on the interface
%   are ALWAYS "near the interface," regardless of the value of this
%   parameters; this parameter only affects their neighbors.
%
% Output Parameters:
%
%   near: Boolean array, same size as data.  A node's value is 1 if that
%   node is on the interface or if that node has a neighbor on the
%   opposite side of the interface.

% Copyright 2007 Ian M. Mitchell (mitchell@cs.ubc.ca).
% This software is used, copied and distributed under the licensing
%   agreement contained in the file LICENSE in the top directory of
%   the distribution.
%
```

```
% Ian Mitchell, 5/5/07

%----------------------------------------------------------------------------
if(nargin < 2)
    interface_level = 0;
end

if(nargin < 3)
    strict_opposite = 0;
end

% All we care about is on which side of the interface a node lies.
sign_data = sign(data - interface_level);

% Nodes exactly on the interface are "near"
near = (sign_data == 0);

% To compare against neighbors, we need some index cell vectors.
data_dims = ndims(data);
data_size = size(data);
indexL = cell(data_dims, 1);
for d = 1 : data_dims
    indexL{d} = 1 : data_size(d);
end
indexR = indexL;

% Work through dimensions, looking left and right for sign differences.
% Neighbors are not on the "opposite side" if they are directly on the
% interface.
for d = 1 : data_dims

    % Offset the index arrays for this dimension.
    indexL{d} = 1 : data_size(d) - 1;
    indexR{d} = 2 : data_size(d);

    % Find the nodes near the interface.

    if strict_opposite
        % Neighbors on the interface don't count.
        near_d = (sign_data(indexL{:}).*sign_data(indexR{:}) < 0);
    else
        % Any neighbor on or across the interface counts.
        near_d = (sign_data(indexL{:}) ~= sign_data(indexR{:}));
    end

    % If we detected a nearness, it applies to both the node on the left
    % and the node on the right.
    near(indexL{:}) = near(indexL{:}) | near_d;
    near(indexR{:}) = near(indexR{:}) | near_d;
```

```
    % Reset the index arrays for the next dimension.
    indexL{d} = 1 : data_size(d);
    indexR{d} = 1 : data_size(d);

end



function laplacian = laplacianSecond(grid, data)
% laplacian: second order centered difference approx of the Laplacian.
%
%   laplacian = laplacianSecond(grid, data)
%
% Computes a second order centered difference approximation to
%   the Laplacian.
%
%       \Delta \phi = \grad \dot \grad \phi
%                   = \grad^2 \phi
%                   = sum_i d^2 \phi / d x_i^2
%
% parameters:
%   grid    Grid structure (see processGrid.m for details).
%   data        Data array.
%
%   laplacian   Laplacian approximation (same size as data).

% Copyright 2004 Ian M. Mitchell (mitchell@cs.ubc.ca).
% This software is used, copied and distributed under the licensing
%   agreement contained in the file LICENSE in the top directory of
%   the distribution.
%
% Ian Mitchell, 02/02/04

% Current implementation uses hessianSecond, which also computes the
%   second order mixed partial terms.
% If a good use is found for this routine, it would make sense to
%   increase its efficiency by computing just the necessary second
%   order terms.

%---------------------------------------------------------------------------
% Get the second derivative terms.
[second,~] = hessianSecond(grid, data);

%---------------------------------------------------------------------------
laplacian = second{1,1};
for i = 2 : grid.dim;
    laplacian = laplacian + second{i,i};
end
```

```
function [ t, y, schemeData ] = ...
                              odeCFL2(schemeFunc, tspan, y0, options, schemeData)
% odeCFL2: integrate a CFL constrained ODE (eg a PDE by method of lines).
%
% [ t, y, schemeData ] = odeCFL2(schemeFunc, tspan, y0, options, schemeData)
%
% Integrates a system forward in time by CFL constrained timesteps
%   using a second order Total Variation Diminishing (TVD) Runge-Kutta
%   (RK) scheme.  Details can be found in O&F chapter 3.
%
% parameters:
%   schemeFunc   Function handle to a CFL constrained ODE system
%                  (typically an approximation to an HJ term, see below).
%   tspan        Range of time over which to integrate (see below).
%   y0           Initial condition vector
%                  (typically the data array in vector form).
%   options      An option structure generated by odeCFLset
%                  (use [] as a placeholder if necessary).
%   schemeData   Structure passed through to schemeFunc.
%
%
%   t            Output time(s) (see below).
%   y            Output state (see below).
%   schemeData   Output version of schemeData (see below).
%
% A CFL constrained ODE system is described by a function with prototype
%
%       [ ydot, stepBound ] = schemeFunc(t, y, schemeData)
%
%   where t is the current time, y the current state vector and
%   schemeData is passed directly through.  The output stepBound
%   is the maximum allowed time step that will be taken by this function
%   (typically the option parameter factorCFL will choose a smaller step size).
%
% The time interval tspan may be given as
%   1) A two entry vector [ t0 tf ], in which case the output will
%      be scalar t = tf and a row vector y = y(tf).
%   2) A vector with three or more entries, in which case the output will
%      be column vector t = tspan and each row of y will be the solution
%      at one of the times in tspan.  Unlike Matlab's ode suite routines,
%      this version just repeatedly calls version (1), so it is not
%      particularly efficient.
%
% Note that using this routine for integrating HJ PDEs will usually
%   require that the data array be turned into a vector before the call
%   and reshaped into an array after the call.  Option (2) for tspan should
%   not be used in this case because of the excessive memory requirements
%   for storing solutions at multiple timesteps.
%
```

```
% The output version of schemeData will normally be identical to the input
%   version, and therefore can be ignored.  However, if a PostTimestep
%   routine is used (see odeCFLset) then schemeData may be modified during
%   integration, and the version of schemeData at tf is returned in this
%   output argument.

% Copyright 2005 Ian M. Mitchell (mitchell@cs.ubc.ca).
% This software is used, copied and distributed under the licensing
%   agreement contained in the file LICENSE in the top directory of
%   the distribution.
%
% Ian Mitchell, 5/14/03.
% Calling parameters modified to more closely match Matlab's ODE suite
%   Ian Mitchell, 2/14/04.
% Modified to allow vector level sets.  Ian Mitchell, 12/13/04.
% Modified to add terminalEvent option, Ian Mitchell, 1/30/05.

  %---------------------------------------------------------------------------
  % How close (relative) do we need to be to the final time?
  small = 100 * eps;

  %---------------------------------------------------------------------------
  % Make sure we have the default options settings
  if((nargin < 4) | isempty(options))
    options = odeCFLset;
  end

  %---------------------------------------------------------------------------
  % This routine includes multiple substeps, and the CFL restricted timestep
  %   size is chosen on the first substep.  Subsequent substeps may violate
  %   CFL slightly; how much should be allowed before generating a warning?

  % This choice allows 20% more than the user specified CFL number,
  %   capped at a CFL number of unity.  The latter cap may cause
  %   problems if the user is using a very aggressive CFL number.
  safetyFactorCFL = min(1.0, 1.2 * options.factorCFL);

  %---------------------------------------------------------------------------
  % Number of timesteps to be returned.
  numT = length(tspan);

  %---------------------------------------------------------------------------
  % If we were asked to integrate forward to a final time.
  if(numT == 2)

    % - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
    % Is this a vector level set integration?
    if(iscell(y0))
      numY = length(y0);
```

```
  % We need a cell vector form of schemeFunc.
  if(iscell(schemeFunc))
    schemeFuncCell = schemeFunc;
  else
    [ schemeFuncCell{1:numY} ] = deal(schemeFunc);
  end

else
  % Set numY, but be careful: ((numY == 1) & iscell(y0)) is possible.
  numY = 1;

  % We need a cell vector form of schemeFunc.
  schemeFuncCell = { schemeFunc };

end

% - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
t = tspan(1);
steps = 0;
startTime = cputime;
stepBound = zeros(numY, 1);
ydot = cell(numY, 1);
y = y0;

while(tspan(2) - t >= small * abs(tspan(2)))

  % - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
  % First substep: Forward Euler from t_n to t_{n+1}.

  % Approximate the derivative and CFL restriction.
  for i = 1 : numY
    [ ydot{i}, stepBound(i), schemeData ] = ...
                                feval(schemeFuncCell{i}, t, y, schemeData);

    % If this is a vector level set, rotate the lists of vector arguments.
    if(iscell(y))
      y = y([ 2:end, 1 ]);
    end

    if(iscell(schemeData))
      schemeData = schemeData([ 2:end, 1 ]);
    end
  end

  % - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
  % Determine CFL bound on timestep, but not beyond the final time.
  %   For vector level sets, use the most restrictive stepBound.
  %   We'll use this fixed timestep for both substeps.
```

```
deltaT = min([ options.factorCFL * stepBound; ...
               tspan(2) - t; options.maxStep ]);

% Take the first substep.
t1 = t + deltaT;
if(iscell(y))
  y1 = cell(numY, 1);
  for i = 1 : numY
    y1{i} = y{i} + deltaT * ydot{i};
  end
else
  y1 = y + deltaT * ydot{1};
end

% - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
% Second substep: Forward Euler from t_{n+1} to t_{n+2}.

% Approximate the derivative.
%   We will also check the CFL condition for gross violation.
for i = 1 : numY
  [ ydot{i}, stepBound(i), schemeData ] = ...
                             feval(schemeFuncCell{i}, t1, y1, schemeData);

  % If this is a vector level set, rotate the lists of vector arguments.
  if(iscell(y1))
    y1 = y1([ 2:end, 1 ]);
  end

  if(iscell(schemeData))
    schemeData = schemeData([ 2:end, 1 ]);
  end
end

% Check CFL bound on timestep:
%   If the timestep chosen on the first substep violates
%   the CFL condition by a significant amount, throw a warning.
%   For vector level sets, use the most restrictive stepBound.
% Occasional failure should not cause too many problems.
if(deltaT > min(safetyFactorCFL * stepBound))
  violation = deltaT / stepBound;
  warning('Second substep violated CFL; effective number %f', violation);
end

% Take the second substep.
t2 = t1 + deltaT;
if(iscell(y1))
  y2 = cell(numY, 1);
  for i = 1 : numY
    y2{i} = y1{i} + deltaT * ydot{i};
```

```
  end
else
  y2 = y1 + deltaT * ydot{1};
end

% - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
% If there is a terminal event function registered, we need
%   to maintain the info from the last timestep.
if(~isempty(options.terminalEvent))
  yOld = y;
  tOld = t;
end

% Average t_n and t_{n+2} to get second order approximation of t_{n+1}.
t = 0.5 * (t + t2);
if(iscell(y2))
  for i = 1 : numY
    y{i} = 0.5 * (y{i} + y2{i});
  end
else
  y = 0.5 * (y + y2);
end

steps = steps + 1;

% - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
% If there is one or more post-timestep routines, call them.
if(~isempty(options.postTimestep))
  [ y, schemeData ] = odeCFLcallPostTimestep(t, y, schemeData, options);
end

% If we are in single step mode, then do not repeat.
if(strcmp(options.singleStep, 'on'))
  break;
end

% If there is a terminal event function, establish initial sign
%   of terminal event vector.
if(~isempty(options.terminalEvent))

  [ eventValue, schemeData ] = ...
        feval(options.terminalEvent, t, y, tOld, yOld, schemeData);

  if((steps > 1) && any(sign(eventValue) ~= sign(eventValueOld)))
    break;
  else
    eventValueOld = eventValue;
  end
end
```

```
    end

    endTime = cputime;

    if(strcmp(options.stats, 'on'))
      fprintf('\t%d steps in %g seconds from %g to %g\n', ...
              steps, endTime - startTime, tspan(1), t);
    end

  %--------------------------------------------------------------------------
  elseif(numT > 2)
    % If we were asked for the solution at multiple timesteps.
    [ t, y, schemeData ] = ...
     odeCFLmultipleSteps(@odeCFL2, schemeFunc, tspan, y0, options, schemeData);

  %--------------------------------------------------------------------------
  else
    % Malformed time span.
    error('tspan must contain at least two entries');
  end
```

```
function options = odeCFLset(varargin)
% odeCFLset: Create/alter options for CFL constrained ode integrators.
%
%   options = odeCFLset('name1', value1, 'name2', value2, ...)
%   options = odeCFLset(oldopts, 'name1', value1, ...)
%
% Creates a new options structure (or alters an old one) for CFL
%   constrained ODE integrators.  Basically the same as Matlab's odeset
%   but with not nearly as many options.
%
% If called with no input or output parameters, then all options,
%   their valid values and defaults are listed.
%
% Available options (options names are case insensitive):
%
%   FactorCFL     Scalar by which to multiply CFL timestep bound in order
%                   to determine the timestep to actually take.
%                   Typically in range (0,1), default = 0.5
%                   choose 0.9 for aggressive integration.
%
%   MaxStep       Maximum step size (independent of CFL).
%                   Default is REALMAX.
%
%   PostTimestep  Function handle to a routine with prototype
%                         [ yOut, schemeDataOut ] = f(t, yIn, schemeDataIn)
```

```
%                   which is called after every timestep and can be used
%                   to modify the state vector y or to modify or record
%                   information in the schemeData structure.
%                 May also be a cell vector of such function handles, in
%                   which case all function handles are called in order
%                   after each timestep.
%                 Defaults to [], which calls no function.
%
%   SingleStep    Specifies whether to exit integrator after a single
%                   CFL constrained timestep (for debugging).
%                   Either 'on' or 'off', default = 'off'.
%
%   Stats         Specifies whether to display statistics.
%                   Either 'on' or 'off', default = 'off'.
%
%   TerminalEvent Function handle to a routine with prototype
%                         [ value, schemeDataOut ] = ...
%                                       f(t, y, tOld, yOld, schemeDataIn)
%                   which is called after every timestep and can be used to
%                   halt time integration before the final time is reached.
%                   The input parameters include the state and time from
%                   the previous timestep.  If any element of the
%                   return parameter value changes sign from one timestep
%                   to the next, then integration is terminated and
%                   control is returned to the calling function.
%                 Integration cannot be terminated in this manner until
%                   after at least two timesteps.
%                 Unlike Matlab's ODE event system, no attempt is made
%                   to accurately locate the time at which the event
%                   function passed through zero.
%                 If both are present, the terminalEvent function will be
%                   called after all postTimestep functions.
%                 Defaults to [], which calls no function.

% Copyright 2005-2008 Ian M. Mitchell (mitchell@cs.ubc.ca).
% This software is used, copied and distributed under the licensing
%   agreement contained in the file LICENSE in the top directory of
%   the distribution.
%
% Created by Ian Mitchell, 2/6/04
% Date : 2010 - 08 - 0921 : 31 : 46 - 0700(Mon, 09Aug2010)
% Id : odeCFLset.m502010 - 08 - 1004 : 31 : 46Zmitchell

%---------------------------------------------------------------------------
% No output, no input means caller just wants a list of available options.
if((nargin == 0) && (nargout == 0))
    fprintf('    factorCFL: [ positive scalar {0.5} ]\n');
    fprintf('      maxStep: [ positive scalar {REALMAX} ]\n');
    fprintf([ ' postTimestep: [ function handle | '...
```

```
        'cell vector of function handles | {[]} ]\n' ]);
    fprintf('   singleStep: [ on | {off} ]\n');
    fprintf('        stats: [ on | {off} ]\n');
    fprintf('terminalEvent: [ function handle | {[]} ]\n');
    fprintf('\n');
    return;
end

%------------------------------------------------------------------------
% First input argument is an old options structure
if((nargin > 0) && isstruct(varargin{1}))
    options = varargin{1};
    startArg = 2;
else
    % Create the default options structure.
    options.factorCFL = 0.5;
    options.maxStep = realmax;
    options.postTimestep = [];
    options.singleStep = 'off';
    options.stats = 'off';
    options.terminalEvent = [];
    startArg = 1;
end

%------------------------------------------------------------------------
% Loop through remaining name value pairs
for i = startArg : 2 : nargin
    name = varargin{i};
    value = varargin{i+1};

    % Remember that the case labels are lower case.
    switch(lower(name))
        case 'factorcfl'
            if(isa(value, 'double') && (prod(size(value)) == 1)...
                    && (value > 0.0))
                options.factorCFL = value;
            else
                error('FactorCFL must be a positive scalar double value');
            end

        case 'maxstep'
            if(isa(value, 'double') && (prod(size(value)) == 1)...
                    && (value > 0.0))
                options.maxStep = value;
            else
                error('MaxStep must be a positive scalar double value');
            end

        case 'posttimestep'
```

```
            if(isa(value, 'function_handle') || isempty(value))
                options.postTimestep = value;
            elseif(isa(value, 'cell'))
                for j = 1 : length(value)
                    if(~isa(value{j}, 'function_handle'))
                        error([ 'Each element in a postTimestep cell' ...
                            'vector must be a function handle.' ]);
                    end
                end
                options.postTimestep = value;
            else
                error([ 'PostTimestep parameter must be a function' ...
                    ' handle ora cell vector of function handles.' ]);
            end

        case 'singlestep'
            if(isa(value, 'char') && (strcmp(value, 'on') ||...
                    (strcmp(value, 'off'))))
                options.singleStep = value;
            else
                error(['SingleStep must be one of the strings'...
                    ' ''on'' or ''off'''']);
            end

        case 'stats'
            if(isa(value, 'char') && (strcmp(value, 'on') ||...
                    (strcmp(value, 'off'))))
                options.stats = value;
            else
                error('Stats must be one of the strings ''on'' or ''off''');
            end

        case 'terminalevent'
            if(isa(value, 'function_handle') || isempty(value))
                options.terminalEvent = value;
            else
                error('PostTimestep parameter must be a function handle.');
            end

        otherwise
            error('Unknown odeCFL option %s', name);

    end
end



function gridOut = processGrid(gridIn, data)
% processGrid: Construct a grid data structure, and check for consistency.
```

```
%
%   gridOut = processGrid(gridIn, data)
%
% Processes all the various types of grid argument allowed.
%
% Input Parameters:
%
%   gridIn: A scalar, a vector, or a structure.
%
%     Scalar: It is assumed to be the dimension.  See below for default
%     settings for other grid fields.
%
%     Vector: It contains the number of grid nodes in each dimension.  See
%     below for default settings for other fields.
%
%     Structure: It must contain some subset of the following fields
%     (where each vector has length equal to the number of dimensions):
%
%         gridIn.dim: Positive integer scalar, dimension of the grid.
%
%         gridIn.min: Double vector specifying the lower left corner of the
%         grid.
%
%         gridIn.max: Double vector specifying the upper right corner of
%         the grid.
%
%         gridIn.N: Positive integer vector specifying the number of grid
%         nodes in each dimension.
%
%         gridIn.dx: Positive double vector specifying the grid spacing in
%         each dimension.
%
%         gridIn.vs: Cell vector, each element is a vector of node locations
%         for that dimension.
%
%         gridIn.xs: Cell vector, each element is an array of node locations
%         (result of calling ndgrid on vs).
%
%         gridIn.bdry: Cell vector of function handles pointing to boundary
%           condition generating functions for each dimension.
%
%       gridIn.bdryData: Cell vector of data structures for the boundary
%           condition generating functions.
%
%       gridIn.axis: Vector specifying computational domain bounds in a
%       format suitable to pass to the axis() command (only defined for 2D
%       and 3D grids, otherwise grid.axis == []).
%
%       gridIn.shape: Vector specifying grid node count in a format suitable
```

```
%       to pass to the reshape() command (usually grid.N', except for 1D
%       grids).
%
%     If any of the following fields are scalars, they are replicated
%     gridIn.dim times: min, max, N, dx, bdry, bdryData.
%
%     In general, it is not necessary to supply the fields: vs, xs, axis,
%     shape.
%
%     If one of N or dx is supplied, the other is inferred.
%     If both are supplied, consistency is checked.
%
%     Dimensional consistency is checked on all fields.
%
%     Default settings (only used if value is not given or inferred)
%         min   = zeros(dim, 1)
%         max   = ones(dim, 1)
%         N     = 101
%         bdry  = periodic
%
%   data: Double array.  Optional.  If present, the data array is checked
%   for consistency with the grid.
%
% Output Parameters:
%
%   gridOut: the full structure described for gridIn above.

% Copyright 2004 Ian M. Mitchell (mitchell@cs.ubc.ca).
% This software is used, copied and distributed under the licensing
%   agreement contained in the file LICENSE in the top directory of
%   the distribution.
%
% Ian Mitchell, 1/22/03
%  new version  5/13/03 added fields dim, dx, vs, xs, bdry.
%  new version  1/13/04 added field bdryData.
%  new version  2/09/04 added field shape.
%  new version  8/23/12 fixed some floating point problems with N and dx.

