NTNU

Norwegian University of
Science and Technology

A Pore Network Investigation of Factors Influencing the Residual Oil Saturation

Anders Torland

Petroleum Geoscience and Engineering
Submission date: June 2018
Supervisor: Carl Fredrik Berg, IGP
Co-supervisor: Ståle Fjeldstad, FEI Trondheim

Norwegian University of Science and Technology
Department of Geoscience and Petroleum

Preface

This thesis is the final product of the five-year MSc program in Petroleum Geosciences and Engineering with specialization in Reservoir Engineering and Petrophysics. It was written at the Department of Geoscience and Petroleum (IGP) at the Norwegian University of Science and Technology (NTNU) in Trondheim, Norway. This master's thesis is a natural extension and continuation of the project report "Implementation and Comparison of Network Simulations for Permeability and Formation Factor", written during the fall of 2017.

I would like to express my gratitude to my supervisor, Associate Professor Carl Fredrik Berg, whose support and advice has been invaluable. Thank you for providing me with an interesting topic and encouraging me along the way. Second, I would like to thank my co-supervisor, Ståle Fjeldstad (FEI Trondheim), for sharing his knowledge and experience. I would also like to thank Sean Hall for proofreading.

Trondheim, June 2018
Anders Torland

Summary

In this thesis a pore network modelling approach is used to investigate the correlation between different pore structure descriptors and the residual oil saturation at the capillary limit. For this study, a range of sandstones, sandpacks and carbonates, including Fontainebleau, Bentheimer and Berea sandstones, were investigated. A suite of scripts were developed in Python to perform simulations and calculate different properties on these network models. The scripts perform a wide variety of tasks, such as potential calculations, streamline and cluster tracking, network manipulation, percolation and plotting.

Primary drainage and waterflooding was simulated at different wetting conditions using the e-Core software to obtain residual oil saturation. Different flow and geometry based pore structure descriptors, such as coordination number, constriction factor, aspect ratio and porosity, were calculated and their correlations with the residual oil saturation at different wetting conditions were compared. The correlation between two different local aspect ratios and the local residual oil saturation was also investigated.

From the obtained results it is evident that the pore structure has a measurable impact on the residual oil saturation. However, the trapping mechanisms are complex, and as the pore structure measures are dependent on each other, it is difficult to find good correlations based on one sole descriptor. The effective porosity yielded the best correlation, which is likely due to the relationship between porosity, aspect ratio and coordination number. The constriction factor, the global aspect ratio and average local aspect ratio exhibit poor correlations with the residual oil saturation. This is partly caused by the carbonates, which deviates from the results obtained from other networks.

A bond percolation process was used to find the bond percolation threshold of the networks studied in this thesis. At the bond percolation threshold, the connectivity through the network is broken and the oil relative permeability is zero. The associated threshold capillary pressure for snap-off exhibited good correlation with the recorded capillary pressure at which the oil relative permeability becomes zero during water flooding. This correspondence indicates that the percolation threshold is the dominating factor for when the oil-phase lose connectivity. The cluster-size distribution of oil after waterflooding was also investigated. However, we did not observe any correlations between the cluster-sizes and the pore structure descriptors.

Sammendrag

I denne oppgaven brukes porenettverksmodellering til å undersøke korrelasjonene mellom ulike mål på porestruktur og den residuelle oljemetningen ved kapillærgrensen. I denne studien ble en rekke sandstener, sandpakker og karbonater, blant annet Fontainebleau, Bentheimer og Berea sandstein, undersøkt. Simuleringer og beregninger av ulike egenskaper fra nettverksmodellene ble utført av en rekke egenutviklede skript. Skriptene ble skrevet i Pyhton og utfører flere ulike oppgaver, blant annet potensialberegninger, sporing av strømlinjer og oljeansamlinger, nettverksmanipulering, perkolering og visualisering.

For å finne residuell oljemetning ble primær drenering og vannfløming simulert ved forskjellige fuktingsforhold ved hjelp av programvaren e-Core. Forskjellige flyt- og geometribaserte porestrukturmål, slik som koordinasjonsnummer (antallet porehalser koblet til en pore), innsnevringsfaktor, aspektforhold (forholdet mellom pore- og porehalsradius) og porøsitet, ble beregnet og deres korrelasjoner med den residuelle oljemetningen ved forskjellige fuktingsforhold ble sammenlignet. Korrelasjonen mellom to forskjellige lokale mål på aspektforholdet og lokal residuell oljemetning ble også undersøkt.