%---------------------------------------------------------------------------
defaultMin = 0;
defaultMax = 1;
defaultN = 101;
defaultBdry = @addGhostPeriodic;
defaultBdryData = [];

% This is just to avoid attempts to allocate 100 dimensional arrays.
maxDimension = 5;

%---------------------------------------------------------------------------
```

```
if(~isstruct(gridIn))
    if(numel(gridIn) == 1)
        gridOut.dim = gridIn;
    elseif(ndims(gridIn) == 2)
        % Should be a vector of node counts.
        if(size(gridIn, 2) ~= 1)
            error('gridIn vector must be a column vector');
        else
            gridOut.dim = length(gridIn);
            gridOut.N = gridIn;
        end
    else
        error('Unknown format for gridIn parameter');
    end
else
    gridOut = gridIn;
end


%--------------------------------------------------------------------------
% Now we should have a partially complete structure in gridOut.

if(isfield(gridOut, 'dim'))
    if(gridOut.dim > maxDimension)
        error('dimension > %d, may be dangerously large', maxDimension);
    end
    if(gridOut.dim < 0)
        error('dimension must be positive');
    end
else
    error('grid structure must contain dimension');
end

%--------------------------------------------------------------------------
% Process grid boundaries.

if(isfield(gridOut, 'min'))
    if(~isColumnLength(gridOut.min, gridOut.dim))
        if(isscalar(gridOut.min))
            gridOut.min = gridOut.min * ones(gridOut.dim, 1);
        else
            error('min field is not column vector of length dim or a scalar');
        end
    else
        gridOut.min = gridOut.min .* ones(gridOut.dim, 1);
    end
else
    gridOut.min = defaultMin * ones(gridOut.dim, 1);
end
```

```
if(isfield(gridOut, 'max'))
    if(~isColumnLength(gridOut.max, gridOut.dim))
        if(isscalar(gridOut.max))
            gridOut.max = gridOut.max * ones(gridOut.dim, 1);
        else
            error('max field is not column vector of length dim or a scalar');
        end
    end
else
    gridOut.max = defaultMax * ones(gridOut.dim, 1);
end

if(any(gridOut.max <= gridOut.min))
    error('max bound must be greater than min bound in all dimensions');
end

%--------------------------------------------------------------------------
% Check N field if necessary.  If N is missing but dx is present, we will
% determine N later.
if(isfield(gridOut, 'N'))
    if(any(gridOut.N <= 0))
        error('number of grid cells must be strictly positive');
    end
    if(~isColumnLength(gridOut.N, gridOut.dim))
        if(isscalar(gridOut.N))
            gridOut.N = gridOut.N * ones(gridOut.dim, 1);
        else
            error('N field is not column vector of length dim or a scalar');
        end
    end
end

%--------------------------------------------------------------------------
% Check dx field if necessary.  If dx is missing but N is present, infer
% dx.  If both are present, we will check for consistency later.  If
% neither are present, use the defaults.
if isfield(gridOut, 'dx')
    if(any(gridOut.dx <= 0))
        error('grid cell size dx must be strictly positive');
    end
    if(~isColumnLength(gridOut.dx, gridOut.dim))
        if(isscalar(gridOut.dx))
            gridOut.dx = gridOut.dx * ones(gridOut.dim, 1);
        else
            error('dx field is not column vector of length dim or a scalar');
        end
    end
elseif isfield(gridOut, 'N')
```

```
    % Only N field is present, so infer dx.
    gridOut.dx = (gridOut.max - gridOut.min) ./ (gridOut.N - 1);
else
    % Neither field is present, so use default N and infer dx
    gridOut.N = defaultN * ones(gridOut.dim, 1);
    gridOut.dx = (gridOut.max - gridOut.min) ./ (gridOut.N - 1);
end

%--------------------------------------------------------------------------
if(isfield(gridOut, 'vs'))
    if(iscell(gridOut.vs))
        if(~isColumnLength(gridOut.vs, gridOut.dim))
            error('vs field is not column cell vector of length dim');
        else
            for i = 1 : gridOut.dim
                if(~isColumnLength(gridOut.vs{i}, gridOut.N(i)))
                    error('vs cell entry is not correctly sized vector');
                end
            end
        end
    else
        error('vs field is not a cell vector');
    end
else
    gridOut.vs = cell(gridOut.dim, 1);
    for i = 1 : gridOut.dim
        gridOut.vs{i} = (1e-10*round(1e10*gridOut.min(i)) :...
            1e-10*round(1e10*gridOut.dx(i)) :...
            1e-10*round(1e10*gridOut.max(i)))';
    end
end

% Now we can check for consistency between dx and N, based on the size of
% the vectors in vs.  Note that if N is present, it will be a vector.  If
% N is not yet a field, set it to be consistent with the size of vs.
if isfield(gridOut, 'N')
    for i = 1 : gridOut.dim
        if(gridOut.N(i) ~= length(gridOut.vs{i}))
            error('Inconsistent grid size in dimension %d', i);
        end
    end
else
    gridOut.N = zeros(gridOut.dim, 1);
    for i = 1 : gridOut.dim
        gridOut.N(i) = length(gridOut.vs{i});
    end
end

%--------------------------------------------------------------------------
```

```
if(isfield(gridOut, 'xs'))
    if(iscell(gridOut.xs))
        if(~isColumnLength(gridOut.xs, gridOut.dim))
            error('xs field is not column cell vector of length dim');
        else
            if(gridOut.dim > 1)
                for i = 1 : gridOut.dim
                    if(any(size(gridOut.xs{i}) ~= gridOut.N'))
                        error('xs cell entry is not correctly sized array');
                    end
                end
            else
                if(length(gridOut.xs{1}) ~= gridOut.N)
                    error('xs cell entry is not correctly sized array');
                end
            end
        end
    else
        error('xs field is not a cell vector');
    end
else
    gridOut.xs = cell(gridOut.dim, 1);
    if(gridOut.dim > 1)
        [ gridOut.xs{:} ] = ndgrid(gridOut.vs{:});
    else
        gridOut.xs{1} = gridOut.vs{1};
    end
end

%--------------------------------------------------------------------------
if(isfield(gridOut, 'bdry'))
    if(iscell(gridOut.bdry))
        if(~isColumnLength(gridOut.bdry, gridOut.dim))
            error('bdry field is not column cell vector of length dim');
        else
            for i = 1 : gridOut.dim
                % I don't know how to check if the entries are
                % function handles
            end
        end
    else
        if(isscalar(gridOut.bdry))
            bdry = gridOut.bdry;
            gridOut.bdry = cell(gridOut.dim, 1);
            [ gridOut.bdry{:} ] = deal(bdry);
        else
            error('bdry field is not a cell vector or a scalar');
        end
    end
```

```
else
    gridOut.bdry = cell(gridOut.dim, 1);
    [ gridOut.bdry{:} ] = deal(defaultBdry);
end

%--------------------------------------------------------------------------
if(isfield(gridOut, 'bdryData'))
    if(iscell(gridOut.bdryData))
        if(~isColumnLength(gridOut.bdryData, gridOut.dim))
            error('bdryData field is not column cell vector of length dim');
        else
            for i = 1 : gridOut.dim
                % Don't know whether it is worth checking that
                % entries are structures
            end
        end
    else
        if(isscalar(gridOut.bdryData))
            bdryData = gridOut.bdryData;
            gridOut.bdryData = cell(gridOut.dim, 1);
            [ gridOut.bdryData{:} ] = deal(bdryData);
        else
            error('bdryData field is not a cell vector or a scalar');
        end
    end
else
    gridOut.bdryData = cell(gridOut.dim, 1);
    [ gridOut.bdryData{:} ] = deal(defaultBdryData);
end

%--------------------------------------------------------------------------
if((gridOut.dim == 2) || (gridOut.dim == 3))
    if(isfield(gridOut, 'axis'))
        for i = 1 : gridOut.dim
            if(gridOut.axis(2 * i - 1) ~= gridOut.min(i))
                error('axis and min fields do not agree');
            end
            if(gridOut.axis(2 * i)     ~= gridOut.max(i))
                error('axis and max fields do not agree');
            end
        end
    else
        gridOut.axis = zeros(1, 2 * gridOut.dim);
        for i = 1 : gridOut.dim
            gridOut.axis(2 * i - 1 : 2 * i) = [ gridOut.min(i),...
                gridOut.max(i) ];
        end
    end
else
```

```
    gridOut.axis = [];
end

%--------------------------------------------------------------------------
if(isfield(gridOut, 'shape'))
    if(gridOut.dim == 1)
        if(any(gridOut.shape ~= [ gridOut.N, 1 ]))
            error('shape and N fields do not agree');
        end
    else
        if(any(gridOut.shape ~= gridOut.N'))
            error('shape and N fields do not agree');
        end
    end
else
    if(gridOut.dim == 1)
        gridOut.shape = [ gridOut.N, 1 ];
    else
        gridOut.shape = gridOut.N';
    end
end

%--------------------------------------------------------------------------
% check data parameter for consistency
if(nargin > 1)
    if(ndims(data) ~= length(gridOut.shape))
        error('data parameter does not agree in dimension with grid');
    end
    if(any(size(data) ~= gridOut.shape))
        error('data parameter does not agree in array size with grid');
    end
end

end % processGrid().

%--------------------------------------------------------------------------
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%--------------------------------------------------------------------------
function bool = isColumnLength(array, vectorLength)
%  bool = isColumnLength(array, vectorLength)
%
% helper function to check that an array is a column vector of some length

bool = ((ndims(array) == 2) & ...
    (size(array, 1) == vectorLength) & (size(array, 2) == 1));

end % isColumnLength().

%--------------------------------------------------------------------------
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%------------------------------------------------------------------------------
function bool = isscalar(array)
%  bool = isscalar(array)
%
% helper function which checks whether the array is a scalar

bool = (numel(array) == 1);

end % isscalar().



function data = signedDistanceIterative(grid, data0, accuracy, tMax, errorMax)
% signedDistanceIterative: Create a signed distance function iteratively.
%
%   data = signedDistanceIterative(grid, data0, accuracy, tMax, errorMax)
%
% Converts an implicit surface function into a signed distance function
%   by iterative solution of the reinitialization equation.
%
% Iterations continue to a fixed time or until the average relative change
%   in the function value between iterations drops low enough, whichever
%   comes first.
%
% In the reinitialization equation, information flows outward from the zero
%   level set at "speed" one, so to get a signed distance function in a band
%   of at least 10 grid cells around the zero level set, choose a minimum
%   tMax = 10 * max(grid.dx).
%
% Parameters:
%
%   grid     Grid structure.
%   data0        Implicit surface function.
%   accuracy     Controls the order of approximations.
%                  'low'         Use odeCFL1 and upwindFirstFirst.
%                  'medium'      Use odeCFL2 and upwindFirstENO2 (default).
%                  'high'        Use odeCFL3 and upwindFirstENO3.
%                  'veryHigh'    Use odeCFL3 and upwindFirstWENO5.
%   tMax         Time at which to halt the reinitialization iteration
%                  (default = max(grid.max - grid.min)).
%                  If tMax < 0, it is interpreted as the number of CFL
%                  limited reinitialization timesteps to take:
%                  number of steps = -round(tMax).
%   errorMax     If the average update of nodes drops below
%                  errorMax * max(grid.dx), then assume that
%                  reinitialization has converged and return early
%                  (default = 1e-3).
%
```

```
%
%   data         Signed distance function.

% Copyright 2005 Ian M. Mitchell (mitchell@cs.ubc.ca).
% This software is used, copied and distributed under the licensing
%   agreement contained in the file LICENSE in the top directory of
%   the distribution.
%
% Ian Mitchell, 2/14/04
% Modified to accept vector level sets, Ian Mitchell 2/16/05

%---------------------------------------------------------------------------
% How close (relative) do we need to get to tMax to be considered finished?
small = 100 * eps;

% How aggressive should we be with the CFL condition?
factorCFL = 0.95;

%---------------------------------------------------------------------------
% Set defaults.
if(nargin < 3)
  accuracy = 'medium';
end

if(nargin < 4)
  tMax = max(grid.max - grid.min);
end

if(tMax < 0)
  stepMax = -round(tMax);
  tMax = +inf; % prod(grid.N) * max(grid.max - grid.min);
else
  stepMax = +inf;
end

if(nargin < 5)
  errorMax = 1e-3;
end

%---------------------------------------------------------------------------
% If this is a vector level set, each element of the vector is
%   reinitialized independently.
if(iscell(data0))
  data = cell(length(data0), 1);
  for i = 1 : length(data0)
    if(iscell(grid))
      data{i} = signedDistanceIterative(grid{i}, data0{i}, accuracy, ...
                                        tMax, errorMax);
    else
```

```
      data{i} = signedDistanceIterative(grid, data0{i}, accuracy, ...
                                        tMax, errorMax);
    end
  end
end

%---------------------------------------------------------------------------
% Set up spatial approximation scheme.
schemeFunc = @termReinit;
schemeData.grid = grid;
% Just in case original data is in column vector format.
schemeData.initial = reshape(data0, grid.shape);

% Set up time approximation scheme.
%   Single step so that we can check convergence criterion.
integratorOptions = odeCFLset('factorCFL', factorCFL, 'singleStep', 'on');

% Choose approximations at appropriate level of accuracy.
switch(accuracy)
 case 'low'
  schemeData.derivFunc = @upwindFirstFirst;
  integratorFunc = @odeCFL1;
 case 'medium'
  schemeData.derivFunc = @upwindFirstENO2;
  integratorFunc = @odeCFL2;
 case 'high'
  schemeData.derivFunc = @upwindFirstENO3;
  integratorFunc = @odeCFL3;
 case 'veryHigh'
  schemeData.derivFunc = @upwindFirstWENO5;
  integratorFunc = @odeCFL3;
 otherwise
  error('Unknown accuracy level %s', accuracy);
end


%---------------------------------------------------------------------------
% Convergence criteria
deltaMax = errorMax * max(grid.dx) * prod(grid.N);

%---------------------------------------------------------------------------
% Reshape data array into column vector for ode solver call (if necessary).
dataSize = size(data0);
if((length(dataSize) == 2) ...
   && (dataSize(1) == prod(grid.N)) && (dataSize(2) == 1))
  % Data is already in column vector form.
  y = data0;
  reshaped = 0;
elseif((length(dataSize) == length(grid.shape)) && all(dataSize == grid.shape))
```

```
  % Reshape to column vector form.
  y = data0(:);
  reshaped = 1;
else
  error('Data array is not the same size as grid');
end

%-----------------------------------------------------------------------------
% Loop until tMax (subject to a little roundoff) or stepMax.
tNow = 0;
step = 0;
while((tMax - tNow >= small * tMax) & (step < stepMax))

  % Check for convergence (except for the first loop).
  if((tNow > 0) && (norm(y - y0, 1) < deltaMax))
    break;
  end

  % Always try to finish (in fact, single timesteps are taken).
  tSpan = [ tNow, tMax ];

  % Take a single timestep.
  y0 = y;
  [ t y ] = feval(integratorFunc, schemeFunc, tSpan, y0,...
                  integratorOptions, schemeData);

  tNow = t(end);
  step = step + 1;

end


if(reshaped)
  % Reshape the column vector back into the appropriate form.
  data = reshape(y, grid.shape);
else
  data = y;
end



function [ ydot, stepBound, schemeData ] = termCurvature(t, y, schemeData)
% termCurvature: approximate a motion by mean curvature term in an HJ PDE.
%
% [ ydot, stepBound, schemeData ] = termCurvature(t, y, schemeData)
%
% Computes an approximation of motion by mean curvature for a
%   Hamilton-Jacobi PDE.  This is a second order equation that simplifies to
%   a heat equation if the function is a signed distance function.
```

```
%   Specifically:
%
%                D_t \phi - b(x,t) \kappa(x) \| \grad \phi \| = 0.
%
%   where \kappa(x) is the mean curvature.
%
% Based on methods outlined in O&F, chapters 4.1 & 4.2.
%
% parameters:
%   t             Time at beginning of timestep.
%   y             Data array in vector form.
%   schemeData    A structure (see below).
%
%   ydot     Change in the data array, in vector form.
%   stepBound     CFL bound on timestep for stability.
%   schemeData    The same as the input argument (unmodified).
%
% schemeData is a structure containing data specific to this type of
%   term approximation.  For this function it contains the field(s)
%
%   .grid      Grid structure (see processGrid.m for details).
%   .curvatureFunc Function handle to finite difference curvature approx.
%                    Should provide both curvature and gradient magnitude.
%   .b         Multiplier (should be non-negative for well-posedness);
%                    b may be a scalar constant or an array of size(data).
%   .passVLS       An optional boolean used for vector level sets (see below).
%                    Default is 0 (ie ignore vector level sets).
%
% It may contain additional fields at the user's discretion.
%
% schemeData.b can provide the multiplier in one of two ways:
%   1) For time invariant multipliers, a scalar or an array the same
%      size as data.
%   2) For general multipliers, a function handle to a function with prototype
%      b = scalarGridFunc(t, data, schemeData), where the output b is the
%      scalar/array from (1) and the input arguments are the same as those
%      of this function (except that data = y has been reshaped to its
%      original size).  In this case, it may be useful to include additional
%      fields in schemeData.
%
% For evolving vector level sets, y may be a cell vector.  If y is a cell
%   vector, schemeData may be a cell vector of equal length.  In this case
%   all the elements of y (and schemeData if necessary) are ignored except
%   the first.  As a consequence, if schemeData.b is a function handle
%   the call to scalarGridFunc will be performed with a regular data array
%   and a single schemeData structure (as if no vector level set was present).
%
% This default behavior of ignoring the vector level set in the call
%   to scalarGridFunc may be overridden by setting schemeData.passVLS = 1.
```

```
%   In this case the data argument (and schemeData argument, if necessary)
%   in the call to velocityFunc will be the full cell vectors.  The current
%   data array (and schemeData structure, if necessary) will be the first
%   element of these cell vectors.  In order to properly reshape the other
%   elements of y, the corresponding schemeData structures must contain
%   an appropriate grid structure.
%
% In the notation of OF text,
%
%   data = y      \phi
%   curvatureFunc function to calculate \kappa and |\grad \phi|
%   b         b
%
%   delta = ydot  +b \kappa |\grad \phi|


% Copyright 2005 Ian M. Mitchell (mitchell@cs.ubc.ca).
% This software is used, copied and distributed under the licensing
%   agreement contained in the file LICENSE in the top directory of
%   the distribution.
%
% Ian Mitchell 6/3/03
% Calling parameters significantly modified, Ian Mitchell 2/13/04.
% Updated to handle vector level sets.  Ian Mitchell 11/23/04.
% Updated to include passVLS option.  Ian Mitchell 02/14/05.

  %---------------------------------------------------------------------------
  if(iscell(schemeData))
    thisSchemeData = schemeData{1};
  else
    thisSchemeData = schemeData;
  end

  checkStructureFields(thisSchemeData, 'grid', 'b', 'curvatureFunc');

  grid = thisSchemeData.grid;

  %---------------------------------------------------------------------------
  if(iscell(y))
    data = reshape(y{1}, grid.shape);
  else
    data = reshape(y, grid.shape);
  end

  %---------------------------------------------------------------------------
  % Get multiplier
  if(isa(thisSchemeData.b, 'double'))
    b = thisSchemeData.b;
```

```
  elseif(isa(thisSchemeData.b, 'function_handle'))

    if(iscell(y))
      % If there is a vector level set.

      if(isfield(thisSchemeData, 'passVLS') && thisSchemeData.passVLS)
        % Pass the vector level set information through.
        numY = length(y);
        dataV = cell(numY, 1);
        for i = 1 : numY
          if(iscell(schemeData))
            dataV{i} = reshape(y{i}, schemeData{i}.grid.shape);
          else
            dataV{i} = reshape(y{i}, schemeData.grid.shape);
          end
        end
        b = feval(thisSchemeData.b, t, dataV, schemeData);

      else
        % Ignore any vector level set.
        b = feval(thisSchemeData.b, t, data, thisSchemeData);
      end

    else
      % There is no vector level set.
      b = feval(thisSchemeData.b, t, data, thisSchemeData);
    end


  else
    error('schemeData.b must be a scalar, array or function handle');

  end

  %----------------------------------------------------------------------------
  % According to O&F equation (4.5).
  [ curvature, gradMag ] = feval(thisSchemeData.curvatureFunc, grid, data);
  delta = -b .* curvature .* gradMag;

  %----------------------------------------------------------------------------
  % According to O&F equation (4.7).
  stepBound = 1 / (2 * max(b(:)) * sum(grid.dx .^ -2));

  % Reshape output into vector format and negate for RHS of ODE.
  ydot = -delta(:);



function [ ydot, stepBound, schemeData ] = ...
```

```
    termNormal_original(t, y, schemeData)
% termNormal: motion in the normal direction in an HJ PDE with upwinding.
%
% [ ydot, stepBound, schemeData ] = termNormal(t, y, schemeData)
%
% Computes an approximation of motion of the interface at speed a(x,t) in
%   the normal direction.  The PDE is:
%
%           D_t \phi + a(x,t) \| \grad \phi \| = 0.
%
%   Based on methods outlined in O&F, chapter 6.  The Godunov scheme from
%   chapter 6.2 is used.
%
% parameters:
%   t            Time at beginning of timestep.
%   y            Data array in vector form.
%   schemeData   A structure (see below).
%
%   ydot     Change in the data array, in vector form.
%   stepBound    CFL bound on timestep for stability.
%   schemeData   The same as the input argument (unmodified).
%
% schemeData is a structure containing data specific to this type of
%   term approximation.  For this function it contains the field(s)
%
%   .grid    Grid structure (see processGrid.m for details).
%   .derivFunc   Function handle to upwinded finite difference
%                  derivative approximation.
%   .speed   A description of the normal speed (see below).
%   .passVLS     An optional boolean used for vector level sets (see below).
%                  Default is 0 (ie ignore vector level sets).
%
% It may contain additional fields at the user's discretion.
%
% schemeData.speed can provide the speed in one of two ways:
%   1) For time invariant speed, a scalar or an array the same
%      size as data.
%   2) For general speed, a function handle to a function with prototype
%      a = scalarGridFunc(t, data, schemeData), where the output a is the
%      scalar/array from (1) and the input arguments are the same as those
%      of this function (except that data = y has been reshaped to its
%      original size).  In this case, it may be useful to include additional
%      fields in schemeData.
%
% For evolving vector level sets, y may be a cell vector.  If y is a cell
%   vector, schemeData may be a cell vector of equal length.  In this case
%   all the elements of y (and schemeData if necessary) are ignored except
%   the first.  As a consequence, if schemeData.speed is a function handle
%   the call to scalarGridFunc will be performed with a regular data array
```

```
%   and a single schemeData structure (as if no vector level set was present).
%
% This default behavior of ignoring the vector level set in the call
%   to scalarGridFunc may be overridden by setting schemeData.passVLS = 1.
%   In this case the data argument (and schemeData argument, if necessary)
%   in the call to velocityFunc will be the full cell vectors.  The current
%   data array (and schemeData structure, if necessary) will be the first
%   element of these cell vectors.  In order to properly reshape the other
%   elements of y, the corresponding schemeData structures must contain
%   an appropriate grid structure.
%
% In the notation of OF text,
%
%   data = y      \phi, reshaped to vector form.
%   derivFunc     Function to calculate phi_i^+-.
%   speed     a.
%
%   delta = ydot  -a \| \grad \phi \|, with upwinded approx to \grad \phi
%                   and reshaped to vector form.


% Copyright 2005 Ian M. Mitchell (mitchell@cs.ubc.ca).
% This software is used, copied and distributed under the licensing
%   agreement contained in the file LICENSE in the top directory of
%   the distribution.
%
% Ian Mitchell 3/1/04.
% Updated to handle vector level sets.  Ian Mitchell 11/23/04.
% Updated to include passVLS option.  Ian Mitchell 02/14/05.

%---------------------------------------------------------------------------
if(iscell(schemeData))
    thisSchemeData = schemeData{1};
else
    thisSchemeData = schemeData;
end

checkStructureFields(thisSchemeData, 'grid', 'derivFunc', 'speed');

grid = thisSchemeData.grid;

%---------------------------------------------------------------------------
% For most cases, we are interested in the first implicit surface function.
if(iscell(y))
    data = reshape(y{1}, grid.shape);
else
    data = reshape(y, grid.shape);
end
```

```
%-------------------------------------------------------------------------
% Get speed field.
if(isa(thisSchemeData.speed, 'double'))
    speed = thisSchemeData.speed;

elseif(isa(thisSchemeData.speed, 'function_handle'))

    if(iscell(y))
        % If there is a vector level set.

        if(isfield(thisSchemeData, 'passVLS') && thisSchemeData.passVLS)
            % Pass the vector level set information through.
            numY = length(y);
            dataV = cell(numY, 1);
            for i = 1 : numY
                if(iscell(schemeData))
                    dataV{i} = reshape(y{i}, schemeData{i}.grid.shape);
                else
                    dataV{i} = reshape(y{i}, schemeData.grid.shape);
                end
            end
            speed = feval(thisSchemeData.speed, t, dataV, schemeData);

        else
            % Ignore any vector level set.
            speed = feval(thisSchemeData.speed, t, data, thisSchemeData);
        end

    else
        % There is no vector level set.
        speed = feval(thisSchemeData.speed, t, data, thisSchemeData);
    end

else
    error('schemeData.speed must be a scalar, array or function handle');

end

%-------------------------------------------------------------------------
% In the end, all we care about is the magnitude of the gradient.
magnitude = zeros(size(data));

% In this case, keep track of stepBound for each node until the very
%   end (since we need to divide by the appropriate gradient magnitude).
stepBoundInv = zeros(size(data));

% Determine the upwind direction dimension by dimension
for i = 1 : grid.dim
```

```
    % Get upwinded derivative approximations.
    [ derivL, derivR ] = feval(thisSchemeData.derivFunc, grid, data, i);

    % Effective velocity in this dimension (scaled by \|\grad \phi\|).
    prodL = speed .* derivL;
    prodR = speed .* derivR;
    magL = abs(prodL);
    magR = abs(prodR);

    % Determine the upwind direction.
    %   Either both sides agree in sign (take direction in which they agree),
    %   or characteristics are converging (take larger magnitude direction).
    flowL = ((prodL >= 0) & (prodR >= 0)) | ...
        ((prodL >= 0) & (prodR <= 0) & (magL >= magR));
    flowR = ((prodL <= 0) & (prodR <= 0)) | ...
        ((prodL >= 0) & (prodR <= 0) & (magL < magR));

    % For diverging characteristics, take gradient = 0
    %   (so we don't actually need to calculate this term).
    %flow0 = ((prodL <= 0) & (prodR >= 0));

    % Now we know the upwind direction, add its contribution to
    % \|\grad \phi\|.
    magnitude = magnitude + derivL.^2 .* flowL + derivR.^2 .* flowR;

    % CFL condition: sum of effective velocities from O&F (6.2).
    effectiveVelocity = magL .* flowL + magR .* flowR;
    dxInv = 1 / grid.dx(i);
    stepBoundInv = stepBoundInv + dxInv * effectiveVelocity;
end

%-----------------------------------------------------------------------------
% Finally, calculate speed * \|\grad \phi\|
magnitude = sqrt(magnitude);
delta = speed .* magnitude;

% Find the most restrictive timestep bound.
nonZero = find(magnitude > 0);
stepBoundInvNonZero = stepBoundInv(nonZero) ./ magnitude(nonZero);
stepBound = 1 / max(stepBoundInvNonZero(:));

% Reshape output into vector format and negate for RHS of ODE.
ydot = -delta(:);



function [ derivL, derivR ] = upwindFirstENO2(grid, data, dim, generateAll)
% upwindFirstENO2: second order upwind approx of first derivative.
%
```

```
%   [ derivL, derivR ] = upwindFirstENO2(grid, data, dim, generateAll)
%
% Computes a second order directional approximation to the first
%   derivative, using a oscillation reducing minimum modulus choice
%   of second order term.  The result is an order 2 ENO scheme.
%
% The approximation is constructed by a divided difference table.
%
% Some details of this scheme can be found in O&F, section 3.3,
%   where this scheme is equivalent to including the Q_1 and Q_2
%   terms of the ENO approximation.
%
% The generateAll option is used for debugging, and possibly by
%   higher order weighting schemes.  Under normal circumstances
%   the default (generateAll = false) should be used.
%
% parameters:
%   grid    Grid structure (see processGrid.m for details).
%   data        Data array.
%   dim         Which dimension to compute derivative on.
%   generateAll Return all possible second order upwind approximations.
%                 If this boolean is true, then derivL and derivR will
%                 be cell vectors containing all the approximations
%                 instead of just the minimum modulus approximation.
%                 (optional, default = 0)
%
%   derivL      Left approximation of first derivative (same size as data).
%   derivR      Right approximation of first derivative (same size as data).

% Copyright 2004 Ian M. Mitchell (mitchell@cs.ubc.ca).
% This software is used, copied and distributed under the licensing
%   agreement contained in the file LICENSE in the top directory of
%   the distribution.
%
% Ian Mitchell, 1/22/04

%---------------------------------------------------------------------------
if((dim < 0) || (dim > grid.dim))
    error('Illegal dim parameter');
end

if(nargin < 4)
    generateAll = 0;
end

dxInv = 1 / grid.dx(dim);

% How big is the stencil?
stencil = 2;
```

```
% Check that approximations that should be equivalent are equivalent
%   (for debugging purposes, only used if generateAll == 1).
checkEquivalentApproximations = 1;
small = 100 * eps;              % a small number for "equivalence"

% Add ghost cells.
gdata = feval(grid.bdry{dim}, data, dim, stencil, grid.bdryData{dim});

%---------------------------------------------------------------------------
% Create cell array with array indices.
sizeData = size(gdata);
indices1 = cell(grid.dim, 1);
for i = 1 : grid.dim
    indices1{i} = 1:sizeData(i);
end
indices2 = indices1;

%---------------------------------------------------------------------------
% First divided differences (first entry corresponds to D^1_{-1/2}).
indices1{dim} = 2 : size(gdata, dim);
indices2{dim} = indices1{dim} - 1;
D1 = dxInv * (gdata(indices1{:}) - gdata(indices2{:}));

% Second divided differences (first entry corresponds to D^2_0).
indices1{dim} = 2 : size(D1, dim);
indices2{dim} = indices1{dim} - 1;
D2 = 0.5 * dxInv * (D1(indices1{:}) - D1(indices2{:}));

%---------------------------------------------------------------------------
% First divided difference array has an extra entry at top and bottom
%   (from stencil width 2), so strip them off.
% Now first entry corresponds to D^1_{1/2}.
indices1{dim} = 2 : size(D1, dim) - 1;
D1 = D1(indices1{:});

%---------------------------------------------------------------------------
% First order approx is just the first order divided differences.
%   Make two copies to build the two approximations
dL = cell(2,1);
dR = cell(2,1);

% Take leftmost grid.N(dim) entries for left approximation.
indices1{dim} = 1 : size(D1, dim) - 1;
[ dL{:} ] = deal(D1(indices1{:}));

% Take rightmost grid.N(dim) entries for right approximation.
indices1{dim} = 2 : size(D1, dim);
[ dR{:} ] = deal(D1(indices1{:}));
```

```
%--------------------------------------------------------------------------
% Each copy gets modified by one of the second order terms.
%   Second order terms are sorted left to right.
indices1{dim} = 1 : size(D2, dim) - 2;
indices2{dim} = 2 : size(D2, dim) - 1;
dL{1} = dL{1} + grid.dx(dim) * D2(indices1{:});
dL{2} = dL{2} + grid.dx(dim) * D2(indices2{:});

indices1{dim} = indices1{dim} + 1;
indices2{dim} = indices2{dim} + 1;
dR{1} = dR{1} - grid.dx(dim) * D2(indices1{:});
dR{2} = dR{2} - grid.dx(dim) * D2(indices2{:});

%--------------------------------------------------------------------------
if(generateAll)

    if(checkEquivalentApproximations)
        % Rightward left and leftward right approximations should be the same
        % (should be centered approximations, but we don't check for that).
        checkEquivalentApprox(dL{2}, dR{1}, small);
    end

    % Caller requested both approximations in each direction.
    derivL = dL;
    derivR = dR;

    %----------------------------------------------------------------------
else

    % Need to figure out which approximation has the least oscillation.
    %   Note that L and R in this section refer to neighboring divided
    %   difference entries, not to left and right approximations.

    % Pick out minimum modulus neighboring D2 term.
    D2abs = abs(D2);
    indices1{dim} = 1 : size(D2, dim) - 1;
    indices2{dim} = indices1{dim} + 1;
    smallerL = (D2abs(indices1{:}) < D2abs(indices2{:}));
    smallerR = ~smallerL;

    %----------------------------------------------------------------------
    % Pick out second order approximation that used the minimum modulus D2
    % term.
    indices1{dim} = 1 : size(smallerL, dim) - 1;
    derivL = dL{1} .* smallerL(indices1{:}) + ...
        dL{2} .* smallerR(indices1{:});

    indices1{dim} = 2 : size(smallerL, dim);
```

```
    derivR = dR{1} .* smallerL(indices1{:}) + ...
        dR{2} .* smallerR(indices1{:});

end
```

Appendix D

Algorithm for Dihedral Angle Measurement

The code below measures the apparent dihedral angles on solid-solid-liquid contact lines in two-dimensional sections. The code is designed to cut through the sample in x, y and z planes and move through the slices and gather information on apparent dihedral angle. Once the data acquisition is done, a Gaussian fit is applied to data to calculate the value of dihedral angle.

```
%% Author: Soheil Ghanbarzadeh
%% Date: 05/17/2016
% Description:
% This script finds the corner in x, y, z slices of a 3-D segmented image.
% It then fits a number of arrows to the solid-liquid interface, with
% arrows starting from the found corners. It then measures the mean angle
% between the arrows, or basically the angle between two interfaces
% intersecting on the corner. Distribution of the apparent dihedral angle
% in all slices and in all corners are studied to determine the median of
% the dihedral angle
            % Input:
            % filename = file name, segmented image. Code assumes .raw file
            %           with 0 and 255 values.
            % n = data size
            % nn = size of box around corners
            % QL = quality of corner detection
            % SF = safety faccor of corner detection

            % Output
            % theta = a vector containing all the appprenet dihedral angle
            %           values
            % Theta_median = median of the apparent dihedral angle


clear all; close all; clc;
filename = 'filename';                  % enter the file name
n = 500;                                % enter the data size
```

```
                                        % here I assume a cube with 500
                                        % pixel on each side
filesize = n^3;
fid = fopen([filename '.raw'], 'r');    % open the file. Assumption is that
                                        % the segmented image is saved in
                                        % .raw format
BW = fread(fid, filesize,'uint8');
fclose(fid);
BW = reshape(BW,[500 500 500]);         % reshaping to a 3-D matirx

BW = BW > 144;
phi = length(BW(BW(:)))...
    /length(BW(:));                    % calculating porosity
theta = zeros(1,1);                     % Initialize the dihedral angle

nn = 7;                                 % The size of box around corners
QL = 0.8;                               % Set quality of corner detection
SF = 0.15;                              % Set the safety faccor of corner
                                        % detection

cc = 0;                                 % Corner counter

%% main loop through x-plane
for i = 1:1:500

    BW2D = squeeze(BW(i,:,:));          % make 2-D slices
    BW2D = bwareaopen(BW2D, 50);        % filter out small particles
    BW2D = medfilt2(BW2D, [3 3]);       % median filter


    % detect corners
    % read MATLAB help for info
    C = corner(BW2D, 500, 'QualityLevel', QL, 'SensitivityFactor', SF);

    % calculating distabce fuction
    % required for solid-liquid boundary detection
    D1 = bwdist(BW2D,'euclidean');
    D2 = bwdist(~BW2D,'euclidean');
    D = D1-D2;

    % Extending the matrix to include the n-by-n box
    corner_mat_A = zeros(size(D)+2*nn);

    % making a hollow square matrix of ones and zeros (outside layer == 1)
    squaremat = ones(2*nn+1,2*nn+1);
    squaremat(nn-2:nn+4,nn-2:nn+4) = zeros(7,7);

    % get number of corner in each 2-D section
    [m, ~] = size(C);
```

```
    % replace hallow square for each corner in the matrix
    % this way we have a matrix, n+nn by n+nn, which has a number of
    % non-zero elements around the corner. corner is located in the center
    % of the square.
    for j = 1 : m
        corner_mat_A(C(j,1):C(j,1)+2*nn,C(j,2):C(j,2)+2*nn) =...
            j * squaremat;
    end

    % change the size back to n-by-n
    corner_mat = corner_mat_A(nn+1:end-nn,nn+1:end-nn);
    location = ((D==1) | D==-1) .* corner_mat';

    for j = 1 : m

        % update the overal corner number
        cc = cc + 1;

        % find the location of the corner j
        index = location == j;

        % find the physical location of corner j
        xcenter = C(j,1); ycenter = C(j,2);

        % find the row and column of all non-zero members of the square
        % around corner j
        [row, col] = find(index == 1);

        % draw a line from solid-liquid interface which falls in neighbor
        % cell of each corner (location matrix), calculate the angle of
        % each line and convert to degree.
        phi = atan2((row-ycenter),(col-xcenter));
        phi(phi<0) = phi(phi<0) + 2*pi;
        phi = phi * 180/pi;

        % devide the vectors to two groups. Upper and lower devisions, get
        % the mean of each group and the difference would be dihedral angle
        % in the corresponding corner (cc)
        avg = mean(phi);
        upper = mean(phi(phi>=avg));
        lower = mean(phi(phi<avg));
        theta(cc,1) = min((upper-lower),360-(upper-lower));
    end
end

%% main loop through y-plane
for i = 1:1:500
```

```
BW2D = squeeze(BW(:,i,:));          % make 2-D slices
BW2D = bwareaopen(BW2D, 50);        % filter out small particles
BW2D = medfilt2(BW2D, [3 3]);       % median filter


% detect corners
% read MATLAB help for info
C = corner(BW2D, 500, 'QualityLevel', QL, 'SensitivityFactor', SF);

% calculating distabce fuction
% required for solid-liquid boundary detection
D1 = bwdist(BW2D,'euclidean');
D2 = bwdist(~BW2D,'euclidean');
D = D1-D2;

% Extending the matrix to include the n-by-n box
corner_mat_A = zeros(size(D)+2*nn);

% making a hollow square matrix of ones and zeros (outside layer == 1)
squaremat = ones(2*nn+1,2*nn+1);
squaremat(nn-2:nn+4,nn-2:nn+4) = zeros(7,7);

% get number of corner in each 2-D section
[m, ~] = size(C);

% replace hallow square for each corner in the matrix
% this way we have a matrix, n+nn by n+nn, which has a number of
% non-zero elements around the corner. corner is located in the center
% of the square.
for j = 1 : m
    corner_mat_A(C(j,1):C(j,1)+2*nn,C(j,2):C(j,2)+2*nn) =...
        j * squaremat;
end

% change the size back to n-by-n
corner_mat = corner_mat_A(nn+1:end-nn,nn+1:end-nn);
location = ((D==1) | D==-1) .* corner_mat';

for j = 1 : m

    % update the overal corner number
    cc = cc + 1;

    % find the location of the corner j
    index = location == j;

    % find the physical location of corner j
    xcenter = C(j,1); ycenter = C(j,2);
```

```
        % find the row and column of all non-zero members of the square
        % around corner j
        [row, col] = find(index == 1);

        % draw a line from solid-liquid interface which falls in neighbor
        % cell of each corner (location matrix), calculate the angle of
        % each line and convert to degree.
        phi = atan2((row-ycenter),(col-xcenter));
        phi(phi<0) = phi(phi<0) + 2*pi;
        phi = phi * 180/pi;

        % devide the vectors to two groups. Upper and lower devisions, get
        % the mean of each group and the difference would be dihedral angle
        % in the corresponding corner (cc)
        avg = mean(phi);
        upper = mean(phi(phi>=avg));
        lower = mean(phi(phi<avg));
        theta(cc,1) = min((upper-lower),360-(upper-lower));
    end
end

%% main loop through z-plane
for i = 1:1:500

    BW2D = squeeze(BW(:,:,i));          % make 2-D slices
    BW2D = bwareaopen(BW2D, 50);        % filter out small particles
    BW2D = medfilt2(BW2D, [3 3]);       % median filter


    % detect corners
    % read MATLAB help for info
    C = corner(BW2D, 500, 'QualityLevel', QL, 'SensitivityFactor', SF);

    % calculating distabce fuction
    % required for solid-liquid boundary detection
    D1 = bwdist(BW2D,'euclidean');
    D2 = bwdist(~BW2D,'euclidean');
    D = D1-D2;

    % Extending the matrix to include the n-by-n box
    corner_mat_A = zeros(size(D)+2*nn);

    % making a hollow square matrix of ones and zeros (outside layer == 1)
    squaremat = ones(2*nn+1,2*nn+1);
    squaremat(nn-2:nn+4,nn-2:nn+4) = zeros(7,7);

    % get number of corner in each 2-D section
    [m, ~] = size(C);
```

```
    % replace hallow square for each corner in the matrix
    % this way we have a matrix, n+nn by n+nn, which has a number of
    % non-zero elements around the corner. corner is located in the center
    % of the square.
    for j = 1 : m
        corner_mat_A(C(j,1):C(j,1)+2*nn,C(j,2):C(j,2)+2*nn) =...
            j * squaremat;
    end

    % change the size back to n-by-n
    corner_mat = corner_mat_A(nn+1:end-nn,nn+1:end-nn);
    location = ((D==1) | D==-1) .* corner_mat';

    for j = 1 : m

        % update the overal corner number
        cc = cc + 1;

        % find the location of the corner j
        index = location == j;

        % find the physical location of corner j
        xcenter = C(j,1); ycenter = C(j,2);

        % find the row and column of all non-zero members of the square
        % around corner j
        [row, col] = find(index == 1);

        % draw a line from solid-liquid interface which falls in neighbor
        % cell of each corner (location matrix), calculate the angle of
        % each line and convert to degree.
        phi = atan2((row-ycenter),(col-xcenter));
        phi(phi<0) = phi(phi<0) + 2*pi;
        phi = phi * 180/pi;

        % devide the vectors to two groups. Upper and lower devisions, get
        % the mean of each group and the difference would be dihedral angle
        % in the corresponding corner (cc)
        avg = mean(phi);
        upper = mean(phi(phi>=avg));
        lower = mean(phi(phi<avg));
        theta(cc,1) = min((upper-lower),360-(upper-lower));
    end
end


%% Calculate the median of the apparent dihedral angle

% at this time, vecotr theta contains approximation of dihedral angle in
```

```
% each corner on all slices in x, y and z planes. We know can plot the
% result, fit a Guassian curve and calculate the median of apparent
% dihedral angle

figure(1)
set(gcf,'Units', 'Normalized', 'OuterPosition', [0 0 1 1]);
hold on
nbin = 50;
h = histogram(theta,nbin);
Y = h.Values;
XX = h.BinEdges;
X = linspace(min(XX),max(XX),nbin);
set(gca,'fontname', 'Helvetica', 'fontsize',36);
set(gca,'XLim',[0 180])
axis square
box on
set(gca,'Xtick',[0 45 90 135 180])
set(gca,'Ytick',[])
set(gca,'linewidth',2)


[xData, yData] = prepareCurveData( X, Y );
ft = fittype( 'gauss1' );
opts = fitoptions( 'Method', 'NonlinearLeastSquares' );
opts.Display = 'Off';
[fitresult, ~] = fit( xData, yData, ft, opts );
coeffvals = coeffvalues(fitresult);
Theta_median = coeffvals(2); display(Theta_median)
plot(linspace(0,180,100), fitresult(linspace(0,180,100)),'k','linewidth',4)

%% plot one slice

% choose a slice
% Here I am plotting slice 100 in z plane.

BW2D = squeeze(BW(:,:,100));          % make 2-D slices
BW2D = bwareaopen(BW2D, 50);        % filter out small particles
BW2D = medfilt2(BW2D, [3 3]);       % median filter


fig = figure(2);
set(fig,'Units', 'Normalized', 'OuterPosition', [0 0 1 1]);
imshow(BW2D,[0 1])
axis equal
hold on
cc = 0;
C = corner(BW2D, 500, 'QualityLevel', QL, 'SensitivityFactor', SF);

D1 = bwdist(BW2D,'euclidean');
```

```
D2 = bwdist(~BW2D,'euclidean');
D = D1-D2;

contour(D,[0 0],'LineWidth',2,'color','g');
plot(C(:,1), C(:,2), '.r','markersize',30);
box on

corner_mat_A = zeros(size(D)+2*nn);
squaremat = ones(2*nn+1,2*nn+1);
squaremat(nn-2:nn+4,nn-2:nn+4) = zeros(7,7);
[m, ~] = size(C);
for j = 1 : m
    corner_mat_A(C(j,1):C(j,1)+2*nn,C(j,2):C(j,2)+2*nn) = j * squaremat;
end

corner_mat = corner_mat_A(nn+1:end-nn,nn+1:end-nn);
location = ((D==1) | D==-1) .* corner_mat';

for j = 1 : m
    % update the overal corner number
    cc = cc + 1;

    % find the location of the corner j
    index = location == j;

    % find the physical location of corner j
    xcenter = C(j,1); ycenter = C(j,2);

    % find the row and column of all non-zero members of the square
    % around corner j
    [row, col] = find(index == 1);

    % draw a line from solid-liquid interface which falls in neighbor
    % cell of each corner (location matrix), calculate the angle of
    % each line and convert to degree.
    phi = atan2((row-ycenter),(col-xcenter));
    phi(phi<0) = phi(phi<0) + 2*pi;
    phi = phi * 180/pi;

    % devide the vectors to two groups. Upper and lower devisions, get
    % the mean of each group and the difference would be dihedral angle
    % in the corresponding corner (cc)
    avg = mean(phi);
    upper = mean(phi(phi>=avg));
    lower = mean(phi(phi<avg));
    theta(cc,1) = min((upper-lower),360-(upper-lower));


    edge_data(cc,1).v = sin(phi*pi/180);
```

```
    edge_data(cc,1).u = cos(phi*pi/180);
    edge_data(cc,1).x = xcenter*ones(size(phi));
    edge_data(cc,1).y = ycenter*ones(size(phi));
    quiver(edge_data(cc,1).x,edge_data(cc,1).y,edge_data(cc,1).u,...
        edge_data(cc,1).v,50,'color','r','linewidth',3)
end
```

Appendix E

Planetesimal-Scale Continuum Model

```
%% Authors: Soheil Ghanbarzadeh, Jake Jordan, Marc Hesse
%% Date: 03/01/2016
% Description:
% This script solves the dimensionless mechanics and enthalpy evolution in
% a planetesimals, assuming imcompressible three-phase system. The details
% of the model and the equations are presented in Chapter 7 of
% Ghanbarzadeh's dissertation. The model considers an evolving
% gravitational field, heat genereation with radiogenic heat source,
% hystresis in pore network topology, darcy flow of liquid melt in a
% deformable solid matrix and finally conservation of mass, momentum and
% thermal energy in the entire planetesimal.

% The code is solving for dimensionless parameters presented in
% Ghanbarzadeh's dissertation. The dimensionless parameter can easily be
% converted to dimensional parameters. Here is the list of parameters used
% in the code below.