Fra de oppnådde resultatene er det tydelig at porestrukturen har stor påvirkning på den residuelle oljemetningen. Imidlertid er fangstmekanismene komplekse og målene på porestruktur er avhengige av hverandre. Det er derfor vanskelig å finne gode korrelasjoner basert på kun et mål. Den effektive porøsiteten ga den beste korrelasjonen, noe som sannsynligvis skyldes forholdet mellom porøsitet, aspektforhold og koordinasjonsnummer. Innsnevringsfaktoren, det globale aspektforholdet og det gjennomsnittlige lokale aspektforholdet ga dårlige korrelasjoner med residuell oljemetning. Dette skyldes delvis karbonatene, som avviker fra resultatene fra de andre nettverkene.

En båndperkoleringsprosess ble brukt for å finne den kritiske perkoleringssansynligheten for nettverkene som ble studert i denne oppgaven. Ved den kritiske perkoleringssansynligheten er forbindelsen gjennom nettverket brutt, og oljens relative permeabiliteten er null. Det assosierte kapillærtrykket for spontan vannfylling av porehalser viste god korrelasjon med det målte kapillærtrykket når oljens relative permeabilitet ble null under vannfløming. Korrespondansen indikerer at den kritiske perkoleringssansynligheten er den dominerende faktoren for når oljefasen mister forbindelsen over systemet. Distribusjonen av oljeansamlingsstørrelsene etter vannfløming ble også undersøkt. Vi observerte ingen sammenheng mellom porestruktur og størrelsene av oljeansamlinger.

Table of Contents

List of Tables

List of Figures

Nomenclature

β	=	Corner half-angle
γ	=	Interfacial tension
ΔP	=	Pressure difference
∇p	=	Pressure gradient
θ_A	=	Advancing contact angle
Θ	=	Angle
κ_i	=	Location term of element i
κ_l	=	Location term of link l
μ	=	Fluid viscosity
ν	=	Fluid velocity
σ	=	Conductance
σ_w	=	Water conductance
σ_o	=	Specific rock conductance
τ	=	Tortuosity
A	=	Cross-sectional area
B_c	=	Zp_{cb}, where Z is the coordination number and p_{cb} is the bond percolation threshold
C	=	Constriction factor
$C(S)$	=	Constriction factor of streamline S
C_s	=	Hydraulic constriction factor
C_{Ip}	=	Coefficient related to pore geometry
d	=	Dimension
$f(X_i)$	=	Volume fraction of oil trapped in clusters of volume X_i
F	=	Formation factor
g_i	=	Conductance of element i
g_{IJ}	=	Conductance of link IJ
g_h	=	Hydraulic conductance
g_e	=	Electrical conductance
G	=	Shape factor
I	=	Current
k_{ro}	=	Relative oil permeability

l_s	=	Streamline length
L_e	=	Length of curve
L	=	Length
L_i	=	Length of element i
L_{IJ}	=	Length of link IJ
N_i	=	Number of clusters of size X_i
N_{cap}	=	Capillary number
p	=	Percolation probability
p_i	=	Pressure in element i
p_a and p_b	=	Intra-link pressures
p_{cb}	=	Bond percolation threshold
P_c	=	Capillary pressure
P_c^s	=	Threshold capillary pressure for snap-off
P_{cR}^I	=	Capillary pressure ratio between pore filling and snap-off
$P(X_i \leq X_j)$	=	Cumulative fraction
q_{IJ}	=	Flowrate in link IJ
$q(S)$	=	Flowrate in streamtube S
Q	=	Flowrate
r_t	=	Radius of throat
r_p	=	Radius of pore
r_{insc}	=	Inscribed radius
R	=	Radius
S_{or}	=	Residual oil saturation
V	=	Voltage
V_{tot}	=	Total volume
w_g	=	Conductance weighting factor
X_i	=	Cluster-size (volume)
Z	=	Coordination number

DRP	=	Digital Rock Physics
CT	=	Computed tomography
BSE	=	Back-scattered electron
MA	=	Medial axis
MB	=	Maximum ball

Chapter 1

Introduction

The understanding of fluid displacement through porous media is relevant for solving many problems, both industrial and scientific. Pore-scale techniques can be used to study capillary trapping and dissolution of CO_2 - processes which are regarded as promising to effectively and safely store CO_2 in order to reduce emissions to the atmosphere. The technology can also be used within fields such as hydrology and radioactive waste disposal (Blunt et al., 2013). Even though pore-scale technology can be useful within a wide range of industries, the applications within the petroleum industry will be the scope of this thesis.