            % Inputs:
            % theta = dihedral angle
            % Param.R = planetesimal radius
            % Param.rho.i = solid iron density
            % Param.rho.m = melt density [kg/m^3]
            % Param.rho.o = solid olivine density
            % Param.phi.i = initial solid iron content [% vol]
            % Param.phi.m = initial liquid iron content [% vol]
            % Param.phi.o = initial olivine content [%vol]
            % xi0 = Bulk solid viscosity [Pa.s]
            % mu = Fluid viscosity [Pa.s]
            % d = ave. grain size [m]
            % Param.u.G = Gravitational constant N m2/kg2
            % yrs2s = year to second
            % perm_scale = permeability scale, 'darcy' or 'lu'
            % grain_type = grain type for permeability and hystgresis
            %           calculations: 'regular or ''irregular'

            % Param.k.i = Thermal cond of solid iron [W/m K]
            % Param.k.m = Thermal cond of liquid iron [W/m K]
```

```
% Param.k.o = Thermal cond of olivine [W/m K]
% Param.cp.i = Heat Cap of solid iron [W/kg K]
% Param.cp.m = Heat Cap of liquid iron [W/kg K]
% Param.cp.o = Heat Cap of olivine [W/kg K]
% Param.h0.i = Reference enthalpy of formation - iron
% Param.h0.o = Reference enthalpy of formation - olivine
% Param.H.L = Latent heat of fusion J/kg - iron
% Param.H.tf = time of formation [yr]
% Param.H.tmax = Max time [yr]
% Param.H.init_26Al_27Al = initial 26Al/27Al ratio [-]
% Param.H.Lambda = 26Al Decay Constant [s^-1]
% Param.H.X_Al = Al content [%wt]
% Param.H.H0 = Heating decay [W/kg 26Al]
% Param.H.Tmelt = iron melting point [K]
% Param.H.Tmelt_o = olivine melting point [K]
% Param.H.T_surf = Surface temp [K]
% Param.H.T0 = Initial temp [K]


close all; clear all; clc;

theta = 90;                       % dihedral angle in degree [o]
Param.R = 50e3;                   % planetesimal radius
Param.rho.i = 8e3;                % solid iron density
Param.rho.m = 7e3;                % melt density [kg/m^3]
Param.rho.o = 2.6e3;              % solid olivine density
Param.phi.i = 20e-2;              % initial solid iron content [% vol]
Param.phi.m = 0;                  % initial liquid iron content [% vol]
Param.phi.o = 1 - Param.phi.i -...
    Param.phi.m;                  % initial olivine content [%vol]
xi0 = 1e19;                       % [Pa.s] Bulk solid viscosity
mu = 1;                           % [Pa.s] Fluid viscosity
d = 1e-3;                         % ave. grain size [m]
Param.u.G =  6.674e-11;           % Gravitational constant N m2/kg2
yrs2s = 60^2*24*365;              % year to second
perm_scale = 'darcy';             % permeability scale
grain_type = 'irregular';         % type of grains
Rm = Param.rho.m / Param.rho.i; % ratio of melt/iron density
Ro = Param.rho.o / Param.rho.i; % ratio of olivine/iron density


% function fit_perm provides the power law fits for the permeability
% values. the output is a series of functions that can easily be evaluated
% and interpolated for permeability calculations at different dihedral
% angle and porosity values.
perm_Funs = fit_perm (perm_scale, grain_type, d);


%% Heat parameters
```

```
Param.k.i = 79.5;                % Thermal cond of solid iron [W/m K]
Param.k.m = 34.6;                % Thermal cond of liquid iron [W/m K]
Param.k.o = 2.5;                 % Thermal cond of olivine [W/m K]
Param.cp.i = 400;                % Heat Cap of solid iron [W/kg K]
Param.cp.m = 400/6;              % Heat Cap of liquid iron [W/kg K]
Param.cp.o = 1e3;                % Heat Cap of olivine [W/kg K]
Param.h0.i = 0;                  % Reference enthalpy of formation - iron
Param.h0.o = 815e3;              % Reference enthalpy of formation - olivine
Param.H.L = 272e3;               % Latent heat of fusion J/kg - iron
Param.H.tf = 1.e6;               % time of formation [yr]
Param.H.tmax = 8e6;              % Max time [yr]
Param.H.init_26Al_27Al = 5e-5;   % initial 26Al/27Al ratio [-]
Param.H.Lambda = 3.0124e-14;     % 26Al Decay Constant [s^-1]
Param.H.X_Al = 1.5e-2;           % Al content [%wt]
Param.H.H0 = 0.355;              % Heating decay [W/kg 26Al]
Param.H.Tmelt = 988 + 273;       % iron melting point [K]
Param.H.Tmelt_o = 19000 + 273;   % olivine melting point [K]
Param.H.T_surf = 250;            % Surface temp [K]
Param.H.T0 = 250;                % Initial temp [K]

% difference between melting point and reference point. it is used in
% dimensionless enthalpy equations.
Param.H.DT0 = Param.H.Tmelt - Param.H.T0;

% radiogenic heat source per volume per time:
Param.H.A0 = Param.rho.o * Param.H.H0 * Param.H.X_Al *...
    Param.H.init_26Al_27Al/Param.phi.o;


%% Characteristic values
phi_c = Param.phi.i;                          % Characteristic porosity
[K_c, ~] = perm_TE_LS(perm_Funs,...           % Characteristic permeability
    [phi_c;phi_c], [theta;theta], [1;1]);
K_c = K_c(1);
xi_c = xi0/phi_c;                             % Characteristic viscosity
r_c = sqrt(K_c*xi_c/mu);                      % Characteristic length
rho_c = Param.rho.i;                          % Characteristic density
u_c = 4 * pi * rho_c * Param.u.G * r_c^2;     % Characteristic gravitational
                                              % potential
g_c = 4 * pi * rho_c * Param.u.G * r_c;       % Characteristic gravity
p_c = rho_c * g_c * r_c;                      % Characteristic pressure
v_c = r_c^2 * rho_c * g_c / xi_c;             % Characteristic velocity
U_c = r_c^3 * rho_c * g_c / xi_c;             % Characteristic solid velocity
                                              % potential
t_c = Param.rho.i * Param.cp.i * r_c^2 ...    % Characteristic time
/ Param.k.i;
t_c2Ma = t_c/31536000/1e6;                    % Characteristic time in
                                              % milions of years
H_c = Param.rho.i * Param.cp.i *...            % Characteristic enthalpy
```

```
    Param.H.DT0;
Param.H.A0_d = Param.H.A0 *...              % Characteristic radiogenic
    t_c / H_c;                             % heat source
Param.H.T_d_surf = (Param.H.T_surf -...    % Dimentionless surface
    Param.H.T0) / Param.H.DT0;             % tempretaure
Param.H.T_d_0 = (Param.H.T0 - Param.H.T0)...% Characteristic temprature
    / Param.H.DT0;                         % ratio
Param.H.Tmelt_d_o = (Param.H.Tmelt_o -...  % Characteristic enthalpy of
    Param.H.T0) / Param.H.DT0;             % olivine


%% Build grid and operators
% set min and max of grid
Grid.xmin = 0.0; Grid.xmax = Param.R/r_c;
% set number of grid points
Grid.Nx = 50;
% set geometry type
Grid.geom = 'spherical1D';
% build grid. see the function for more details.
Grid = build_grid(Grid);
% build operators. see the function for more details.
[D,G,I] = build_ops(Grid); LAP = D*G;


%% Time evolution parmeters
% dim-less start time and end time
t_d = Param.H.tf * yrs2s / t_c;
t_d_max = Param.H.tmax * yrs2s / t_c;
% cfl for stability of explicit solution
cfl = 0.5; i = 1;
% Enthalpy timestep
maxdiff = 1; dt_d_H = Grid.dx^2/2/maxdiff;


%% Initial Condition
% uniform distribution of materials throughout the planet.
phi_i = Param.phi.i * ones(Grid.Nx,1);
phi_m = Param.phi.m * ones(Grid.Nx,1);
% uniform dihedral angle thoughout the planetesimal. Note that the dihedral
% angle can change locally.
theta = linspace(theta,theta,Grid.N)';
% melt is initially considered disconnected
connectivity = false(Grid.N,1);
phi_o = 1 - phi_i - phi_m;
% initial enthalpy is evaluated at surface temp (initial temp) and initial
% material distibution (phi_i, phi_m, phi_o)
% see the funcation enthalpy_d for more details
H_d = enthalpy_d(phi_i, phi_m, Param.H.T_d_0 * ones(size(phi_o)), Param);
% initial dimless temperature is calculated from dimless enthalpy
```

```
% see the function T_dfun for more details.
T_d = T_dfun(H_d, phi_o, phi_m, phi_i, Param);
% initial melt velocity is set to zero
v_d = zeros(size(Grid.xf)); v_d_m = zeros(size(Grid.xf));


%% Specify boundary conditions
% the boundary conditions in here are either set naturally (due to natural
% boundary conditions in spherical coordinate system) or specified with
% their type (neumann or drichlet), their degree of freedom and their
% direction.
% see build_bnd for more information how to set boundary conditions.


% Overpressure - pure Neumann
Param.p.dof_dir = [];
Param.p.dof_f_dir = [];
Param.p.dof_neu = [Grid.dof_xmax];
Param.p.dof_f_neu =[Grid.dof_f_xmax];
Param.p.g = 0;


% Solid velocity potential
Param.v.dof_dir   = Grid.dof_xmax;
Param.v.dof_f_dir = Grid.dof_f_xmax;
Param.v.dof_neu   = [Grid.dof_xmin];
Param.v.dof_f_neu = [Grid.dof_f_xmin];
Param.v.qb = 0;
Param.v.g = 0;


% Gravitational Potential
Param.u.dof_dir = Grid.dof_xmax;
Param.u.dof_f_dir = Grid.dof_f_xmax;
Param.u.dof_neu = [];
Param.u.dof_f_neu = [];
Param.u.g = 0;


% Enthalpy Equation
Param.H.dof_dir = Grid.dof_xmax;
Param.H.dof_f_dir = Grid.dof_f_xmax;
Param.H.dof_neu = [];
Param.H.dof_f_neu = [];
Param.H.g = enthalpy_d(Param.phi.i, Param.phi.m, Param.H.T_d_surf, Param);


%% Initilize plot
figure(1);
```

```
set(gcf, 'Units', 'Normalized', 'OuterPosition', [0 0 1 1]);
set(gca,'fontsize',18)

figure(2);
set(gcf, 'Units', 'Normalized', 'OuterPosition', [0 0 1 1]);
set(gca,'fontsize',18)

%% Solve time-dependent system

while t_d < t_d_max

    %% Update t_d
    dt_d_v =  min(abs(Grid.dx * (r_c / (v_c * t_c)) ./ v_d));
    % choose between porosity evolution and enthalpy evolution boundary
    % conditions.
    dt_d = 0.5 * cfl * min(dt_d_H, dt_d_v);
    t_d = t_d + dt_d;


    %% Olivine Mass balance
    % upwinding solid velocity
    A = flux_upwind(v_d,Grid);
    % upwinding melt velocity
    Avm = flux_upwind(v_d_m,Grid);
    % updating olivine volume fraction
    dphi_o = - (v_c * t_c / r_c) * dt_d * D * A * phi_o;
    phi_o = phi_o + dphi_o;
    % update phi_m and phi_i with current enthalpy and updated phi_o
    [phi_m, phi_i] = phifun (H_d, phi_o, Param);

    %% Enthalpy Equation
    % set the boundary condition
    [B,N,fn] = build_bnd(Param.H, Grid);
    % calculate effective conductivity and moving it to a matrix
    k_d = phi_i + (Param.k.m/Param.k.i) * phi_m + ...
        (Param.k.o/Param.k.i) * phi_o;
    kd_d = comp_mean(k_d,-1,1,Grid); kd_d(1,1) = kd_d(2,2);
    kd_d(end,end) = kd_d(end-1,end-1);

    % dimensionless specific phase enthalpies from current T_d
    h_d_i = T_d;
    h_d_m = T_d + Param.H.L / (Param.cp.i * Param.H.DT0);
    h_d_o = T_d * (Param.cp.o / Param.cp.i) + (Param.h0.o -...
        Param.h0.i)/(Param.cp.i * Param.H.DT0);

    % radiogenic source term
    Gamma_d = phi_o * Param.H.A0_d * exp (-Param.H.Lambda * t_c...
        * (t_d-dt_d/2));
    fs =  Gamma_d + D * (kd_d*G*T_d) - (v_c * t_c / r_c) * D * ...
```

```
    ((Param.rho.o / Param.rho.i) * A * (phi_o .* h_d_o) +...
    A * (phi_i .* h_d_i) + ...
    (Param.rho.m / Param.rho.i) * Avm * (phi_m .* h_d_m));

% Update dimensionless enthalpy, temprature and porosities
H_d = solve_lbvp(I,dt_d*(fs+fn)+H_d,B,Param.H.g,N);
[phi_m, phi_i] = phifun (H_d, phi_o, Param);
T_d = T_dfun(H_d, phi_o, phi_m, phi_i, Param);

% check if olivine is melting
% here we consider olivine never melts, that's why melting point of
% olivine is considered 20,000K!
if max(T_d) > Param.H.Tmelt_d_o
    display('Olivine is melting! Stopping simulation...')
    break;
end


%% Update Xid_d, density and permeability
% update dimensionless bulk solid viscosity
Xi_d_inv = (xi_c)./(xi0./phi_m);
Xid_d_inv = spdiags(Xi_d_inv,0,Grid.N,Grid.N);

% update dimensionless density
rho_d = phi_i + Rm * phi_m + Ro * phi_o;

% compute dimensionless permeability
[K, connectivity] = perm_TE_LS(perm_Funs, phi_m, theta, connectivity);
K_d = K/K_c;
Kd_d = comp_mean(K_d,-1,1,Grid); Kd_d(1,1) = 1; Kd_d(end,end) = 1;


%% Solve for Gravitational Field
% see the function for more details
[~,gvec_d] = gravity_d(D, G, Grid, Param.u, rho_d);


%% Solve for overpressure and relative melt flux
% Discrete operators
Lp = -D*(Kd_d*G) + Xid_d_inv;
% right hand side
drho_d = (phi_i + Ro * phi_o) ./ (phi_i + phi_o) - Rm;
Drho_d = comp_mean(drho_d,-1,1,Grid); Drho_d(1,1)=Drho_d(2,2);
Drho_d(end,end)=Drho_d(end-1,end-1);
fs_p = D * (Kd_d * Drho_d * gvec_d);
% Update overpressure BC at surface
Param.p.qb = - gvec_d(end) * drho_d(end);
% Build boundary conditions
[Bp,Np,fn_p] = build_bnd(Param.p,Grid);
```

```
% Solve linear system in places where phi_m is not zero
% if we solve the whole system, the matrix would be singular
if any(phi_m)
    ind = find(phi_m~=0,1,'last');
    p_d(1:ind,1) = solve_lbvp(Lp(1:ind,1:ind),fs_p(1:ind)+...
        fn_p(1:ind),Bp,Param.p.g,Np);
    p_d(ind+1:end,1) = 0;
else
    p_d = zeros(size(phi_m));
end


%% Compute relative melt flux
% here we compute dimensionless melt flux q_r_d
% the residual is also calculated to check if the answer is
% correct and mass is conserved
[qr_d, res_q_d] = comp_relative_melt_flux_d(D,Kd_d,Drho_d,...
    Xid_d_inv,gvec_d,G,p_d,fs_p,Grid,Param.p);


%% Solve for solid velocity and potential
% Discrete operators
Lv = LAP;
fs_v = Xid_d_inv*p_d;
if abs(sum(fs_v .* Grid.V))/Grid.N > 1e-6;
    error('Compatability condition violated.'); end
% Build boundary conditions
[Bv,Nv,fn_v] = build_bnd(Param.v,Grid);
% Solve linear system
U_d = solve_lbvp(Lv,fs_v+fn_v,Bv,Param.v.g,Nv);
% obtain the solid velocity as gradient solid potential
v_d = G * U_d;


%% Plot Dimensionless Parameters
if mod(i,100) == 0
    figure(1)
    suptitle(['Dimensionless Time = ', num2str(t_d)])

    % dim-less gravity vs dim-less length
    subplot 181
    plot(gvec_d,Grid.xf,'linewidth',2), xlabel 'g', ylabel 'r_d [-]';

    % porosities vs dim-less length
    subplot 182
    plot(phi_i, Grid.xc, phi_o, Grid.xc, phi_m, Grid.xc,...
        'linewidth',2), xlabel '\phi';
    set(gca, 'xlim',[0 1])
```

```
    % connectivity of pore space vs dim-less length
    subplot 183
    plot(connectivity,Grid.xc,'linewidth',2), xlabel 'connectivity'

    % dim-less permeability vs dim-less length
    subplot 184
    plot(K_d,Grid.xc,'linewidth',2), xlabel 'k'
    set(gca, 'xscale','log');

    % dim-less over-pressure vs dim-less length
    subplot 185
    plot(p_d,Grid.xc,'linewidth',2), xlabel 'p'

    % dim-less solid velocity vs dim-less length
    subplot 186
    plot(v_d,Grid.xf,qr_d,Grid.xf,'linewidth',2), xlabel ' v_s'

    % dim-less enthalpy vs dim-less length
    subplot 187
    plot(H_d, Grid.xc,'linewidth',2), xlabel 'H',

    % dim-less temprature vs dim-less length
    subplot 188
    plot(T_d, Grid.xc,'linewidth',2), hold on;
    plot(ones(size(Grid.xc)), Grid.xc,'r','linewidth',2)
    hold off, xlabel 'T';

    drawnow
end


%% Plot Dimensional parameters

if mod(i,100) == 0
    figure(2)
    suptitle(['Time = ', num2str(t_d*t_c2Ma) 'Myr'])

    % gravity vs radial distance from center of planetesimal
    subplot 181
    plot(g_c * gvec_d, Grid.xf * r_c/1000,'linewidth',3)
    xlabel 'g', ylabel 'r [km]';
    set(gca,'fontsize',18), set(gca,'linewidth',1)

    % porosities vs radial distance from center of planetesimal
    subplot 182
    plot(phi_i, Grid.xc * r_c/1000, phi_o, Grid.xc * r_c/1000,...
        phi_m, Grid.xc * r_c/1000, 'linewidth',3), xlabel '\phi';
    set(gca, 'xlim',[0 1]); set(gca,'Ytick',[])
    set(gca,'fontsize',18), set(gca,'linewidth',1)
```

```
% connectivity of melt vs radial distance from center
%of planetesimal
subplot 183
plot(connectivity, Grid.xc * r_c/1000,'linewidth',3),
xlabel 'connectivity'
set(gca,'Ytick',[]); set(gca,'fontsize',18)
set(gca,'linewidth',1); set(gca, 'xlim',[-.2 1.2])
set(gca,'Xtick',[0 1]);

% permeability vs radial distance from center of planetesimal
subplot 184
plot(K_d * K_c, Grid.xc * r_c/1000,'linewidth',3), xlabel 'k'
set(gca,'Ytick',[]); set(gca,'fontsize',14)
set(gca,'linewidth',1); set(gca, 'xlim',[0 1e-8]);

% over-pressure vs radial distance from center of planetesimal
subplot 185
plot(p_d * p_c, Grid.xc * r_c/1000,'linewidth',3), xlabel 'p'
set(gca,'Ytick',[]); set(gca,'fontsize',18)
set(gca,'linewidth',1)

% relative melt flux vs radial distance from center of
% planetesimal
qr_d(abs(qr_d)<1e-15) = 0;
subplot 186
plot(v_d * v_c, Grid.xf * r_c/1000, qr_d * v_c,...
    Grid.xf * r_c/1000,'linewidth',3), xlabel ' v'
set(gca,'Ytick',[]); set(gca,'fontsize',18)
set(gca,'linewidth',1), set(gca,'Xlim',[-5e-10 5e-10]);

% total specific enthalpy vs radial distance from center
% of planetesimal
subplot 187
plot(H_d * (Param.cp.i * Param.H.DT0), Grid.xc * r_c/1000,...
    'linewidth',3), xlabel 'H',
set(gca,'Ytick',[]); set(gca,'fontsize',18)
set(gca,'linewidth',1)

% temprature vs radial distance from center of planetesimal
% melting point of iron is shown with red color
subplot 188
plot(T_d * Param.H.DT0 + Param.H.T0, Grid.xc * r_c/1000,...
    'linewidth',3), hold on;
plot(Param.H.Tmelt * ones(size(Grid.xc)), Grid.xc * ...
    r_c/1000,'r','linewidth',2), hold off, xlabel 'T';
set(gca,'Ytick',[]); set(gca,'fontsize',18),
set(gca,'linewidth',1)
```

```
        drawnow
    end

    i = i+1;
end



%% Author: Soheil Ghanbarzadeh
%% Date: 03/01/2016
% Description:
% This function contains the values of permeability for the network of
% regular and irregualr grains at different porosity and dihedral angles.
% It fits power-type curves (k = a * phi ^ n) to the data for each dihedral
% angle and outputs the fit functions. The function also outputs two fit
% curve for percolation and trapping threshold in percolation map. It is
% useful in determining the connectivity of the pore network for a given
% porosity and dihedral angle.

% Fit functions are evaluated in simulations, to calculate the permeability
% in appropriate units and also check wether the fluid is connected or
% trapped.