End-point saturation is a first order uncertainty in reservoir modeling and simulations. The economic gain associated with even a slight increase in recovery is huge, and obtaining reliable estimates for the residual oil saturation is therefore essential. Studying the factors which determine and influence the distribution and magnitude of the residual oil is crucial for predictive reservoir modelling.

Residual oil saturation is commonly determined from core analysis. However, there are many factors affecting the accuracy of such measurements. The core samples are commonly cleaned and the wettability is reestablished by aging prior to the experimental determination of the residual oil saturation. This might alter the wettability of the core, yielding a misrepresentation of the wettability in the reservoir rock. In addition, the residual oil saturation is dependent on how the core is recovered. In conventional coring, the core might expulse fluids during transportation.

Performing numerical simulations on digitized rock samples can be be both time and cost

saving. In addition, the uncertainty related to the assumption of sister-plugs can be eliminated. This thesis is based on the application of pore network models. Such pore network models can easily capture wetting layer flow, which has been proven essential for the prediction of end-point saturations (Berg et al., 2017).

Advanced techniques, such as micro-tomography and back-scattered electron imaging, can be used to capture images of rock samples (Berg et al., 2017). Network representations of the pore space can be numerically extracted. From these digital representations the pore structure can be assessed and it would be valuable to be able to predict the residual oil saturation based on descriptors of the pore microstructure.

After waterflooding oil is trapped within the pore space. The confined oil can be trapped in single pores or form clusters, spanning multiple pores and throats. Trapping of oil is highly dependent on the pore structure (Yuan (1981), Sahimi (2011), Tanino and Blunt (2012), Nie et al. (2016)), and might lead to significantly reduced oil recovery (Blunt, 2017). Snap-off is a common trapping mechanism, especially in water-wet systems, and it is dependent on the pore structure. Chatzis et al. (1983) presented results for two dimensional regular lattices which indicated that the size and distribution of trapped oil clusters are linked to the pore structure of porous media. In this thesis, pore network modelling is used to investigate the correlation between pore structure and residual oil saturation at the capillary limit.

Several authors (Melrose and Brandner (1974), Larson et al. (1981)) have pointed out the resemblance between percolation processes and hydrocarbon trapping. Assuming water-wet conditions and wetting phase connectivity, throats throughout the rock can be snapped-off. The snap-off trapping mechanism in real porous media is therefore similar to the random closing of bonds in a percolation process. By applying a bond percolation process to a network model, the radius of the link disconnecting the non-wetting phase between the inlet and the outlet can be found. This radius can be associated with a threshold capillary pressure for snap-off, which should correspond to the recorded capillary pressure at $k_{ro} = 0$ during waterflooding. When the oil phase becomes discontinuous the relative oil permeability becomes zero. The oil-phase connected to the outlet can still be produced. For smaller systems, such as pore network models and possibly even core samples, this incremental oil production after $k_{ro} = 0$ could be significant. The magnitude of this effect can not be directly applied to the field scale, and might result in a potential mismatch with residual oil saturations estimated from smaller systems.

In this thesis a suite of scripts are developed to perform simulations and calculate different parameters from network models. The scripts cover a wide range of tasks, such as pressure and flowrate calculations, streamline tracking, percolation processes, network manipulation and visualization. The scripts are used to calculate and compare different flow and geometry based pore structure descriptors, such as coordination number, constriction factor, aspect ratio, and porosity, with the residual oil saturation at different wetting conditions. The streamline script is used to calculate effective descriptors, such as the constriction factor, along the conducting pore space. Local measures of pore structure can also be calculated with the developed scripts.

Structure of the Report

Chapter 2 is mainly based on work conducted in the specialization project, and presents a brief literature review of Digital Rock Technology and pore network modelling. Chapter 3 presents the theoretical framework which forms the basis for the implementations in the developed scripts. The governing theory of pore network models is briefly explained, and the theory behind the pore structure measures, trapping mechanisms and percolation processes are presented. Chapter 4 presents the suite of scripts developed in this thesis. This chapter is meant to link the theory with the implementation. The implemented equations are derived and the methods of implementation are explained. The different network models and the software used in this study are also presented. In Chapter 5 the obtained results are presented and discussed. Chapter 6 presents the main conclusions and recommendations for further work.