            % Inputs:
            % scale = the desired scale for permeability
            %         it can be in darcy or lattice units
            % grain_type = type of grains to be considered: regular
            %              truncated octahedron grains or irregular grains
            % d = average grain size in m (usually 1e-3 is a good choice)


            % Output: F
            % F.theta10 = power-law function of phi-k for theta = 10
            % F.theta30 = power-law function of phi-k for theta = 30
            % F.theta60 = power-law function of phi-k for theta = 60
            % F.theta70 = power-law function of phi-k for theta = 70
            % F.theta90 = power-law function of phi-k for theta = 90
            % F.theta05 = power-law function of phi-k for theta = 105
            % F.theta120 = power-law function of phi-k for theta = 120
            % F.trap = curve-fit function for trapping threshold
            % F.perc = curve-fit function for percolation threshold

function F = fit_perm (scale, grain_type, d)

% check the scale and calculate appropriate factor for grain size and unit
% of the permeability
switch scale
    case 'darcy'
```

```
        % the factors come from the average grain size and simple scaling
        scale_factor_RG = (0.4*1.4*1e-6) * (1000/55) * (1000 * d);
        scale_factor_TOH = 5.07e-6 * (1000 * d);
    case 'lu'
        % factor only a function of grain size as the original values are
        % already in lattice units.
        scale_factor_RG = 1 * (1000 * d);
        scale_factor_TOH = 1 * (1000 * d);
end


% scaling the calculated permeabilty for irregular grains
k_RG = scale_factor_RG^2 * [1.22676E-05 6.4834E-05  0.000355273...
    0.001008603 0.002200075 0.004393044 0.007038431 0.010766166...
    0.015645963 0.023328705 0.030696179 0.039311622 0.062640629...
    0.093359956 0.126867789 0.166016877 0.268038345 0.382055021...
    0.532844177 0.744212358 1.019796061 1.340343288 1.741105384...
    2.356262493 3.018230214 3.814375765;
1.89157E-05 0.000110376 0.000554669 0.00139879  0.00295678...
    0.00538238  0.00991287  0.0152039   0.0218547   0.0300889...
    0.0399974   0.0515769   0.07767 0.112751    0.155   0.204248...
    0.321827    0.461727    0.634954    0.864714    1.16798 1.55732...
    2.04936 2.65774 3.39994 4.29371;
3.10508E-05 0.000168618 0.000771492 0.001953219 0.004027511...
    0.007144772 0.011148843 0.017417445 0.025802037 0.034715688...
    0.044903194 0.058797108 0.091924141 0.136049735 0.186566893...
    0.240409    0.394149    0.589058    0.824539    1.09774 1.511860164...
    2.035584519 2.671407528 3.364657247 4.141396391 5.237399576;
0   0.000225579 0.001056708 0.002364987 0.00461876  0.00932324...
    0.012937677 0.019488055 0.029230034 0.038575123 0.049212367...
    0.056077419 0.0823737   0.125329485 0.17455303  0.238116715...
    0.387100653 0.592880367 0.799445809 1.065532372 1.537976146...
    1.966807764 2.600594091 3.6682  4.23295 5.62066;
0   0   0.000598254 0.00365013  0.00678597  0.010544    0.0147378...
    0.0190026   0.0234539   0.0279235   0.0323668   0.0377834...
    0.0479186   0.0613504   0.0754416   0.0945069   0.14319 0.209742...
    0.31016 0.365642 0.632418   1.00342 1.54396 2.25194 3.13451 4.22243;
0   0   0   0.001145037 0.00217691  0.00357076  0.00483398  0.00614587...
    0.00799036  0.00947761  0.01165729  0.01417898  0.01981568...
    0.0267757   0.0343718   0.0431613   0.0693305   0.111434    0.160881...
    0.215904    0.278668    0.329798    0.365455    0.457907    0.551743...
    0.896827;
0   0   0   0.000827317 0.00145374  0.00197424  0.00293595  0.00391217...
0.00551303  0.00657892  0.00761507  0.00892913  0.0114677   0.014633...
    0.0179478   0.0219445   0.0315457   0.0446616   0.0585432...
    0.0758741   0.0911693   0.102354    0.123056    0.145986...
    0.194465    0.235073];
```

```
% porosity of the irregular grain network
phi_RG = 0.01 * [0.451  0.833   1.293   1.748   2.211   2.749   3.125...
    3.579   4.044   4.597   5.061   5.528   6.464   7.405   8.285   9.234...
    11.146  13.072  15.003  16.933  18.861  20.791  22.721  24.649...
    26.575  28.498;
0.419   0.853   1.351   1.757   2.214   2.665   3.191   3.655   4.117...
    4.582   5.048   5.516   6.409   7.351   8.298   9.247   11.161...
    13.097  15.033  16.967  18.896  20.818  22.741  24.662  26.584  28.505;
0.422   0.863   1.331   1.789   2.249   2.687   3.147   3.609   4.071...
    4.533   4.997   5.46    6.399   7.336   8.281   9.227   11.134...
    13.057  14.998  16.965  18.942  20.904  22.826  24.715  26.602  28.566;
0.413   0.841   1.269   1.704   2.137   2.657   3.02    3.461   3.905...
    4.349   4.793   5.088   5.792   6.67    7.545   8.424   10.224...
    12.048  13.922  15.784  17.665  19.56   21.437  23.731  25.257  27.571;
0.469   0.769   1.349   1.869   2.348   2.819   3.292   3.757   4.227...
    4.701   5.179   5.668   6.601   7.542   8.493   9.499   11.393...
    13.287  15.086  16.049  17.856  19.69   21.496  23.323  25.14   26.936;
0.484   0.96    1.444   1.873   2.264   2.741   3.212   3.685   4.216...
    4.688   5.165   5.654   6.58    7.51    8.452   9.424   11.318...
    13.188  15.047  16.934  18.8    20.53   21.312  23.136  24.972  26.8;
0.435   0.903   1.452   1.872   2.34    2.806   3.334   3.761   4.173...
    4.636   5.1 5.587   6.495   7.402   8.321   9.351   11.134  12.985...
    14.794  16.598  18.396  20.164  21.722  22.255  24.034  25.792];


% scaling the calculated permeabilty for regular grains
k_TOH = scale_factor_TOH^2 * [0.0001001 0.000152552 0.00074301...
    0.00145755  0.00314389  0.00588577  0.00846031  0.0117285...
    0.015997    0.0224598   0.0287197   0.0381741   0.0560078...
    0.0807439   0.109   0.14483 0.237031    0.351389    0.502918...
    0.667989    0.884574    1.1469  1.42098 1.74817 2.1276  2.57873;
0.00014382  0.000292658 0.000819684 0.00174862  0.00357323  0.00612459...
    0.00956462  0.0126419   0.0168994   0.0234189   0.0311664   0.039938...
    0.0584232   0.0877298   0.118572    0.161001    0.250298    0.387549...
    0.543871    0.737217    0.971655    1.23696 1.55989 1.90094 2.3058...
    2.73754;
0.00020312  0.000380871 0.000897062 0.00190157  0.00401133  0.00764725...
    0.0110562   0.0155362   0.0215349   0.0315348   0.0392548...
    0.0485963   0.0692595   0.102737    0.140593    0.190941...
    0.301016    0.458889    0.631674    0.859089    1.11025 1.4217...
    1.78942 2.16091 2.61742 3.13389;
0   0.000239283 0.00135791  0.00314842  0.0025044   0.00857113...
    0.0148821   0.0232932   0.0343663   0.0445282   0.0537998...
    0.0687731   0.0994653   0.130214    0.170001    0.202702...
    0.305518    0.449403    0.622774    0.853677    1.12932...
    1.46716 1.93511 2.45249 3.04648 3.62042;
0   0   0.00181882  0.0101504   0.0210141   0.0355998   0.0481045...
    0.0678561   0.0876696   0.117216    0.145395    0.168993...
    0.224234    0.310375    0.40882 0.511669    0.68977 0.803795...
```

```
    1.03523 1.35953 1.73262 2.17418 2.66556 3.28424 4.25919 5.18886;
0   0   0   0.0102045   0.0184779   0.0416518   0.0589981...
    0.0856041   0.107748    0.134276    0.165915    0.193108...
    0.260177    0.343068    0.48956 0.635666    0.800757    1.04308...
    1.1686  1.44352 1.83941 2.32152 2.89365 3.59644 4.63495 6.391;
0   0   0   0   0   0.00474149  0.0236574   0.0857512   0.120767...
    0.155766    0.194907    0.23846 0.299972    0.429728    0.673184...
    0.872373    0.938573    1.53573 1.92326 2.76955 3.53313 4.56106...
    6.02498 7.46901 8.0439  9.68519];


% porosity of the regular network
phi_TOH = 0.01 * [0.49  0.811   1.261   1.864   2.353   2.628   3.126...
    3.589   4.05    4.597   5.061   5.61    6.575   7.531   8.504...
    9.447   11.438  13.334  15.391  17.239  19.218  21.238  23.17...
    25.076  27.033  29.006;
0.484   0.848   1.275   1.762   2.247   2.679   3.198   3.647   4.107...
    4.645   5.139   5.682   6.564   7.61    8.553   9.584   11.404...
    13.458  15.371  17.323  19.265  21.223  23.244  25.149  27.135...
    29.005;
0.485   0.968   1.446   1.919   2.443   2.942   3.416   3.878   4.36...
    4.958   5.409   5.876   6.777   7.83    8.783   9.772   11.687...
    13.742  15.641  17.631  19.588  21.514  23.538  25.421  27.345  29.393;
0.264   0.72    1.275   1.77    2.292   2.867   3.273   3.831   4.445...
    4.954   5.452   6.061   7.061   7.905   8.886   9.538   11.305...
    13.225  15.076  17.081  19.176  21.304  23.434  25.526  27.734  29.765;
0.775   1.591   2.158   3.107   3.887   4.654   5.351   6.115   6.77...
    7.507   8.216   8.881   10.129  11.435  12.756  13.892  15.723...
    16.709  18.475  20.604  22.636  24.713  26.671  28.815  31.128  33.037;
0.607   1.304   1.991   3.299   3.809   4.879   5.616   6.43    7.128...
    7.841   8.513   9.103   10.361  11.613  12.906  14.166  16.461...
    18.308  19.137  20.883  22.889  25.046  27.117  29.45   32.104  34.017;
0.514   1.532   2.19    2.822   3.001   4.182   5.194   6.734   7.547...
    8.249   9.024   9.782   10.871  12.387  14.4    15.85   18.745...
    21.496  23.914  26.158  29.387  32.435  34.467  36.214  37.296  39.095];


% the best way to fit a power law in matlab, is to get the log10 of the
% values in x- and y-axis and then fit a linear plot to the resulting
% values. The results from linear curves on log-log scale, must be powered
% by 10^x or 10^y in order to get back to original scale.

logkTOH = log10(k_TOH);
logphiTOH = log10(phi_TOH);
logkRG = log10(k_RG);
logphiRG = log10(phi_RG);

% as mentiones, the best fit is linear in log-log space.
ft = fittype( 'poly1' );
```

```
switch grain_type

    % create the permeability fit functions for the case of irregualr
    % grains.
    case 'irregular'
        % prepare the data for dihedral angle 10 and create the fit
        % function
        [xData, yData] = prepareCurveData(logphiRG(1,:), logkRG(1,:));
        [F.theta10, ~] = fit( xData, yData, ft, 'Exclude', yData == -Inf);

        % prepare the data for dihedral angle 30 and create the fit
        % function
        [xData, yData] = prepareCurveData(logphiRG(2,:), logkRG(2,:));
        [F.theta30, ~] = fit( xData, yData, ft, 'Exclude', yData == -Inf);

        % prepare the data for dihedral angle 60 and create the fit
        % function
        [xData, yData] = prepareCurveData(logphiRG(3,:), logkRG(3,:));
        [F.theta60, ~] = fit( xData, yData, ft, 'Exclude', yData == -Inf);

        % prepare the data for dihedral angle 70 and create the fit
        % function
        [xData, yData] = prepareCurveData(logphiRG(4,2:end),...
            logkRG(4,2:end));
        [F.theta70, ~] = fit( xData, yData, ft, 'Exclude', yData == -Inf);

        % prepare the data for dihedral angle 90 and create the fit
        % function
        [xData, yData] = prepareCurveData(logphiRG(5,3:end),...
            logkRG(5,3:end));
        [F.theta90, ~] = fit( xData, yData, ft, 'Exclude', yData == -Inf);

        % prepare the data for dihedral angle 105 and create the fit
        % function
        [xData, yData] = prepareCurveData(logphiRG(6,4:end),...
            logkRG(6,4:end));
        [F.theta105, ~] = fit( xData, yData, ft, 'Exclude', yData == -Inf);

        % prepare the data for dihedral angle 120 and create the fit
        % function
        [xData, yData] = prepareCurveData(logphiRG(7,4:end),...
            logkRG(7,4:end));
        [F.theta120, ~] = fit( xData, yData, ft, 'Exclude', yData == -Inf);


        % Percolation-trapping curves
        phi_perc = 0.01 * [0 (2.775+3.461)/2 (11.393+13.287)/2 ...
            (16.934+18.8)/2 (18.396+20.164)/2];
```

```
        phi_trap = 0.01 * [0 (0.413+0.841)/2 (0.769+1.349)/2 ...
            (0.96+1.444)/2 (1.452+1.872)/2];


    % create the permeability fit functions for the case of regualr
    % grains.
    case 'regular'
        % prepare the data for dihedral angle 10 and create the fit
        % function
        [xData, yData] = prepareCurveData(logphiTOH(1,:), logkTOH(1,:));
        [F.theta10, ~] = fit( xData, yData, ft, 'Exclude', yData == -Inf);

        % prepare the data for dihedral angle 30 and create the fit
        % function
        [xData, yData] = prepareCurveData(logphiTOH(2,:), logkTOH(2,:));
        [F.theta30, ~] = fit( xData, yData, ft, 'Exclude', yData == -Inf);

        % prepare the data for dihedral angle 60 and create the fit
        % function
        [xData, yData] = prepareCurveData(logphiTOH(3,:), logkTOH(3,:));
        [F.theta60, ~] = fit( xData, yData, ft, 'Exclude', yData == -Inf);

        % prepare the data for dihedral angle 70 and create the fit
        % function
        [xData, yData] = prepareCurveData(logphiTOH(4,:), logkTOH(4,:));
        [F.theta70, ~] = fit( xData, yData, ft, 'Exclude', yData == -Inf);

        % prepare the data for dihedral angle 90 and create the fit
        % function
        [xData, yData] = prepareCurveData(logphiTOH(5,:), logkTOH(5,:));
        [F.theta90, ~] = fit( xData, yData, ft, 'Exclude', yData == -Inf);

        % prepare the data for dihedral angle 105 and create the fit
        % function
        [xData, yData] = prepareCurveData(logphiTOH(6,:), logkTOH(6,:));
        [F.theta105, ~] = fit( xData, yData, ft, 'Exclude', yData == -Inf);

        % prepare the data for dihedral angle 120 and create the fit
        % function
        [xData, yData] = prepareCurveData(logphiTOH(7,:), logkTOH(7,:));
        [F.theta120, ~] = fit( xData, yData, ft, 'Exclude', yData == -Inf);

        % Percolation-trapping curves
        phi_perc = 0.01 * [0 (1.542+2.086)/2 (5.991+7.507)/2 ...
            (7.614+10.613)/2 (10.109+12.85)/2];
        phi_trap = 0.01 * [0 (0.264+0.72)/2 (1.591+2.158)/2 ...
            (1.991+3.299)/2 (3.001+4.182)/2];
end
```

```
% there is no percolation and trapping threshold below 60 degree
theta = [60 70 90 105 120];

% We need to smooth the dataset for percolation trapping lines
% smooth the data set
n = numel(phi_perc);
phi_perc_ref = interp1( 1:n, phi_perc, linspace(1, n, 10*n), 'PCHIP')';
theta_perc = interp1( phi_perc, theta, phi_perc_ref, 'PCHIP');
phi_trap_ref = interp1( 1:n, phi_trap, linspace(1, n, 10*n), 'PCHIP')';
theta_trap = interp1( phi_trap, theta, phi_trap_ref, 'PCHIP');

% Fit the curves
ft = fittype( 'poly1' );
[xData, yData] = prepareCurveData( theta_perc, phi_perc_ref );
[F.perc, ~] = fit( xData, yData, ft);

[xData, yData] = prepareCurveData( theta_trap, phi_trap_ref );
[F.trap, ~] = fit( xData, yData, ft);



%% Author: Soheil Ghanbarzadeh
%% Date: 03/01/2016
% Description:
% This function uses the curve fit functions from fit_perm to evaluate the
% permeability at appropriate porosities and dihedral angles. It first
% obtains the values of dihedral angle, then uses appropriate fit functions
% to evaluate or interpolate the permeability values at given porosties.


            % Inputs:
            % F = contains the fit curves from function fit_perm
            % conectivity = it is both input and output. the input part
            % serves the hystresis loop and has the history of the pore
            % network: wether it was connected to not



            % Outputs:
            % k = value of permeability, corresponding to porosity and
            % dihedral angle values.
            % connectivity = updates the connectivity of the system. if
            % not initially connected, and porosoity is increased above
            % percolation threshold, the connectivity becomes one. during
            % the driange, if porosity decreases below trapping threshold,
            % the connectivity becomes zero.

function [k, connectivity] = perm_TE_LS(F, phi, theta, connectivity)
```

```
% find places where theta is bigger than 60. other places don't have
% percolation threshold
index = theta > 60;

% locating the dihedral angle values in order to use appropriate fit
% function and interpolation
% in cases with theta < 10 we use the theta = 10 fit function, no
% interpolation
index10 = theta <= 10;

index30 = theta > 10 & theta <= 30;
index60 = theta > 30 & theta <= 60;
index70 = theta > 60 & theta <= 70;
index90 = theta > 70 & theta <= 90;
index105 = theta > 90 & theta <= 105;
index120 = theta > 105 & theta <= 120;

% in cases with theta > 120 we use the theta = 120 fit function, no
% interpolation
index120plus = theta > 120;

% locally save the value of previous connectivity
conn_past = connectivity(index);

% update connectivity based on current porosity and dihedral angle values
connectivity(index10) = 1;
connectivity(index30) = 1;
connectivity(index60) = 1;
conn = phi(index) >= F.perc(theta(index));

% include hystresis
conn(conn_past == 1) = 1;
% only disconnect the ones that have passed the trapping threshold
conn(phi(index) < F.trap(theta(index))) = 0;
connectivity(index) = conn;

% as the fit functiuons are still in log-log space, the porosity needs to
% be converted to log. At the end, permeability is converted back to linear
% values
philog = log10(phi);


%% Magic happens here.
% Simple evaluation of peremability in corresponding porosity and
% interpolation for theta


% log10 of permeability for grids with theta < 10
```

```
k(index10,1) = F.theta10(philog(index10));
% log10 of permeability for grids with theta > 10 and theta < 30
k(index30,1) = ((F.theta30(philog(index30)) - ...
    F.theta10(philog(index30)))/(30-10)) .* (theta(index30)-10) + ...
    F.theta10(philog(index30));
% log10 of permeability for grids with theta > 30 and theta < 60
k(index60,1) = ((F.theta60(philog(index60)) - ...
    F.theta30(philog(index60)))/(60-30)) .* (theta(index60)-30) + ...
    F.theta30(philog(index60));
% log10 of permeability for grids with theta > 60 and theta < 70
k(index70,1) = ((F.theta70(philog(index70)) - ...
    F.theta60(philog(index70)))/(70-60)) .* (theta(index70)-60) + ...
    F.theta60(philog(index70));
% log10 of permeability for grids with theta > 70 and theta < 90
k(index90,1) = ((F.theta90(philog(index90)) - ...
    F.theta70(philog(index90)))/(90-70)) .* (theta(index90)-70) + ...
    F.theta70(philog(index90));
% log10 of permeability for grids with theta > 90 and theta < 105
k(index105,1) = ((F.theta105(philog(index105)) - ...
    F.theta90(philog(index105)))/(105-90)) .* (theta(index105)-90) + ...
    F.theta90(philog(index105));
% log10 of permeability for grids with theta > 105 and theta < 120
k(index120,1) = ((F.theta120(philog(index120)) - ...
    F.theta105(philog(index120)))/(120-105)) .* (theta(index120)-105) + ...
    F.theta105(philog(index120));
% log10 of permeability for grids with theta > 120
k(index120plus,1) = F.theta120(philog(index120plus));


% changing the permeability back to linear values
k = 10.^k;
% just to make sure the permeability of disconnected network is zero.
k(~connectivity,1) = 0;
% just to make sure, where we have no melt, we have no permeability
k(phi==0) = 0;



function [H_d] = enthalpy_d (phi_i, phi_m, T_d, Param)

%% author: Soheil Ghanbarzadeh
%% Date: 03/01/2016
% Description:
% This functions uses the iron and melt volume fractions, as well as
% dimensionless temprature and calculates the dimesnionless enthalpy. It is
% only used for initial and boundary conditions. As in the code, the
% dimensionless enthalpy is evolved via conservation of energy.

            % Input:
```

```
            % phi_m = volume fraction of melt
            % phi_i = volume fraction of iron
            % T_d = dimensionless temperature
            % Param = structure contaning problem paramters and information
            %         about BC's
            %
            % Output:
            % H_d = dimensionless enthalpy

% dimensionless solid iron enthalpy at reference temp
h0d = Param.h0.i / (Param.cp.i * Param.H.DT0);
% initialize the phase enthalpies
h_d_i = zeros(size(phi_i)); h_d_m = h_d_i; h_d_o = h_d_i;



% look for places where no melt exists
index = T_d < 1;
% calculate phase enthalpy in these places
h_d_i(index) = T_d(index);
h_d_o(index) = T_d(index) * (Param.cp.o / Param.cp.i) +...
    (Param.h0.o - Param.h0.i)/(Param.cp.i * Param.H.DT0);



% look for places where melt and solid iron coexist
index = T_d == 1;
% calculate phase enthalpy in these places
h_d_i(index) = T_d(index);
h_d_o(index) = T_d(index) * (Param.cp.o / Param.cp.i) + ...
    (Param.h0.o - Param.h0.i)/(Param.cp.i * Param.H.DT0);
h_d_m(index) = T_d(index) + Param.H.L / (Param.cp.i * Param.H.DT0);



% look for places where no solid iron is left
index = T_d > 1;
% calculate phase enthalpy in these places
h_d_o(index) = T_d(index) * (Param.cp.o / Param.cp.i) + ...
    (Param.h0.o - Param.h0.i)/(Param.cp.i * Param.H.DT0);
h_d_m(index) = T_d(index) * (Param.cp.m / Param.cp.i) + ...
    Param.H.L / (Param.cp.i * Param.H.DT0) + (1 - Param.cp.m / Param.cp.i);


% the total enthalpy of system can be easily calculated. Look for
% dimensionless equations in Ghanbarzadeh's dissertation or Ghanbarzadeh
% et al 2016 supplementary materials.
H_d = phi_i .* (h_d_i + h0d) + ...
```

```
    phi_m .* (Param.rho.m/Param.rho.i) .* (h_d_m + h0d) + ...
    (1 - phi_i - phi_m) .* (Param.rho.o/Param.rho.i) ...
    .* (h_d_o + h0d) - h0d;



function [phi_m, phi_i] = phifun (H_d, phi_o, Param)
%% author: Soheil Ghanbarzadeh
%% Date: 03/01/2016
% Description:
% This function receives the dimensionless enthalpy from energy equation
% and olivine volume fraction from solid evolution equation. It then
% calculates the volume fractions of iron and melt. in places where H_d is
% not enough to melt the iron, phi_m is zero. and when H_d is large enough
% to melt all the avilable iron, phi_m is 1 - phi_o. In all other places,
% phi_m must be calculated, which is a linear function of H_d and a
% nonlinear function of phi_o.

            % Input:
            % H_d = dimensionless total specific enthalpy obtained from
            %       energy equation (enthalpy evolution equation)
            % phi_o = volume fraction of olivine. This is obtianed from
            %         solid olivine evolution equation
            % Param = structure contaning problem paramters and information
            %         about BC's
            %
            % Output:
            % phi_m = volume fraction of melt. Of course, when H_d is not
            % enough to melt the iron, phi_m is zero. and when H_d is large
            % enough to melt all the avilable iron, phi_m is 1 - phi_o. In
            % all other places, phi_m must be calculated, which is a linear
            % function of H_d.
            % phi_i = volume fraction of iron, simply 1 - phi_o - phi_m

%% initialize volume fraction of iron and melt
phi_m = zeros(size(phi_o));
phi_i = zeros(size(phi_o));

% dimensionless solid iron enthalpy at reference temp
h0d = Param.h0.i / (Param.cp.i * Param.H.DT0);
% dimensionless olivine phase enthalpy at reference temp
h0od = (Param.h0.o - Param.h0.i)/ (Param.cp.i * Param.H.DT0);
% specific heat capacity ratio: olivine to solid iron
Rcp_o = Param.cp.o / Param.cp.i;
% density ratio: olivine to solid iron
Rrho_o = Param.rho.o / Param.rho.i;
% density ratio: melt to solid iron
Rrho_m = Param.rho.m / Param.rho.i;
% dimensionless latent heat
```

```
Lstar = Param.H.L / (Param.cp.i * Param.H.DT0);

% maximum dimensionless enthalpy not enough to melt the iron
H_d_s = (1 - phi_o) .* (1 + h0d) + phi_o .* Rrho_o .* ...
    (Rcp_o + h0od + h0d) - h0d;
% minimum dimensionless enthalpy not enough to solidify the molten iron
H_d_m = (1 - phi_o) .* Rrho_m * (h0d + 1 + Lstar) + phi_o * Rrho_o * ...
    (Rcp_o + h0od + h0d);



% look for places where enthalpy is not enough to melt iron
index = H_d <= H_d_s;
% melt volume fraction in these places is zero then
phi_m(index) = 0;
% update iron volume fraction from zero value
phi_i(index) = 1 - phi_o(index);



% look for places where solid and molten iron coexist
index = (H_d < H_d_m) & (H_d > H_d_s);
% melt volume fraction is a linear function of dimensionless enthalpy and
% of course non-linear function of olivine melt fraction.
phi_m(index) = (H_d(index) + h0d - (1 - phi_o(index)) .*...
    (1 + h0d) - phi_o(index) * Rrho_o .* (h0od + Rcp_o + h0d)) ./...
    (Rrho_m * (1 + Lstar + h0d) - (1 + h0d));
% update iron volume fraction from initial zero values in these places
phi_i(index) = 1 - phi_m(index) - phi_o(index);



% look for places where enthalpy is larger to leave any solid iron
index = H_d >= H_d_m;
% in these places the volume fraction of solid iron is zero
phi_i(index) = 0;
% update the melt fraction in these places
phi_m(index) = 1 - phi_o(index);



function T_d = T_dfun(H_d, phi_o, phi_m, phi_i, Param)
%% author: Soheil Ghanbarzadeh
%% Date: 03/01/2016
% Description:
% This function calculates the dimensionless temprature from dimensionless
% enthalpy and volume fraction of phases. Dimensionless temprature is equal to
% one at iron melting point, below one at temp below melting point, and
% above one at temp above the melting point. The code detects the location
```

```
% where iron exists in either of: solid, liquid or solid-liquid phases and
% calculates dimensionless temp with appropriate properties.

            % Input:
            % H_d = dimensionless total specific enthalpy obtained from
            %       energy equation (enthalpy evolution equation)
            % phi_o = volume fraction of olivine
            % phi_m = volume fraction of melt
            % phi_i = volume fraction of iron
            % Param = structure contaning problem paramters and information
            %         about BC's
            %
            % Output:
            % T_d = dimensionless temprature

% dimensionless solid iron enthalpy at reference temp
h0d = Param.h0.i / (Param.cp.i * Param.H.DT0);
% dimensionless olivine phase enthalpy at reference temp
h0od = (Param.h0.o - Param.h0.i)/ (Param.cp.i * Param.H.DT0);
% specific heat capacity ratio: olivine to solid iron
Rcp_o = Param.cp.o / Param.cp.i;
% density ratio: olivine to solid iron
Rrho_o = Param.rho.o / Param.rho.i;
% specific heat capacity ratio: melt to solid iron
Rcp_m = Param.cp.m / Param.cp.i;
% density ratio: melt to solid iron
Rrho_m = Param.rho.m / Param.rho.i;
% dimensionless latent heat
Lstar = Param.H.L / (Param.cp.i * Param.H.DT0);
% initialize the output
T_d = zeros(size(phi_m));

% find the places where we don't have melt
index = phi_m == 0;
% we only have two phase: solid iron and solid olivine, the temprature can
% easily be calculated from enthalpy of the system
T_d(index) = (H_d(index) + h0d * (1 - Rrho_o) * phi_o(index) -...
    h0od * Rrho_o * phi_o(index)) ./...
    (1 - phi_o(index) + phi_o(index) * Rcp_o * Rrho_o);




% look for places where we have both molt and solid iron
index = (phi_m ~= 0) & (phi_i ~= 0);
% T_d in places where melt and iron coexist, is 1.
T_d(index) = 1;
```

```
% look for places where we don't have any solid iron left
index = (phi_m ~= 0) & (phi_i == 0);
% in these locations, the enthalpy has overcome latent heat. besides all
% iron is molten so we need to use the properties of liquid iron
T_d(index) = (H_d(index) - (1-phi_o(index)) * Rrho_m * (h0d ...
    + 1 - Rcp_m + Lstar) - phi_o(index) * Rrho_o * (h0d + h0od) ) ./ ...
    ((1-phi_o(index)) * Rrho_m * Rcp_m + phi_o(index) * Rcp_o * Rrho_o);



function [q_d,res_q_d] = comp_relative_melt_flux_d(D, Kd, Drho_d, ...
    Xid_d, gvec_d, G, p_d, fs, Grid,Param)

%% author: Soheil Ghanbarzadeh, Jake Jordan, Marc Hesse
%% Date: 03/01/2016
% Description:
% Computes the mass conservative fluxes across all boundaries from the
% residual of the compatability condition over the boundary cells. This
% special version of flux computation, computes relative melt flux (of
% course dimensionless) from darcy's equation. On the boundary, flux is
% computed using compaction equation (over pressure equation)

% Note: Current implmentation works for all cases where one face
%       is assigend to each bnd cell. So corner cells must have
%       natural BC's on all but one face.

            % Input:
            % D = N by Nf discrete divergence matrix.
            % Kd = Nf by Nf conductivity matrix.
            % Drho_d = dimensionless solid-liquid difference in density
            % Xid_d = dimensionless solid viscosity
            % gvec_d = dimensionless gravity
            % G = Nf by N discrete gradient matrix.
            % p_d = N by 1 vector of flow potential in cell centers.
            % fs = N by 1 right hand side vector containing only source
            %      terms.
            % Grid = structure containing grid information.
            % Param = structure contaning problem paramters and information
            %         about BC's
            %
            % Output:
            % q_d = Nf by 1 vector of fluxes across all cell faces
            % res_q_d = Nf by 1 vector of flux residual


%% Compute interior fluxes
q_d = -Kd*(G*p_d);
```

```
% Compute boundary fluxes
dof_cell = [Param.dof_dir;Param.dof_neu];
dof_face = [Param.dof_f_dir;Param.dof_f_neu];
sign = ismember(dof_face,[Grid.dof_f_xmin;Grid.dof_f_ymin])...
      -ismember(dof_face,[Grid.dof_f_xmax;Grid.dof_f_ymax]);
q_d(dof_face) =  sign.*( D(dof_cell,:) * q_d - fs(dof_cell) +...
    Xid_d(dof_cell,:)*p_d)...
    .*Grid.V(dof_cell)./Grid.A(dof_face);

% compute residual
res_q_d = D * q_d - fs + Xid_d*p_d;

% incorporate the buouyancy effect
q_d = q_d - Kd * Drho_d * gvec_d;



function [u,gvec_d] = gravity_d(D, G, Grid, Param, rho_d)

%% author: Soheil Ghanbarzadeh
%% Date: 03/01/2016
% Description:
% This function calculated the gravitational field in an evolving
% planetesimal. As the volume fraction of phases change with time, both due
% to melting of iron and draingae of melt toward the core, the density
% distribution inside the planetesimal also changes. Therefore, the
% graviational field needs to be evolving as well. See the euqations in
% Ghanbarzadeh's disertation or Ghanbarzadeh et al 2016.


            % Input:
            % D = N by Nf discrete divergence matrix.
            % G = Nf by N discrete gradient matrix.
            % Grid = structure containing grid information
            % Param = structure contaning problem paramters and information
            %         about BC's
            % rho_d = dimensionless density
            %
            % Output:
            % u = dimensionless graviational potential
            % g = dimensionless gravity field


% Laplacian
L = D*G;
% Right hand side
fs = rho_d;
% Build boundary conditions
```

```
[B,N,fn] = build_bnd(Param,Grid);
% Solve linear system
u = solve_lbvp(L,fs+fn,B,Param.g,N);
% Compute fluxes
gvec_d = comp_g(D,G,u,fs,Grid,Param);


% compatibility
if abs(sum(fs.* Grid.V) + gvec_d(Grid.Nfx).* Grid.A(Grid.Nfx))/Grid.N > 1e-8
    error ('Compatibility condition for gravitational field not satisfied')
end

% check g at center
if abs(gvec_d(1)) > 1e-8
    error ('Gravity at center is not zero')
end



function [B,N,fn] = build_bnd(Param,Grid)
% author: Marc Hesse
% date: 06/09/2015
% Description:
% This function computes the operators and r.h.s vectors for both Dirichlet
% and Neumann boundary conditions.
%
% Input:
% Grid = structure containing all pertinent information about the grid.
% Param = structure containing all information about the physical problem
%         in particular this function needs the fields
%         Param.dof_dir = Nc by 1 column vector containing
%                         the dof's of the Dirichlet boundary.
%         Param.dof_neu = N by 1 column vector containing
%                         the dof's of the Neumann boundary.
%         Param.qb      = column vector of prescribed fluxes on Neuman bnd.
%
% Output:
% B = Nc by N matrix of the Dirichlet constraints
% N = (N-Nc) by (N-Nc) matrix of the nullspace of B
% fn = N by 1 r.h.s. vector of Neuman contributions
%
% Example call:
% >> Grid.xmin = 0; Grid.xmax = 1; Grid.Nx = 10;
% >> Grid = build_grid(Grid);
% >> [D,G,I]=build_ops(Grid);
% >> Param.dof_dir = Grid.dof_xmin;  % identify Dirichlet bnd
% >> Param.dof_neu = Grid.dof_xmax;  % identify Neumann bnd
% >> Param.qb = 1;                   % set bnd flux
% >> [B,N,fn] = build_bnd(Param,Grid);
```

```
%% Dirichlet boundary conditions
if isempty(Param.dof_dir)
    B = [];
    N = [];
else
    N = speye(Grid.N); % use N as temp storage for identity
    B = N(Param.dof_dir,:);
    N(:,Param.dof_dir) = [];
end

%% Neumann boundary conditions
if isempty(Param.dof_neu)
    fn = spalloc(Grid.N,1,0);          % allocate sparse zero vector
else
    fn = spalloc(Grid.N,1,length(Param.dof_neu)); % allocate sparse vector
    fn(Param.dof_neu) = Param.qb.*...
        Grid.A(Param.dof_f_neu)./Grid.V(Param.dof_neu);
end



function [Grid] = build_grid(Grid)
% Author: Marc Hesse
% Date: 09/12/2014
% Description:
% This function computes takes in minimal definition of the computational
% domain and grid and computes all containing all pertinent information
% about the grid.
% Input:
% Grid.xmin = left boundary of the domain
% Grid.xmax = right bondary of the domain
% Grid.Nx   = number of grid cells
% Output: (suggestions)
% Grid.Lx = length of the domain
% Grid.dx = cell width
% Grid.xc = vector of cell center locations
% Grid.xf = vector of cell face locations
% Grid.Nfx = number of fluxes in x-direction
% Grid.dof_xmin = degrees of fredom corrsponding to
% the cells along the x-min boundary
% Grid.dof_xmax = degrees of fredom corrsponding to
% the cells along the x-max boundary
% Grid.dof_ymin = degrees of fredom corrsponding to
% the cells along the y-min boundary
% Grid.dof_ymax = degrees of fredom corrsponding to
% the cells along the y-max boundary
%
```

```
% Grid.dof_f_xmin = degrees of fredom corrsponding to
% the faces at the x-min boundary
% Grid.dof_f_xmax = degrees of fredom corrsponding to
% the faces at the x-max boundary
% Grid.dof_f_ymin = degrees of fredom corrsponding to
% the faces at the y-min boundary
% Grid.dof_f_ymax = degrees of fredom corrsponding to
% the faces at the y-max boundary
% Grid.psi_x0 = reference location for streamfunction
% Grid.psi_dir = diretion of integration for streamfunction
% + anything else you might find useful
%
% Example call:
% >> Grid.xmin = 0; Grid.xmax = 1; Grid.Nx = 10;
% >> Grid = build_grid(Grid);

%% Set up catesian geometry
if ~isfield(Grid,'geom'); Grid.geom = 'cartesian'; end
if ~isfield(Grid,'xmin'); Grid.xmin = 0;
    fprintf('Grid.xmin is not defined and has been set to zero.\n');end
if ~isfield(Grid,'xmax'); Grid.xmax = 10;
    fprintf('Grid.xmax is not defined and has been set to 10.\n'); end
if ~isfield(Grid,'Nx');   Grid.Nx   = 10;
    fprintf('Grid.Nx is not defined and has been set to 10.\n');end
Grid.Lx = Grid.xmax-Grid.xmin;    % domain length in x
Grid.dx = Grid.Lx/Grid.Nx;        % dx of the gridblocks

if ~isfield(Grid,'ymin'); Grid.ymin = 0; end
if ~isfield(Grid,'ymax'); Grid.ymax = 1; end
if ~isfield(Grid,'Ny');   Grid.Ny   = 1; end
Grid.Ly = Grid.ymax-Grid.ymin;    % domain length in y
Grid.dy = Grid.Ly/Grid.Ny;        % dy of the gridblocks

if ~isfield(Grid,'zmin'); Grid.zmin = 0; end
if ~isfield(Grid,'zmax'); Grid.zmax = 1; end
if ~isfield(Grid,'Nz');   Grid.Nz   = 1; end
Grid.Lz = Grid.zmax-Grid.zmin;    % domain length in z
Grid.dz = Grid.Lz/Grid.Nz;        % dz of the gridblocks

%% Number for fluxes
Grid.Nfx = (Grid.Nx+1)*Grid.Ny;
Grid.Nfy = Grid.Nx*(Grid.Ny+1);
Grid.Nf = Grid.Nfx + Grid.Nfy;

% x, y, z coords of the 12 corners of the domain
Grid.xdom = [Grid.xmin Grid.xmin Grid.xmin Grid.xmin Grid.xmax Grid.xmax...
    Grid.xmax Grid.xmin Grid.xmin Grid.xmin Grid.xmax Grid.xmin; ...
    Grid.xmax Grid.xmin Grid.xmin Grid.xmax Grid.xmax Grid.xmax...
    Grid.xmax Grid.xmin Grid.xmax Grid.xmin Grid.xmax Grid.xmax];
```

```
Grid.ydom = [Grid.ymin Grid.ymin Grid.ymin Grid.ymax Grid.ymin Grid.ymin...
    Grid.ymax Grid.ymax Grid.ymin Grid.ymin Grid.ymin Grid.ymax;...
    Grid.ymin Grid.ymax Grid.ymin Grid.ymax Grid.ymax Grid.ymin...
    Grid.ymax Grid.ymax Grid.ymin Grid.ymax Grid.ymax Grid.ymax];
Grid.zdom = [Grid.zmin Grid.zmin Grid.zmin Grid.zmin Grid.zmin Grid.zmin...
    Grid.zmin Grid.zmin Grid.zmax Grid.zmax Grid.zmax Grid.zmax;...
    Grid.zmin Grid.zmin Grid.zmax Grid.zmin Grid.zmin Grid.zmax...
    Grid.zmax Grid.zmax Grid.zmax Grid.zmax Grid.zmax Grid.zmax];

% Set up mesh for plotting
% x, y, z coords of the cell centers
Grid.xc = [Grid.xmin+Grid.dx/2:Grid.dx:Grid.xmax-Grid.dx/2]';
% x-coords of gridblock centers
Grid.yc = [Grid.ymin+Grid.dy/2:Grid.dy:Grid.ymax-Grid.dy/2]';
% y-coords of gridblock centers
Grid.zc = [Grid.zmin+Grid.dz/2:Grid.dz:Grid.zmax-Grid.dz/2]';
% z-coords of gridblock centers
Grid.xf = [Grid.xmin:Grid.dx:Grid.xmax]'; % x-coords of gridblock faces
Grid.yf = [Grid.ymin:Grid.dy:Grid.ymax]'; % y-coords of gridblock faces
Grid.zf = [Grid.zmin:Grid.dz:Grid.zmax]'; % z-coords of gridblock faces

%% Set up dof vectors
Grid.N = Grid.Nx*Grid.Ny*Grid.Nz; % total number of gridblocks
Grid.dof = 1:Grid.N;   % cell centered degree of freedom/gridblock number
Grid.dof_f = 1:Grid.Nf;            % face degree of freedom/face number

%% Boundary dof's
% Boundary cells
% make more efficient by avoidng DOF
DOF = reshape(Grid.dof,Grid.Ny,Grid.Nx);
Grid.dof_xmin = DOF(:,1);
Grid.dof_xmax = DOF(:,Grid.Nx);
Grid.dof_ymin = DOF(1,:)';
Grid.dof_ymax = DOF(Grid.Ny,:)';

% Boundary faces
% DOFx = reshape([1:Grid.Nfx],Grid.Ny,Grid.Nx+1);
% Grid.dof_f_xmin = DOFx(:,1);
% Grid.dof_f_xmax = DOFx(:,Grid.Nx+1);
Grid.dof_f_xmin = Grid.dof_xmin;
Grid.dof_f_xmax = [Grid.Nfx-Grid.Ny+1:Grid.Nfx]';


% note: y-fluxes are shifted by Nfx
% make more efficient by avoidng DOFy
DOFy = reshape(Grid.Nfx+[1:Grid.Nfy],Grid.Ny+1,Grid.Nx);
Grid.dof_f_ymin = DOFy(1,:)';
Grid.dof_f_ymax = DOFy(Grid.Ny+1,:)';
```

```
% In prep. for unstructured grid, store all cell volumes and face areas
% Volumes are stored and indexed like unknowns
% Areas are stored and indexed like fluxes

switch Grid.geom
    case 'cartesian' % 1D and 2D
        Grid.A = [ones(Grid.Nfx,1)*Grid.dy*Grid.dz;...
                  ones(Grid.Nfy,1)*Grid.dx*Grid.dz;
                  Grid.dx*Grid.dy];
        Grid.V  = ones(Grid.N,1)*Grid.dx*Grid.dy*Grid.dz;
    case 'polar1D'
        Grid.A = 2*pi*Grid.xf*Grid.dz;
        Grid.V  = pi*Grid.dz*(Grid.xf(2:Grid.Nx+1).^2-Grid.xf(1:Grid.Nx).^2);
    case 'spherical1D'
        Grid.A = 4*pi*Grid.xf.^2;
        Grid.V  = 4*pi*Grid.xc.^2*Grid.dx;
    case 'cylindrical_rz'
        % assumes: y-dir is radial direction and x-dir is cylinder-axis
        Grid.A = [repmat(pi*(Grid.yf(2:Grid.Ny+1).^2-...
            Grid.yf(1:Grid.Ny).^2),Grid.Nx+1,1);... % Ax-faces
                  repmat(2*pi*Grid.yf*Grid.dz,Grid.Nx,1)]; % Ay-faces
        Grid.V  = repmat(Grid.A(1:Grid.Nfx),Grid.Nx,1)*Grid.dx;
    otherwise
        error('Unknown grid geometry.')
end

%% Streamfunction
% Set standard integration origin and direction if not specified
if ~isfield(Grid,'psi_x0');  Grid.psi_x0 = 'xmin_ymin'; end
if ~isfield(Grid,'psi_dir'); Grid.psi_dir = 'xy'; end



function [D,G,I]=build_ops(Grid)
% author: Marc Hesse
% date: 09/08/2014
% description:
% This function computes the discrete divergence and gradient matrices on a
% regular staggered grid using central difference approximations. The
% discrete gradient assumes homogeneous boundary conditions.
% Input:
% Grid = structure containing all pertinent information about the grid.
% Output:
% D = discrete divergence matrix
% G = discrete gradient matrix
% I = identity matrix
%
% Example call:
% >> Grid.xmin = 0; Grid.xmax = 1; Grid.Nx = 10;
```

```
% >> Grid = build_grid(Grid);
% >> [D,G,I]=build_ops(Grid);

Nx = Grid.Nx; Ny = Grid.Ny; Nz = Grid.Nz; N = Grid.N;

if (Nx>1) && (Ny>1) % 2D case
    %% One dimensinal divergence
    Dy = spdiags([-ones(Ny,1) ones(Ny,1)]/Grid.dy,[0 1],Ny,Ny+1);

    %% Two-dimensional divergence
    Dy = spblkdiag(Dy,Nx); % y-component

    e =   ones(Ny*(Nx+1),1);
    Dx = spdiags([-e e]/Grid.dx,[0 Ny],N,(Nx+1)*Ny); % x-component
    D = [Dx Dy];

    dof_f_bnd = [Grid.dof_f_xmin; Grid.dof_f_xmax;... % boundary faces
                 Grid.dof_f_ymin; Grid.dof_f_ymax];
elseif (Nx>1) && (Ny==1)
    D = spdiags([-ones(Nx,1) ones(Nx,1)]/Grid.dx,[0 1],Nx,Nx+1);
    dof_f_bnd = [Grid.dof_f_xmin; Grid.dof_f_xmax];   % boundary faces
elseif (Nx==1) && (Ny>1)
    D = spdiags([-ones(Ny,1) ones(Ny,1)]/Grid.dy,[0 1],Ny,Ny+1);
    dof_f_bnd = [Grid.dof_f_ymin; Grid.dof_f_ymax];   % boundary faces
end


%% Gradient
G = -D';
G(dof_f_bnd,:) = 0;

%% Identity
I = speye(Grid.N);

% % Boundary faces (update to use Gri.dof_f!)
% if (Nx>1) && (Ny>1) % 2D case
%     bnd_x = [1:Ny,Grid.Nfx-Ny+1:Grid.Nfx];
%     bnd_y = [1:Ny+1:Grid.Nfy-Ny+1]; bnd_y = [bnd_y bnd_y+Ny];
%     bnd = [bnd_x Grid.Nfx+bnd_y];
% elseif (Nx>1) && (Ny==1)
%     bnd = [1:Ny,Grid.Nfx-Ny+1:Grid.Nfx]; % Too complicated?
% elseif (Nx==1) && (Ny>1)
%     bnd = [1 Ny+1];
% end
```

```
%% Adjust divergence for different coordinate systems
if strcmp(Grid.geom,'polar1D')
%     Rf = comp_mean(Grid.xc,-1, Grid);
    Rf = spdiags(Grid.xf,0,Nx+1,Nx+1);
    Rcinv = spdiags(1./Grid.xc,0,Nx,Nx);
    D = Rcinv*D*Rf;
elseif strcmp(Grid.geom,'spherical1D')
    Rf = spdiags(Grid.xf.^2,0,Grid.Nx+1,Nx+1);
    Rcinv = spdiags(1./(Grid.xc.^2),0,Nx,Nx);
    D = Rcinv*D*Rf;
elseif strcmp(Grid.geom,'cylindrical_rz')
    % assumes: y-dir is radial direction
    %          simplifies the assembly because grid is ordered y-first
    % The change in geometry goes into 1D matrix before Dy is reassembled
    Rf = spdiags(Grid.yf,0,Ny+1,Ny+1);
    Rcinv = spdiags(1./Grid.yc,0,Ny,Ny);
    Dy = Rcinv*spdiags([-ones(Ny,1) ones(Ny,1)]/Grid.dy,[0 1],Ny,Ny+1)*Rf;
    Dy = spblkdiag(Dy,Nx);
    D = [Dx Dy];
end
```