Chapter 2

Literature Review

This thesis is mainly focused on the development of several pore-scale simulation tools and the results they yield. A comprehensive literature review is therefore not conducted, but a brief introduction to Digital Rock Physics (DRP) and especially pore network modelling is given in this chapter.

2.1 Digital Rock Physics

DRP combines the methods of imaging and analysis of the pore space of a rock sample. By constructing digital 3D-models of the porous medium, different properties, such as permeability and electrical conductivity, can be estimated through simulation of the physical processes (Andrä et al., 2013). The use of DRP has increased over the last decade, as the technology is advancing and necessary equipment has become more available (Wildenschild and Sheppard, 2013).

There are mainly two different methods of performing simulations on digital representations of porous media. The most computational demanding is direct simulation on a grid representation. In this method the simulations are conducted on binarized images of the pore structure, thus the original geometry of the rock is maintained. Several different simulation techniques with varying capabilities exist, and the most popular to date is the lattice-Boltzmann method (Blunt, 2017). This particle-based method approximates the Navier-Stokes equation, and is fairly simple to implement (Berg et al., 2017).

The work conducted in this thesis is based on the other method, pore network modelling, where physical processes are simulated on a network representation of the void space.

2.2 Pore Network Modelling

The highly complex and chaotic nature of porous media is simplified in a pore network model. The void space of the rock matrix is represented by a network of throats, narrow passages which through the fluid flow, and pores, larger voids where the throats meet. Pore network representations of different rocks are illustrated in Figure 2.1. The surface of the pores and throats can be highly irregular and rough, and can often be composed of different materials. Due to the simplification, these details of the void space is not accurately represented in such a network, but the level of details needed is dependent upon the application. When studying the primary drainage and the associated trapping of hydrocarbons in porous media, the topology of the pore space becomes dominating (Sahimi, 2011). The topology describes the overall connectivity of porous media. This information can be described by a network of pores, and therefore makes network models applicable to study end-point saturations.

The simple description of the pore space in a network model results in computationally efficient simulations compared to grid-based simulators. The pore network model can be applied to any scale which gives it practically infinite resolution. As long as the fine details of the porous medium is observed by imaging or other experimental techniques, they can be included in the network model (Xiong et al., 2016).

2.2.1 The History of the Pore Network Model

The pore network model was introduced when Fatt (1956) used a lattice of resistors to represent the porous media. Fatt (1956) took advantage of the analogy between Poisuille's law and Ohm's law and were able to calculate capillary pressure and relative permeability curves that were closer to experimental measurements than the bundle of tubes model (Valvatne, 2004).

In 1977 Chatzis and Dullien (1977) created a three-dimensional model, as they found out that the two-dimensional model proposed by Fatt (1956) lacked spatial interconnectivity and could not accuratley predict flow in three-dimensions. They also noticed that the

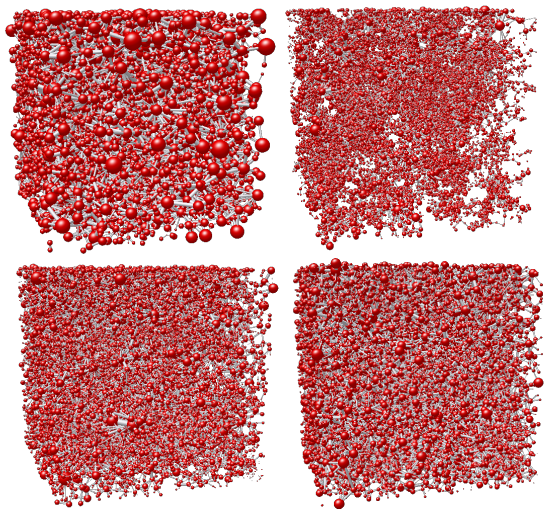

Figure 2.1: Visualizations of the network representations of different rocks. Top row: Synthetic silica and Estaillades. Bottom row: Bentheimer and Fontainebleau. The networks have total porosity of 42.9%, 12.7%, 21.7% and 24.5% respectively. The balls represent pore bodies and the sticks represent pore throats.

breakthrough time was dependent on the coordination number.

Another major advance came when Bryant and Blunt (1992) mimicked real porous media by packing of equal spheres. They were successfully able to predict relative permeability, which closely matched existing experimental results. The predictive abilities of their more realistic models was a major breakthrough in pore-scale modeling (Blunt et al., 2002). However, the application of the model was restricted to simple porous media with grains of more or less equal size.

A similar method was proposed by Bakke et al. (1997). They extended the method with randomly sized spheres, and included numerical modelling of compaction and diagenese. They based the reconstruction process on grain size distributions extracted from representative thin-sections (Blunt et al., 2002). This step-wise reconstruction technique yielded geologic realism and more representative connectivity compared to stochastic models based on correlation functions (Blunt et al., 2002).

Pore network modelling has emerged in complexity and the technology is still rapidly advancing, driven by practical application and advances in imaging techniques (Blunt, 2017). Pore network models can be divided into two categories. Dynamic models are

able to model viscous forces by taking time into consideration, while in quasi-static models all the fluid-fluid interfaces remain static at a given saturation step (Valvatne, 2004). Today simulations can be performed on irregular lattices and they allow both randomly distributed wettability and flow in wetting-layers (Blunt et al., 2013).

2.2.2 Imaging and Construction of Pore Network Models

The first process of DRP is to obtain a digital representation of the porous medium. Advanced techniques, such as micro-tomography (micro-CT) and back-scattered electron (BSE) imaging, is used to either capture 3D and 2D images respectively. Thin sections can also be used to reconstruct 3D-models. Reconstructing a representative 3D-model is crucial in order to obtain reliable properties from the rock. Porosity, grain and pore size distribution, mineralogy and clay content can be found by analysis of 2D thin sections. From 3D models effective properties like relative permeability and conductivity can be estimated (Berg et al., 2017).

There are several imaging and reconstruction techniques available, but as this thesis is focused more on pore-scale simulation, only a brief overview of the most common methods will be given. Three different reconstruction methods are described; experimental reconstruction from micro-CT, statistical and process-based methods. Mercury intrusion porosimetry and gas adsorption are other non-destructive methods that can be used in the characterization of the pore space (Xiong et al., 2016).

X-ray Computed Tomography

X-ray computed tomography is an entirely non-destructive technique to obtain a 3D-image of a rock sample. The principle of the method is to take multiple X-ray images from different angles in order to reconstruct a three-dimensional image (Berg et al., 2017). This produces a gray-scale image, like illustrated in Figure 2.2, where the color is proportional to the X-ray attenuation value of the material (Andrä et al., 2013). These values can be linked to the density of the material, which enables the distinction between matrix, clay and air filled pore space. Air has a low density and will therefore be represented by darker colors, while the rock matrix yield brighter colors. There are mainly three different micro-tomography systems: medical CT, micro-CT and synchotron micro-tomography, which all vary in resolution. Medical CT scanners are frequently used for core inspection, but they are not suited for DRP as their resolution is 200-500 μm. Micro-CT scanners, with a resolution down to approximately 1 μm, are readily available and more commonly used for this application (Berg et al., 2017). Some industrial systems can obtain resolution

better than 100 nm (Wildenschild and Sheppard, 2013). Synchotron based micro-CT systems can achieve similar resolution as micro-CT scanners, however at a much shorter acquisition time (Berg et al., 2017). Synchotrons can therefore be used to capture transient phenomena.

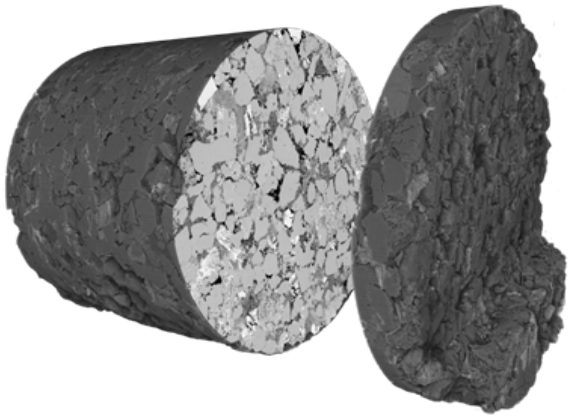

Figure 2.2: An example of a micro-CT image of a rock sample (FEI, 2018).

2D-to-3D Reconstruction

Another method of obtaining a three-dimensional representation of a porous medium is by using stochastic methods applied to thin sections (2D-images). This method is based on using statistical information, often correlation functions (Berg et al., 2017), obtained from analysis of 2D-images of thin sections. This method will not accurately recreate the heterogeneous features of a porous medium, however it gives the opportunity to generate multiple models with similar structural properties (Andrä et al., 2013).