```
function [Kd] = comp_mean(K,p,kvkh,Grid)
% author: Marc Hesse
% date: 2 Oct 2014
% Description:
% Takes coefficient field, K, defined at the cell centers and computes the
% mean specified by the power, p and returns it in a sparse diagonal
% matrix, Kd.
%
% Input:
% K = Ny by Nx matrix of cell centered values
% p = power of the generalized mean
%       1 (arithmetic mean)
%      -1 (harmonic mean)
% kvkh = ratio of vertical to horizontal conductivity/permeability
% (anisotropy)
% Grid = structure containing information about the grid.
%
% Output:
% Kd = Nf by Nf diagonal matrix of power means at the cell faces.
%
% Example call:
% K = @(x) 1+x.^3;
% Grid.xmin = 0; Grid.xmax = 1; Grid.Nx = 10;
% Grid = build_grid(Grid);
% Kd = comp_mean(K(Grid.xc),1,Grid);
```

```
if (p == -1) | (p == 1)
    if (Grid.Nx == Grid.N) | (Grid.Ny == Grid.N) % 1D
        mean = zeros(Grid.Nx+1,1);
        mean(2:Grid.Nx) = sum(.5*[K(1:Grid.Nx-1),...
            K(2:Grid.Nx)].^p,2).^(1/p);
        Kd = spdiags([mean],[0],Grid.Nx+1,Grid.Nx+1);
    elseif (Grid.N > Grid.Nx) | (Grid.N > Grid.Ny) % 2D
        mean_x = zeros(Grid.Ny,Grid.Nx+1);
        mean_x(:,2:Grid.Nx) = ( (K(:,1:Grid.Nx-1).^p+...
            K(:,2:Grid.Nx).^p)/2 ).^(1/p);

        mean_y = zeros(Grid.Ny+1,Grid.Nx);
        mean_y(2:Grid.Ny,:) = ( (K(1:Grid.Ny-1,:).^p+...
            K(2:Grid.Ny,:).^p)/2 ).^(1/p);

        Kd = spdiags([mean_x(:);kvkh*mean_y(:)],0,Grid.Nf,Grid.Nf);
    else
        error('3D permeability is not implemented')
    end
else
    error('This power does not have significance.')
end



function [A] = flux_upwind(q,Grid)
% author: Marc Hesse
% date: 04/15/2015
% Description:
% This function computes the upwind flux matrix from the flux vector.
%
% Input:
% q = Nf by 1 flux vector from the flow problem.
% Grid = structure containing all pertinent information about the grid.
%
% Output:
% A = Nf by Nf matrix contining the upwinded fluxes
%
% Example call:
% >> Grid.xmin = 0; Grid.xmax = 1; Grid.Nx = 10;
% >> Grid = build_grid(Grid);
% >> q = ones(Grid.Nf,1);
% >> [A] = flux_upwind(q,Grid);

Nx = Grid.Nx; Ny = Grid.Ny; Nz = Grid.Nz; N = Grid.N;
Nfx = Grid.Nfx; % # of x faces
Nfy = Grid.Nfy; % # of y faces

if ((Nx>1) && (Ny==1)) || ((Nx==1) && (Ny>1)) % 1D
```

```
    %% One dimensinal
    qn = min(q(1:Nx),0);
    qp = max(q(2:Nx+1),0);
    A = spdiags([qp,qn],[-1 0],Grid.Nx+1,Grid.Nx);
elseif (Nx>1) && (Ny>1) % 2D
    %% Two dimensional
    % x - fluxes
    qxn = min(q(1:Nfx-Ny),0);
    qxp = max(q(Ny+1:Nfx),0);
    Ax = spdiags([qxp,qxn],[-Ny 0],Nfx,N);

    % y-fluxes
    QY = reshape(q(Nfx+1:end),Grid.Ny+1,Grid.Nx);
    qyn = min(reshape(QY(1:Grid.Ny,:),Grid.N,1),0);
    qyp = max(reshape(QY(2:Grid.Ny+1,:),Grid.N,1),0);
    row_p = (Grid.Ny+1)*repmat([0:Grid.Nx-1],Grid.Ny,1)+...
        repmat([2:Grid.Ny+1]',1,Grid.Nx);
    row_n = (Grid.Ny+1)*repmat([0:Grid.Nx-1],Grid.Ny,1)+...
        repmat([1:Grid.Ny]',  1,Grid.Nx);
    Ay = sparse([row_p(:);row_n(:)],[Grid.dof';Grid.dof'],[qyp;qyn]);

    A = [Ax; Ay];
end



function [u] = solve_lbvp(L,f,B,g,N)
% author: Marc Hesse
% date: 26 Sept 2014
% Description
% Computes the solution u to the linear differential problem given by
%
% \mathcal{L}(u)=f \quad x\in \Omega
%
% with boundary conditions
%
% \mathcal{B}(u)=g \quad x\in\partial\Omega.
%
% Input:
% L = matrix representing the discretized linear operator of size N by N,
%     where N is the number of degrees of fredom
% f = column vector representing the discretized r.h.s. and contributions
%     due non-homogeneous Neumann BC's of size N by 1
% B = matrix representing the constraints arising from Dirichlet BC's of
%     size Nc by N
% g = column vector representing the non-homogeneous Dirichlet BC's of size
%     Nc by 1.
% N = matrix representing a orthonormal basis for the null-space of B and
%     of size N by (N-Nc).
```

```
% Output:
% u = column vector of the solution of size N by 1
%
% Example call:
% >> Grid.xmin = 0; Grid.xmax = 1; Grid.Nx = 10;
% >> Grid = build_grid(Grid);
% >> [D,G,I] = build_ops(Grid);
% >> L = -D*G; fs = ones(Grid.Nx,1);
% >> dof_dir = 1;
% >> B = I(dof_dir,:); g = 1;
% >> N = I; N(:,dof_dir) = [];
% >> h = solve_lbvp(L,fs,B,g,N);

if isempty(B) % no constraints
    u = L\f;
else
    %u0 = zeros(length(f));
    up = spalloc(length(f),1,length(g));
    up = B'*(B*B'\g);
    u0 = N*(N'*L*N\(N'*(f-L*up)));
    u = u0 + up;
end



function [g] = comp_g(D,G,h,fs,Grid,Param)
% author: Marc Hesse
% date: 25 Nov 2014, 10 Jul 2015
% Description:
% Computes the mass conservative fluxes across all boundaries from the
% residual of the compatability condition over the boundary cells.
% Note: Current implmentation works for all cases where one face
%       is assigend to each bnd cell. So conrner cells must have
%       natural BC's on all but one face.
%
% Input:
% D = N by Nf discrete divergence matrix.
% Kd = Nf by Nf conductivity matrix.
% G = Nf by N discrete gradient matrix.
% h = N by 1 vector of flow potential in cell centers.
% fs = N by 1 right hand side vector containing only source terms.
% Grid = structure containing grid information.
% Param = structure contaning problem paramters and information about BC's
%
% Output:
% q = Nf by 1 vector of fluxes across all cell faces
%
% Example call:
% >> Grid.xmin = 0; Grid.xmax = 1; Grid.Nx = 10;
```

```
% >> Grid = build_grid(Grid);
% >> [D,G,I] = build_ops(Grid);
% >> L = -D*G; fs = ones(Grid.Nx,1);
% >> Param.dof_dir = Grid.dof_xmin;
% >> Param.dof_f_dir = Grid.dof_f_xmin;
% >> g = 0;
% >> Param.dof_neu = []; Param.dof_f_neu =[];
% >> [B,N,fn] = build_bnd(Param,Grid);
% >> h = solve_lbvp(L,fs+fn,B,g,N);
% >> q = comp_flux(D,1,G,h,fs,Grid,Param);

%% Compute interior fluxes
g = -G*h;

%% Compute boundary fluxes
dof_cell = [Param.dof_dir;Param.dof_neu];
dof_face = [Param.dof_f_dir;Param.dof_f_neu];
sign = ismember(dof_face,[Grid.dof_f_xmin;Grid.dof_f_ymin])...
      -ismember(dof_face,[Grid.dof_f_xmax;Grid.dof_f_ymax]);