Computer Constructed Models

It is also possible to generate digital porous media models based on measurements of grain size distribution, porosity and other rock properties found from thin section analysis. The software used in this thesis, e-Core, has such capabilities. In addition to creating sphere packs, the software can model both compaction and diagenesis. This step-based method was described by Bakke et al. (1997), and consists of numerical modeling of grain sedimentation, followed by compaction and diagenesis. The first step is based on information obtained from thin section analysis, where the diameter of all grains are measured and a

grain size distribution is determined. The compaction process is modelled by a downwards vertical shift applied to every grain in the model, and mimics the overburden stress applied to the porous medium over time. In the last step, diagenese, quartz overgrowth and clay coating are modeled.

2.2.3 Pore Network Extraction

Pore networks can be extracted directly from experimentally or numerically generated 3D images. The constructed models have to reflect the essence of the topology and geometry of the pore space. It is also important that geometric properties such as shape and size distribution of pores and throats are adequately represented (Xiong et al., 2016).

Some of the grain-based methods (Bakke et al. (1997); Bryant and Blunt (1992)) have been discussed in Sections 2.2.1 and 2.2.2. There also exist several non grain-based methods, where different algorithms are applied to locate the pores and throats from representations of rock samples. Two of the most used methods are the medial axis algorithm (MA) and the maximum ball algorithm (MB).

In the MA method the pore space is burned by using different algorithms until the ”spine” of the object is positioned approximately in the middle of the pore channels (Lindquist et al., 1996). The pores are then defined as the junctions and the throats as the branches between them (Xiong et al., 2016). Instead of eroding the pore space, the MB algorithm locates the largest spheres that can fit in the pore space. Only the largest spheres are kept, and the smaller spheres, confined inside the larger ones are removed. The remaining spheres are called maximum balls, where the largest of them defines the pores. The smallest balls located between the largest balls are defined as throats (Dong and Blunt, 2009).

The different methods will extract different networks with different properties. Dong et al. (2008) found that the networks constructed with the MA method contained more isolated and dead-end pores compared to the MB method. The MA method also experienced difficulties in constructing a realistic network for a sample with a high degree of noise. Dong et al. (2008) also found that the permeability tended to be over-predicted, and that the under-estimation of tortuosity of the networks resulted in an under-estimation of the formation factor.

2.2.4 Limitations of Pore Network Modelling

While network modeling is proven to be a powerful tool, there are also limitations to consider. The difference in scale, from the whole field down to the smallest pore is huge. Porous media are heterogeneous, and the uncertainty associated with applying pore-scale results to the field scale may be large. As for conventional core analysis, the problem of upscaling is also an issue for digital rock physics.

In pore network modeling the fluid distribution is determined by contact angles. Obtaining a representative estimation for these properties for the full range of minerals in a certain rock sample is nearly impossible. This problem is partly solved by experimentally estimating a range of contact angles, and then assigning a value to each network element from a statistical distribution. The contact angle can be measured by a variety of methods. In one of the most common methods an image of a liquid drop resting on a plane solid surface is captured, and the contact angle between them is measured directly from the image. In pore network models the geometry of the pore space is simplified, which can make simulation results less credible (Xiong et al., 2016).

Chapter 3

Theory

3.1 Pore Network Modelling

3.1.1 Fundamental Theory

The ratio between viscous and capillary forces is quantified by the capillary number, N_{cap}, which is defined as (Øren et al. (1998), Valvatne (2004)),

$$N_{cap} = \frac{\mu\nu}{\gamma} \tag{3.1}$$

where γ is interfacial tension, μ is a fluid viscosity and ν is a fluid velocity. In pore network modelling the viscous forces are assumed to be negligible, and the flow regime is dominated by the capillary forces. This assumption is valid for processes with low capillary number (10^{-6} or less), and implies that only a single element is filled at a time (Blunt et al., 2002). There are several examples where this approximation is not reasonable, i.e. near wellbore flow or flow in fractures, however in this thesis it is assumed valid.

3.1.2 Shape Factor

An important feature of porous media is the presence of angular corners and crevices that allow the simultaneous flow of multiple fluids. The wetting layers have a small effect on the total saturation, but they enable wetting phase connectivity at low saturations (Valvatne, 2004). By modeling the pores and throats as non-circular shapes, like triangles or squares, wetting layers can be accounted for (Øren et al., 1998). It is not possible to encompass the exact shape of the pores and throats, but the cross-sectional area, A, and perimeter length,

P, can be preserved quantitatively through the dimensionless shape factor G (Mason and Morrow, 1991), which is defined as,

$$G = \frac{A}{P^2} \tag{3.2}$$

The shape factor is measured by image analysis, using averaged properties along the

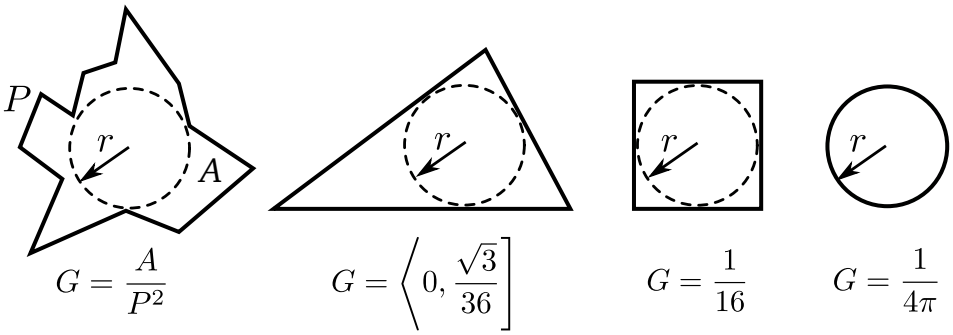

Figure 3.1: The shape factor is constant for squares and circles, but can vary depending on the shape of a triangle. Adapted from Valvatne (2004).

pores and throats (Valvatne, 2004). Figure 3.1 illustrates the shape factors of elements of different cross-section. The shape factor decreases as the pore space becomes more irregular. This results in smoother elements, which have a higher shape factor, being modelled as circles. Rougher elements are associated with lower shape factors and are modelled as triangles. A circle has area $A = \pi R^2$ and perimeter length $P = 2\pi R$ yielding,

$$G_{circle} = \frac{\pi R^2}{(2\pi R)^2} = \frac{1}{4\pi} \tag{3.3}$$

Both circles and squares have constant shape factors and using the same procedure as above for a square results in $G_{square} = 1/16$. For triangles the shape factor decreases as the triangle becomes more slit-like, as illustrated in Figure 3.2. A triangle has a maximum shape factor when its side-lengths are equal i.e. an equilateral triangle. For an equilateral triangle with side length l, the area is $A = l^2 sin(\pi/3) = l^2\sqrt{3}/4$ and perimeter length $P = 3l$, yielding,

$$G_{triangle,max} = \frac{\sqrt{3}l^2}{4(3l)^2} = \frac{\sqrt{3}}{36} \tag{3.4}$$

Most pores and throats are irregular, and when extracting the networks from the models used in this thesis it was found that almost every network element was modelled as trian-

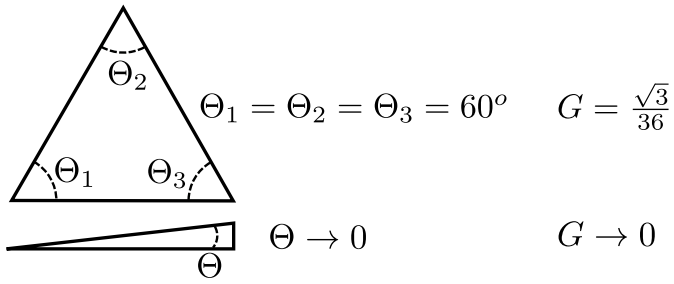

Figure 3.2: As the triangle becomes more slit-like the shape factor decrease from the maximum value of $\sqrt{3}/36$.

gles by e-Core. Therefore, to ease the implementation every network element is modelled as triangles.

3.1.3 Corner Half-Angles

The corner half-angles, β, are not uniquely defined for a triangle solely given by a shape factor. Patzek and Silin (2001) proposed a method to back calculate the corner half-angles from the shape factor. Consider a triangle with an inscribed circle touching each of the three bases. Three lines can be drawn from the center of this circle to each of the vertices, while three lines connect the circle's center perpendicularly to each of the bases, as illustrated in Figure 3.3. The area of the triangle, composed of six smaller triangles, is given by $A = \frac{1}{2}Pr_{insc}$. Thus, the inscribed radius, r_{insc}, can be given in terms of area and perimeter (Mason and Morrow, 1991),