g(dof_face) =  sign.*( D(dof_cell,:) * g + fs(dof_cell)).*...
    Grid.V(dof_cell)./Grid.A(dof_face);
```

Glossary

Symbols

a	=	tortuosity factor
$\left[\frac{^{26}Al}{^{27}Al}\right]^i$	=	initial ^{26}Al to ^{27}Al ratio
A	=	interfacial area
c_p	=	specific heat capacity at constant pressure
C	=	Celsius
Ca	=	capillary number
d	=	grain size
D	=	stiffness matrix
D	=	Darcy
f	=	elongation factor
F	=	formation factor
$\mathbf{g}$	=	gravity field
G	=	gravitational constant
GHz	=	giga hertz
h	=	specific enthalpy
H_0	=	heating production of ^{26}Al decay
H	=	total enthalpy of system
$H(\phi)$	=	smeared-out Heaviside function
j	=	average current density
$\bar{k}$	=	average thermal conductivity of the medium
k	=	permeability
l_C	=	characteristic length
lu	=	lattice units
n	=	derivative matrix
N_{gird}	=	number of grid points
Pe	=	Peclet number
P	=	pressure
q_r	=	volumetric flux of the melt relative to the solid
r	=	radial distance
r	=	mean radius of disconnected pores
R	=	electrical resistivity
$S(\phi)$	=	sign function
S	=	saturation

t	=	pseudo time for evolution of level set
T	=	temperature
T	=	gray-scale threshold
U	=	total dissipated energy
U	=	solid velocity potential
v	=	phase velocity
V	=	computational domain volume
V	=	acoustic velocity
V	=	voltage
X_{Al}	=	initial aluminum content (%weight)
z	=	coordination number

Greek Letters

ϕ	=	porosity (%)
θ	=	dihedral angle ($^\circ$)
κ	=	mean curvature
φ	=	liquid phase level set function
ψ	=	solid phase level set function
γ	=	surface free energy of solid-solid interface
$\vec{n}$	=	normal pointing outward
$\Delta_s\kappa$	=	surface Laplacian of curvature
Θ	=	polar angle
Φ	=	azimuthal angle, final level set
$\delta(\phi)$	=	smeared-out delta function
Δx	=	grid size
Δt	=	time step
ϵ	=	interface bandwidth tuning parameter
ϵ_κ	=	error threshold for mean curvature
ϵ_θ	=	error threshold for dihedral angle
$\circ$	=	degree
μ	=	micro (10^{-6})
ξ	=	tip angle
Ω_i	=	computational domain for level set i
σ	=	effective conductivity
Δp	=	capillary pressure introduced by surface tension forces
$\Delta\sigma$	=	shear stress
ρ	=	density
σ	=	principal stress
γ	=	decay constant
Γ_T	=	radiogenic enthalpy generation

Λ	=	mass melt production rate
μ	=	viscosity
Φ	=	gravitational potential
ξ	=	bulk viscosity of the solid

Superscripts

tr	=	trapped
m	=	cementation factor
n	=	permeability power law
0	=	reference
m	=	melting point

Subscripts

i	=	level set number
ss	=	solid-solid
sl	=	solid-liquid
HC	=	hydrocarbons
p	=	compressional wave
s	=	shear wave
x	=	differential versus x
y	=	differential versus y
ss	=	surface Laplacian
w	=	water
eff	=	effective
s	=	sediments
h	=	halite
sb	=	seabed
o	=	measured at in-situ or formation conditions
o	=	olivine
i	=	iron
m	=	melt
s	=	solid
p	=	at constant pressure
D	=	dimensionless parameters
C	=	characteristic value

Abbreviations

API	=	American Petroleum Institute
AT	=	Atwater Valley
CAI	=	Calcium-Aluminum-rich Inclusion
CFD	=	Computational Fluid Dynamics
DCT	=	X-ray diffraction contrast tomography
DO	=	dead oil
Exp	=	experiment
FL	=	fluorescence
GB	=	gigabytes
GC	=	Green Canyon
GoM	=	Gulf of Mexico
KC	=	Keathley Canyon
LBGK	=	Lattice Bhatnagar-Gross-Krook
LBM	=	Lattice Boltzmann Method
MC	=	Mississippi Canyon
NWA	=	North West Africa
OC	=	oil cut
OS	=	oil stain
PG	=	pressure gradient
SI	=	International System of Units
TACC	=	Texas Advanced Computing Center
TG	=	temperature gradient
WR	=	Walker Ridge

Bibliography

Armstrong, R. T., Porter, M. L., and Wildenschild, D. (2012). Linking pore-scale interfacial curvature to column-scale capillary pressure. *Advances in Water Resources*, 46:55–62.

Bagdassarov, N., Golabek, G. J., Solferino, G., and Schmidt, M. W. (2009). Constraints on the Fe–S melt connectivity in mantle silicates from electrical impedance measurements. *Physics of the Earth and Planetary Interiors*, 177(3–4):139–146.

Ballhaus, C. and Ellis, D. J. (1996). Mobility of core melts during Earth's accretion. *Earth and Planetary Science Letters*, 143(1–4):137–145.

Beere, W. (1975). A unifying theory of the stability of penetrating liquid phases and sintering pores. *Acta Metallurgica*, 23(1):131–138.

Bourbie, T. and Zinszner, B. (1985). Hydraulic and acoustic properties as a function of porosity in Fontainebleau Sandstone. *Journal of Geophysical Research: Solid Earth*, 90(B13):11524–11532.

Bourgoyne Jr, A. T., Millheim, K. K., Chenevert, M. E., and Young Jr., F. S. (1986). *Applied Drilling Engineering*. SPE Textbook Series. Society of Petroleum Engineers, Richardson, TX.

Bredehoeft, J. D. (1988). Will salt repositories be dry? *Eos, Transactions American Geophysical Union*, 69(9):121–131.

Briceño, C., Vivas, A. K., Calvet, N., Hartmann, L., Pacheco, R., Herrera, D., Romero, L., Berlind, P., Sánchez, G., Snyder, J. A., and Andrews, P. (2001). The CIDA-QUEST

large-scale survey of Orion OB1: evidence for rapid disk dissipation in a dispersed stellar population. *Science (New York, N.Y.)*, 291(5501):93–96.

Bruhn, D., Groebner, N., and Kohlstedt, D. L. (2000). An interconnected network of core-forming melts produced by shear deformation. *Nature*, 403(6772):883–886.

Bulau, J. R., Waff, H. S., and Tyburczy, J. A. (1979). Mechanical and Thermodynamic Constraints on Fluid Distribution in Partial Melts. *Journal of Geophysical Research*, 84:6102–6108.

Bunch, T. E., Irving, A. J., Wittke, J. H., Rumble, D., and Aaronson, A. A. (2007). Northwest Africa 2993: A Coarse-grained Lodran-like Achondrite with Affinities to Winonaites. In *2007 Lunar and Planetary Science Conference*, volume 38, page 2211.

Canham, P. B. (1970). The minimum energy of bending as a possible explanation of the biconcave shape of the human red blood cell. *Journal of theoretical biology*, 26(1):61–81.

Carter, N. L., Horseman, S. T., Russell, J. E., and Bredehoeft, J. (1993). Rheology of rocksalt. *Journal of Structural Geology*, 15(9–10):1257–1271.

Carter, W. and Glaeser, A. (1987). The morphological stability of continuous intergranular phases: thermodynamic considerations. *Acta Metallurgica*, 35(I):237–245.

Casas, E. and Lowenstein, T. K. (1989). Diagenesis of saline pan halite; comparison of petrographic features of modern, Quaternary and Permian halites. *Journal of Sedimentary Research*, 59(5):724–739.

Cheadle, M. J. (1989). *Properties of texturally equilibrated two-phase aggregates.* Ph.D., University of Cambridge.

Cheadle, M. J., Elliott, M. T., and McKenzie, D. (2004). Percolation threshold and permeability of crystallizing igneous rocks: The importance of textural equilibrium. *Geology*, 32(9):757–760.

Chopp, D. L. and Sethian, J. A. (1999). Motion by Intrinsic Laplacian of Curvature. *Interfaces and Free Boundaries*, 1:1–18.

Clarke, D. (1989). Intergranular Phases in Polycrystalline Ceramics. In Dufour, L.-C., Monty, C., and Petot-Ervas, G., editors, *Surfaces and Interfaces of Ceramic Materials*, volume 173 of *NATO ASI Series*, pages 57–79. Springer Netherlands.

Cmiral, M., Gerald, J. D. F., Faul, U. H., and Green, D. H. (1998). A close look at dihedral angles and melt geometry in olivine-basalt aggregates: a TEM study. *Contributions to Mineralogy and Petrology*, 130(3-4):336–345.

Courant, R. and Hilbert, D. (1989). *Methods of Mathematical Physics*. Wiley-VCH, volume 1 edition.

Cristescu, N. D. and Hunsche, U. (1998). Time Effects in Rock Mechanics. In *Time Effects in Rock Mechanics*, Wiley Series in Materials, Modeling and Computation. John Wiley & Sons, New York.

Dash, J. G., Fu, H., and Wettlaufer, J. S. (1995). The premelting of ice and its environmental consequences. *Reports on Progress in Physics*, 58(1):115.

Dauphas, N. and Chaussidon, M. (2011). A Perspective from Extinct Radionuclides on a Young Stellar Object: The Sun and Its Accretion Disk. *Annual Review of Earth and Planetary Sciences*, 39(1):351–386.

Davison, I. (2009). Faulting and fluid flow through salt. *Journal of the Geological Society*, 166(2):205–216.

de Gennes, P. G. (1985). Wetting: statics and dynamics. *Reviews of Modern Physics*, 57(3):827–863.

Desbois, G., Urai, J. L., Kukla, P. A., Wollenberg, U., Pérez-Willard, F., Radí, Z., and Riholm, S. (2012). Distribution of brine in grain boundaries during static recrystallization in wet, synthetic halite: insight from broad ion beam sectioning and SEM observation at cryogenic temperature. *Contributions to Mineralogy and Petrology*, 163(1):19–31.

Deuling, H. and Helfrich, W. (1976a). The curvature elasticity of fluid membranes : A catalogue of vesicle shapes. *Journal de Physique*, 37(11):1335–1345.

Deuling, H. J. and Helfrich, W. (1976b). Red blood cell shapes as explained on the basis of curvature elasticity. *Biophysical Journal*, 16(8):861–868.

DiDonna, B. A. and Kamien, R. D. (2002). Smectic Phases with Cubic Symmetry: The Splay Analog of the Blue Phase. *Physical Review Letters*, 89(21).

Downey, M. W. (1984). Evaluating seals for hydrocarbon accumulations. *AAPG Bulletin*, 68(11):1752–1763.

Drury, M. R. and Urai, J. L. (1990). Deformation-related recrystallization processes. *Tectonophysics*, 172(3–4):235–253.

Elkins-Tanton, L. T., Weiss, B. P., and Zuber, M. T. (2011). Chondrites as samples of differentiated planetesimals. *Earth and Planetary Science Letters*, 305(1–2):1–10.

Endres, A. L., Murray, T., Booth, A. D., and West, L. J. (2009). A new framework for estimating englacial water content and pore geometry using combined radar and seismic wave velocities. *Geophysical Research Letters*, 36(4):L04501.

Evans, R. L., Tarits, P., Chave, A. D., White, A., Heinson, G., Filloux, J. H., Toh, H., Seama, N., Utada, H., Booker, J. R., and Unsworth, M. J. (1999). Asymmetric Electrical Structure in the Mantle Beneath the East Pacific Rise at 17°S. *Science*, 286(5440):752–756.

Faul, U. H. (1997). Permeability of partially molten upper mantle rocks from experiments and percolation theory. *Journal of Geophysical Research: Solid Earth*, 102(B5):10299–10311.

Faul, U. H., Toomey, D. R., and Waff, H. S. (1994). Intergranular basaltic melt is distributed in thin, elongated inclusions. *Geophysical Research Letters*, 21:29–32.

Forrest, J., Marcucci, E., and Scott, P. (2005). Geothermal gradients and subsurface temperatures in the northern Gulf of Mexico. *Gulf Coast Association of Geological Societies Transactions*, 55:233–248.

Gaetani, G. A. and Grove, T. L. (1999). Wetting of mantle olivine by sulfide melt: implications for Re/Os ratios in mantle peridotite and late-stage core formation. *Earth and Planetary Science Letters*, 169(1–2):147–163.

Garapic, G., Faul, U. H., and Brisson, E. (2013). High-resolution imaging of the melt distribution in partially molten upper mantle rocks: evidence for wetted two-grain boundaries. *Geochemistry, Geophysics, Geosystems*, 14(3):556–566.

Garboczi, E. J. (1998). Finite Element and Finite Difference Programs for Computing the Linear Electric and Elastic Properties of Digital Images of Random Materials.

Garboczi, E. J. and Douglas, J. F. (1996). Intrinsic conductivity of objects having arbitrary shape and conductivity. *Physical Review E*, 53(6):6169–6180.

German, R. M., Suri, P., and Park, S. J. (2009). Review: liquid phase sintering. *Journal of Materials Science*, 44(1):1–39.

Ghanbarzadeh, S., Hanafizadeh, P., and Saidi, M. H. (2012). Intelligent Image-Based Gas-Liquid Two-Phase Flow Regime Recognition. *Journal of Fluids Engineering*, 134(6):061302–061302.

Ghanbarzadeh, S., Hanafizadeh, P., Saidi, M. H., and Boozarjomehry, R. B. (2010a). Intelligent Regime Recognition in Upward Vertical Gas-Liquid Two Phase Flow Using Neural Network Techniques. In *ASME Proceedings*, volume 2, pages 293–302, Montreal, Quebec, Canada.

Ghanbarzadeh, S., Hanafizadeh, P., Saidi, M. H., and Bozorgmehry B., R. (2010b). Fuzzy Clustering of Vertical Two Phase Flow Regimes Based on Image Processing Technique. In *ASME Proceedings*, volume 2, pages 303–313, Montreal, Quebec, Canada.

Ghanbarzadeh, S., Hesse, M. A., Prodanovic, M., and Gardner, J. E. (2015a). Synthetic rock salt. *Digital Rocks Portal of The University of Texas at Austin*.

Ghanbarzadeh, S., Hesse, M. A., and Prodanović, M. (2015b). A level set method for materials with texturally equilibrated pores. *Journal of Computational Physics*, 297:480–494.

Ghanbarzadeh, S., Hesse, M. A., Prodanović, M., and Gardner, J. E. (2015c). Deformation-assisted fluid percolation in rock salt. *Science*, 350(6264):1069–1072.

Ghanbarzadeh, S., Prodanovic, M., and Hesse, M. A. (2014). Percolation and Grain Boundary Wetting in Anisotropic Texturally Equilibrated Pore Networks. *Phys. Rev. Lett.*, 113(4):048001.

Gibbs, J. W. (1957). *The collected works of J. Willard Gibbs*. Yale University Press.

Gibiansky, L. V. and Milton, G. W. (1993). On the Effective Viscoelastic Moduli of Two-Phase Media. I. Rigorous Bounds on the Complex Bulk Modulus. *Proceedings of the Royal Society of London. Series A: Mathematical and Physical Sciences*, 440(1908):163–188.

Gibiansky, L. V., Milton, G. W., and Berryman, J. G. (1999). On the effective viscoelastic moduli of two–phase media. III. Rigorous bounds on the complex shear modulus in two dimensions. *Proceedings of the Royal Society of London. Series A: Mathematical, Physical and Engineering Sciences*, 455(1986):2117–2149.

Hanafizadeh, P., Ghanbarzadeh, S., and Saidi, M. H. (2011a). Visual technique for detection of gas–liquid two-phase flow regime in the airlift pump. *Journal of Petroleum Science and Engineering*, 75(3–4):327–335.

Hanafizadeh, P., Saidi, M. H., Nouri Gheimasi, A., and Ghanbarzadeh, S. (2011b). Experimental investigation of air–water, two-phase flow regimes in vertical mini pipe. *Scientia Iranica*, 18(4):923–929.

Hansen, F. D. and Leigh, C. D. (2011). Salt disposal of heat-generating nuclear waste. Technical Report SAND2011-0161, Sandia National Laboratories, Albuquerque, NM.

Hewitt, I. J. and Fowler, A. C. (2008). Partial melting in an upwelling mantle column. *Proceedings of the Royal Society A: Mathematical, Physical and Engineering Science*, 464(2097):2467–2491.

Hildebrandt, S. and Tromba, A. (1985). *Mathematics and Optimal Form*. Scientific American Library.

Hildenbrand, A. and Urai, J. (2003). Investigation of the morphology of pore space in mudstones—first results. *Marine and Petroleum Geology*, 20(10):1185–1200.

Hirth, G. and Kohlstedt, D. L. (1995). Experimental constraints on the dynamics of the partially molten upper mantle: Deformation in the diffusion creep regime. *Journal of Geophysical Research: Solid Earth*, 100(B2):1981–2001.

Holness, B, M. (2010). Decoding dihedral angles in melt-bearing and solidified rocks. In: (Eds.) M.A. Forster and John D. Fitz Gerald,. *Journal of the Virtual Explorer*, 35:paper 2.

Holness, M. B. and Lewis, S. (1997). The structure of the halite-brine interface inferred from pressure and temperature variations of equilibrium dihedral angles in the halite-H2o-CO2 system. *Geochimica et Cosmochimica Acta*, 61(4):795–804.

Huber, C., Parmigiani, A., Latt, J., and Dufek, J. (2013). Channelization of buoyant non-wetting fluids in saturated porous media. *Water Resources Research*, 49(10):6371–6380.

Ingebritsen, S. E., Sanford, W. E., and Neuzil, C. E. (2006). *Groundwater in geologic processes*. Cambridge University Press, Cambridge; New York.

Jackson, M. P. A. and Talbot, C. J. (1986). External shapes, strain rates, and dynamics of salt structures. *Geological Society of America Bulletin*, 97(3):305–323.

Jettestuen, E., Helland, J. O., and Prodanovic, M. (2013). A level set method for simulating capillary-controlled displacements at the pore scale with nonzero contact angles. *Water Resources Research*, 49(8):4645–4661.

Jones, J. H. and Drake, M. J. (1986). Geochemical constraints on core formation in the Earth. *Nature*, 322(6076):221–228.

Jurewicz, S. R. and Watson, E. (1985). The distribution of partial melt in a granitic system: The application of liquid phase sintering theory. *Geochimica et Cosmochimica Acta*, 49:1109–1121.

Kleine, T., Munker, C., Mezger, K., and Palme, H. (2002). Rapid accretion and early core formation on asteroids and the terrestrial planets from Hf-W chronometry. *Nature*, 418(6901):952–955.

Land, L. S., Kupecz, J. A., and Mack, L. (1988). Louann salt geochemistry (Gulf of Mexico sedimentary basin, U.S.A.): A preliminary synthesis. *Chemical Geology*, 74(1–2):25–35.

Laporte, D. and Provost, A. (2000). The Grain-Scale Distribution of Silicate, Carbonate and Metallosulfide Partial Melts: a Review of Theory and Experiments. In Bagdassarov, N., Laporte, D., and Thompson, A. B., editors, *Physics and Chemistry of Partially Molten Rocks*, number 11 in Petrology and Structural Geology, pages 93–140. Springer Netherlands. DOI: 10.1007/978-94-011-4016-4_4.

Lee, W., Son, G., and Jeong, J. J. (2010). Numerical Analysis of Bubble Growth and Departure from a Microcavity. *Numerical Heat Transfer, Part B: Fundamentals*, 58(5):323–342.

Lewis, S. and Holness, M. (1996). Equilibrium halite-{H$_2$O} dihedral angles: High rock-salt permeability in the shallow crust? *Geology*, 24(5):431–434.

Li, Z., Lai, M.-C., He, G., and Zhao, H. (2010). An augmented method for free boundary problems with moving contact lines. *Computers & Fluids*, 39(6):1033–1040.

Liang, Y., Price, J. D., Wark, D. A., and Watson, E. B. (2001). Nonlinear pressure diffusion in a porous medium: Approximate solutions with applications to permeability measurements using transient pulse decay method. *Journal of Geophysical Research: Solid Earth*, 106(B1):529–535.

Lindquist, W. B., Lee, S.-M., Coker, D. A., Jones, K. W., and Spanne, P. (1996). Medial axis analysis of void structure in three-dimensional tomographic images of porous media. *Journal of Geophysical Research: Solid Earth*, 101(B4):8297–8310.

Lindquist, W. B., Lee, S.-M., Oh, W., Venkatarangan, A. B., Shin, H., and Prodanovic, M. (2005). 3dma-Rock Primer: A Software Package for Automated Analysis of Rock Pore Structure in 3-D Computed Microtomography Images.

Liu, H., Krishnan, S., Marella, S., and Udaykumar, H. S. (2005). Sharp interface Cartesian grid method II: A technique for simulating droplet interactions with surfaces of arbitrary shape. *Journal of Computational Physics*, 210(1):32–54.

Lorensen, W. E. and Cline, H. E. (1987). Marching cubes: A high resolution 3d surface construction algorithm. In *SIGGRAPH 1987*, volume 21 of *4*, pages 163–169, Anaheim, CA, USA.

Ludwig, W., Reischig, P., King, A., Herbig, M., Lauridsen, E. M., Johnson, G., Marrow, T. J., and Buffière, J. Y. (2009). Three-dimensional grain mapping by x-ray diffraction contrast tomography and the use of Friedel pairs in diffraction data analysis. *Review of Scientific Instruments*, 80(3):033905.

Lundstrom, C. C., Gill, J., Williams, Q., and Perfit, M. R. (1995). Mantle Melting and Basalt Extraction by Equilibrium Porous Flow. *Science*, 270:1958–1961.

Mader, H. M. (1992). Observations of the water-vein system in polycrystalline ice. *Journal of Glaciology*, 38(130):333–347.

Mader, H. M., Pettitt, M. E., Wadham, J. L., Wolff, E. W., and Parkes, R. J. (2006). Subsurface ice as a microbial habitat. *Geology*, 34(3):169–172.

McKenzie, D. (1984). The generation and compaction of partially molten rock. *Journal of Petrology*, 25(3):713–765.

McKenzie, D. (1985). The extraction of magma from the crust and mantle. *Earth and Planetary Science Letters*, 74(1):81–91.

McKenzie, D. (1989). Some remarks on the movement of small melt fractions in the mantle. *Earth and Planetary Science Letters*, 95(1–2):53–72.

Meille, S. and Garboczi, E. J. (2001). Linear elastic properties of 2d and 3d models of porous materials made from elongated objects. *Modelling and Simulation in Materials Science and Engineering*, 9(5):371.

Miller, K. J., Zhu, W.-l., Montési, L. G. J., and Gaetani, G. A. (2014). Experimental quantification of permeability of partially molten mantle rock. *Earth and Planetary Science Letters*, 388:273–282.

Milton, G. W., Seppecher, P., and Bouchitte, G. (2009). Minimization variational principles for acoustics, elastodynamics, and electromagnetism in lossy inhomogeneous bodies at fixed frequency. *Proceedings of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 465(2102):367–396. arXiv:0807.1336 [cond-mat, physics:math-ph, physics:physics].

Minarik, B. (2003). The core of planet formation. *Nature*, 422(6928):126–128.

Minarik, W. G., Ryerson, F. J., and Watson, E. B. (1996). Textural Entrapment of Core-Forming Melts. *Science*, 272(5261):530–533.

Minarik, W. G. and Watson, E. B. (1995). Interconnectivity of carbonate melt at low melt fraction. *Earth and Planetary Science Letters*, 133(3–4):423–437.

Mitchell, I. M. (2008). The Flexible, Extensible and Efficient Toolbox of Level Set Methods. *Journal of Scientific Computing*, 35(2-3):300–329.

Murthy, V. R. (1991). Early Differentiation of the Earth and the Problem of Mantle Siderophile Elements: A New Approach. *Science*, 253(5017):303–306.

Noseck, U., Wolf, J., Steininger, W., and Miller, B. (2015). Identification and applicability of analogues for a safety case for a HLW repository in evaporites: results from a NEA workshop. *Swiss Journal of Geosciences*, pages 1–8.

Nye, J. F. (1989). The Geometry of Water Veins and Nodes in Polycrystalline Ice. *Journal of Glaciology*, 35(119):17–22.

Oh, W. and Lindquist, W. B. (1999). Image thresholding by indicator kriging. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 21(7):590–602.

Oprea, J. (2000). *The Mathematics of Soap Films: Explorations With Maple (Student Mathematical Library, Vol. 10)*. Amer Mathematical Society.

Oprea, J. (2007). *Differential Geometry and its Applications*. (Mathematical Association of America, Washington, D.C.

Osher, S. and Sethian, J. A. (1988). Fronts propagating with curvature-dependent speed: Algorithms based on Hamilton-Jacobi formulations. *Journal of Computational Physics*, 79(1):12–49.

Osher, S. J. and Fedkiw, R. P. (2002). *Level Set Methods and Dynamic Implicit Surfaces*. Springer.

Parsons, R. A., Nimmo, F., Hustoft, J. W., Holtzman, B. K., and Kohlstedt, D. L. (2008). An experimental and numerical study of surface tension-driven melt flow. *Earth and Planetary Science Letters*, 267(3–4):548–557.

Paulsen, W. H. (1994). What Is the Shape of a Mylar Balloon? *The American Mathematical Monthly*, 101(10):953.

Peach, C. J. and Spiers, C. J. (1996). Influence of crystal plastic deformation on dilatancy and permeability development in synthetic salt rock. *Tectonophysics*, 256(1–4):101–128.

Peach, C. J., Spiers, C. J., and Trimby, P. W. (2001). Effect of confining pressure on dilatation, recrystallization, and flow of rock salt at 150°C. *Journal of Geophysical Research*, 106(B7):13315–13328.

Peng, D., Merriman, B., Osher, S., Zhao, H., and Kang, M. (1999). A PDE-Based Fast Local Level Set Method. *Journal of Computational Physics*, 155(2):410–438.

Perona, P. and Malik, J. (1990). Scale-space and edge detection using anisotropic diffusion. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 12(7):629–639.

Pervukhina, M. and Kuwahara, Y. (2008). Correlations between electrical and elastic properties of solid–liquid composites with interfacial energy-controlled equilibrium microstructures. *Earth and Planetary Science Letters*, 265(3–4):410–422.

Popp, T., Kern, H., and Schulze, O. (2001). Evolution of dilatancy and permeability in rock salt during hydrostatic compaction and triaxial deformation. *Journal of Geophysical Research: Solid Earth*, 106(B3):4061–4078.

Price, P. B. (2000). A habitat for psychrophiles in deep Antarctic ice. *Proceedings of the National Academy of Sciences*, 97(3):1247–1251.

Prodanovic, M. and Bryant, S. L. (2006). A level set method for determining critical curvatures for drainage and imbibition. *Journal of Colloid and Interface Science*, 304(2):442–458.

Raoof, A. and Hassanizadeh, S. M. (2009). A New Method for Generating Pore-Network Models of Porous Media. *Transport in Porous Media*, 81(3):391–407.

Rempel, A. W., Waddington, E. D., Wettlaufer, J. S., and Worster, M. G. (2001). Possible displacement of the climate signal in ancient ice by premelting and anomalous diffusion. *Nature*, 411(6837):568–571.

Ribe, N. M. (1985). The deformation and compaction of partial molten zones. *Geophysical Journal of the Royal Astronomical Society*, 83(2):487–501.

Riley Jr., G. N. and Kohlstedt, D. L. (1991). Kinetics of melt migration in upper mantle-type rocks. *Earth and Planetary Science Letters*, 105(4):500–521.

Roberts, J. J., Kinney, J. H., Siebert, J., and Ryerson, F. J. (2007). Fe-Ni-S melt permeability in olivine: Implications for planetary core formation. *Geophysical Research Letters*, 34(14):L14306.

Sagan, H. (1992). *Introduction to the Calculus of Variations*. Courier Dover Publications.

Saleh, S., Williams, K., Rizvi, A., and others (2013). Rubble Zone Below Salt: Identification and Best Drilling Practices. In *SPE Annual Technical Conference and Exhibition*, page SPE166115, New Orleans, LA, USA. Society of Petroleum Engineers.

Schléder, Z. and Urai, J. L. (2005). Microstructural evolution of deformation-modified primary halite from the Middle Triassic Röt Formation at Hengelo, The Netherlands. *International Journal of Earth Sciences*, 94(5-6):941–955.

Schneider, C. A., Rasband, W. S., and Eliceiri, K. W. (2012). NIH Image to ImageJ: 25 years of image analysis. *Nature Methods*, 9(7):671–675.

Schoenherr, J., Urai, J. L., Kukla, P. A., Littke, R., Schléder, Z., Larroque, J.-M., Newall, M. J., Al-Abry, N., Al-Siyabi, H. A., and Rawahi, Z. (2007). Limits to the sealing capacity of rock salt: A case study of the infra-Cambrian Ara Salt from the South Oman salt basin. *AAPG Bulletin*, 91(11):1541–1557.

Schulze, O., Popp, T., and Kern, H. (2001). Development of damage and permeability in deforming rock salt. *Engineering Geology*, 61(2–3):163 – 180. Geosciences and Nuclear Waste Disposal.

Schwarz, U. S. and Gompper, G. (2000). Stability of Inverse Bicontinuous Cubic Phases in Lipid-Water Mixtures. *Physical Review Letters*, 85(7):1472–1475.

Sethian, J. A. (1999). *Level set methods and fast marching methods*. Cambridge University Press, Cambridge, 2 edition.

Shannon, M. C. and Agee, C. B. (1996). High pressure constraints on percolative core formation. *Geophysical Research Letters*, 23(20):2717–2720.

Shannon, M. C. and Agee, C. B. (1998). Percolation of Core Melts at Lower Mantle Conditions. *Science*, 280(5366):pp. 1059–1061.

Shi, C. Y., Zhang, L., Yang, W., Liu, Y., Wang, J., Meng, Y., Andrews, J. C., and Mao, W. L. (2013). Formation of an interconnected network of iron melt at Earth's lower mantle conditions. *Nature Geoscience*, 6(11):971–975.

Smereka, P. (2003). Semi-Implicit Level Set Methods for Curvature and Surface Diffusion Motion. *Journal of Scientific Computing*, 19(1-3):439–456.

Smith, C. S. (1948). Grains, Phases, And Interfaces: An Interpretation Of Microstructure. *Transactions of Metallurgical Society of AIME*, 175:15–51.

Smith, C. S. (1964). Some Elementary Principles of Polycrystalline Microstructure. *Metallurgical Reviews*, 9(1):1–48.

Sobolev, A. V. and Shimizu, N. (1993). Superdepleted melts and ocean mantle permeability. *Transactions Doklady-Russian Academy of Sciences Earth Science Sections*, 328:182–182.

Spelt, P. D. M. (2005). A level-set approach for simulations of flows with multiple moving contact lines with hysteresis. *Journal of Computational Physics*, 207(2):389–404.

Spiegelman, M. and Elliott, T. (1993). Consequences of melt transport for uranium series disequilibrium in young lavas. *Earth and Planetary Science Letters*, 118:1–20.

Stevenson, D. J. (1990). Fluid dynamics of core formation. In Newsom, H. E. and Jones, J. H., editors, *Origin of the Earth*, pages 231–250. Oxford University Press, Oxford.

Stewart, S. A. (2007). Salt tectonics in the North Sea Basin: a structural style template for seismic interpreters. *Geological Society, London, Special Publications*, 272(1):361–396.

Takei, Y. (2002). Effect of pore geometry on VP/VS: From equilibrium geometry to crack. *Journal of Geophysical Research: Solid Earth*, 107(B2):ECV6 1–ECV6 12.

Takei, Y. and Hier-Majumder, S. (2009). A generalized formulation of interfacial tension driven fluid migration with dissolution/precipitation. *Earth and Planetary Science Letters*, 288(1 - 2):138 – 148.

Taylor, G. J. (1992). Core formation in asteroids. *Journal of Geophysical Research: Planets*, 97(E9):14717–14726.

Thiemeyer, N., Habersetzer, J., Peinl, M., Zulauf, G., and Hammer, J. (2015). The application of high resolution X-ray computed tomography on naturally deformed rock salt: Multi-scale investigations of the structural inventory. *Journal of Structural Geology*, 77:92 – 106.

Thompson, D. W. and Biology (1992). *On Growth and Form: The Complete Revised Edition*. Dover Publications, revised edition.

Trinquier, A., Birck, J.-L., Allègre, C. J., Göpel, C., and Ulfbeck, D. (2008). 53mn- 53cr systematics of the early Solar System revisited. *Geochimica et Cosmochimica Acta*, 72:5146–5163.

Tromans, D. and Meech, J. A. (2002). Fracture toughness and surface energies of minerals: theoretical estimates for oxides, sulphides, silicates and halides. *Minerals Engineering*, 15(12):1027–1041.

Troutman, J. L. (1995). *Variational Calculus and Optimal Control: Optimization with Elementary Convexity*. Springer, 2nd edition.

Tucker, M. (1979). A simple description of interconnected grain edge porosity. *Journal of Nuclear Materials*, 79(1):199–205.

Urai, J. L. (1983). Water assisted dynamic recrystallization and weakening in polycrystalline bischofite. *Tectonophysics*, 96(1-2):125–157.

Urai, J. L., Spiers, C. J., Zwart, H. J., and Lister, G. S. (1986). Weakening of rock salt by water during long-term creep. *Nature*, 324(6097):554–557.

van der Marck, S. C. (1999). Evidence for a nonzero tansport threshold in porous media. *Water Resources Research*, 35(2):595–599.

von Bargen, N. and Waff, H. S. (1986). Permeabilities, interfacial areas and curvatures of partially molten systems: results of numerical computations of equilibrium microstructures. *Journal of Geophysical Research*, 91(B9):9261–9276.

Waff, H. S. and Faul, U. H. (1992). Effects of crystalline anisotropy on fluid distribution in ultramafic partial melts. *Journal of Geophysical Research: Solid Earth*, 97(B6):9003–9014.

Wark, D. A. and Watson, E. (1998). Grain-scale permeabilities of texturally equilibrated, monomineralic rocks. *Earth and Planetary Science Letters*, 164(3–4):591–605.

Watanabe, T. and Peach, C. J. (2002). Electrical impedance measurement of plastically deforming halite rocks at 125°C and 50 MPa. *Journal of Geophysical Research: Solid Earth*, 107(B1):ECV 2–1.

Watson, H. C. and Roberts, J. J. (2011). Connectivity of core forming melts: Experimental constraints from electrical conductivity and X-ray tomography. *Physics of the Earth and Planetary Interiors*, 186(3–4):172–182.

Weinstock, R. (1974). *Calculus of Variations*. Courier Dover Publications.

Wildenschild, D. and Sheppard, A. P. (2013). X-ray imaging and analysis techniques for quantifying pore-scale structure and processes in subsurface porous medium systems. *Advances in Water Resources*, 51:217–246.

Wimert, J. and Hier-Majumder, S. (2012). A three-dimensional microgeodynamic model of melt geometry in the Earth's deep interior. *Journal of Geophysical Research: Solid Earth*, 117(B4):B04203.

Wolff, E. W. and Paren, J. G. (1984). A two-phase model of electrical conduction in polar ice sheets. *Journal of Geophysical Research: Solid Earth*, 89(B11):9433–9438.

Yaramanci, U. (1994). Relation of in situ resistivity to water content in salt rocks. *Geophysical Prospecting*, 42(3):229–239.

Yaramanci, U. and Flach, D. (1992). Resistivity of Rock-Salt in Asse (germany) and Petrophysical Aspects. *Geophysical Prospecting*, 40(1):85–100.

Yin, Q., Jacobsen, S. B., Yamashita, K., Blichert-Toft, J., Telouk, P., and Albarede, F. (2002). A short timescale for terrestrial planet formation from Hf-W chronometry of meteorites. *Nature*, 418(6901):949–952.

Yoshino, T., Matsuzaki, T., Shatskiy, A., and Katsura, T. (2009). The effect of water on the electrical conductivity of olivine aggregates and its implications for the electrical structure of the upper mantle. *Earth and Planetary Science Letters*, 288(1–2):291–300.

Yoshino, T., Walter, M. J., and Katsura, T. (2003). Core formation in planetesimals triggered by permeable flow. *Nature*, 422(6928):154–157.

Zhu, W., Gaetani, G. A., Fusseis, F., Montesi, L. G. J., and Carlo, F. D. (2011). Microtomography of Partially Molten Rocks: Three-Dimensional Melt Distribution in Mantle Peridotite. *Science*, 332(6025):88–91.

Vita

Soheil Ghanbarzadeh was born in Ahavz, Khuzestan province, Iran on 15 February 1987, the son of Hamid Ghanbarzadeh and Maasoumeh Amand. He received the Bachelor of Science degree in Mechanical Engineering from Sharif University of Technology in 2008 and immediately started the graduate studies for a Master of Science in Mechanical Engineering in the same school. He earned his Master's degree in 2010 and joined the Sharif Energy Research Institute as a research engineer. His duties included analysis of efficiency of gas compressor stations located along the natural gas pipelines and providing solutions for increasing the efficiency. He developed an energy and mass transfer model for natural gas compressor stations and implemented optimization methods in order to minimize energy consumption and mass loss. He also performed field experiments on turbo compressors, exhaust stack and scrubbers. Always having the dream of pursuing a doctoral education, he applied to the University of Texas at Austin's petroleum engineering program. He was accepted and started his doctoral studies in January, 2012.

Permanent address: 3563C Lake Austin Blvd
Austin, Texas 78703

This dissertation was typeset with LaTeX[†] by the author.

[†]LaTeX is a document preparation system developed by Leslie Lamport as a special version of Donald Knuth's TeX Program.