$$r_{insc} = \frac{2A}{P} \tag{3.5}$$

By scaling r_{insc} with P, Eq. 3.5 can be rewritten as,

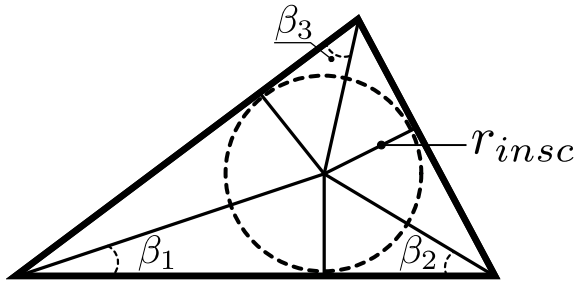

Figure 3.3: The cross-section of a triangular pore and the corner half angles. Adapted from Patzek and Silin (2001).

$$r_{insc}^2 = 4GA \tag{3.6}$$

Using simple trigonometry, the area of each (of the six) triangle, A_i, can be expressed by the inscribed radius and the corner half angle,

$$A_i = \frac{1}{2} r_{insc} \frac{r_{insc}}{\tan(\beta_i)} \tag{3.7}$$

the total area of the cross-section is given as the sum of the area of the inner triangles by applying Eq. 3.7 to all of them,

$$A = r_{insc}^2 \sum_{i=1}^{3} \frac{1}{\tan(\beta_i)} \tag{3.8}$$

Combining Eqs. 3.6 and 3.8 yields,

$$r_{insc}^2 = 4G r_{insc}^2 \sum_{i=1}^{3} \frac{1}{\tan(\beta_i)} \tag{3.9}$$

Since $\beta_3 = \pi/2 - \beta_1 - \beta_2$ and $\tan(\frac{\pi}{2} - \theta) = \cot(\theta)$, Eq. 3.9 can be rewritten to,

$$G = \frac{1}{4}\left[\frac{1}{\frac{1}{\tan(\beta_1)} + \frac{1}{\tan(\beta_2)} + \tan(\beta_1 + \beta_2)}\right] \tag{3.10}$$

By rearranging Eq. 3.10 we get,

$$G = \frac{1}{4}\left[\frac{\tan(\beta_1)\tan(\beta_2)}{\tan(\beta_1) + \tan(\beta_2) + \tan(\beta_1 + \beta_2)\tan(\beta_1)\tan(\beta_2)}\right] \tag{3.11}$$

By writing the denominator, D, of Eq. 3.11 in terms of sine and cosine, and using the angle sum identity $\cos(\alpha + \beta) = \cos(\alpha)\cos(\beta) - \sin(\alpha)\sin(\beta)$, we get,

$$D = \frac{1}{\cos(\beta_1 + \beta_2) + \sin(\beta_1)\sin(\beta_2)}\left(\cos(\beta_1)\sin(\beta_2) + \sin(\beta_1)\cos(\beta_2) + \frac{\sin(\beta_1 + \beta_2)\sin(\beta_1)\sin(\beta_2)}{\cos(\beta_1 + \beta_2)}\right) \tag{3.12}$$

Using the angle sum identity $\sin(\alpha+\beta) = \sin(\alpha)\cos(\beta) + \cos(\alpha)\sin(\beta)$, we can further simplify Eq. 3.12 yielding,

$$D = \frac{\sin(\beta_1+\beta_2)}{\cos(\beta_1+\beta_2)+\sin(\beta_1)\sin(\beta_2)}\left(1+\frac{\sin(\beta_1)\sin(\beta_2)}{\cos(\beta_1+\beta_2)}\right) \tag{3.13}$$

By rearranging Eq. 3.13, we get,

$$D = \frac{\sin(\beta_1+\beta_2)(\cos(\beta_1+\beta_2)+\sin(\beta_1)\sin(\beta_2))}{\cos(\beta_1+\beta_2)(\cos(\beta_1+\beta_2)+\sin(\beta_1)\sin(\beta_2))} \tag{3.14}$$

Eq. 3.14 simplifies to,

$$D = \tan(\beta_1+\beta_2) \tag{3.15}$$

Finally, by combining Eq. 3.15 and Eq. 3.11 we get,

$$G = \frac{1}{4}\tan(\beta_1)\tan(\beta_2)\cot(\beta_1+\beta_2) \tag{3.16}$$

From Eq. 3.16, Patzek and Silin (2001) derived the formulas for the corner half angles,

$$\beta_1 = -\frac{1}{2}\beta_2 + \frac{1}{2}\arcsin\left(\frac{\tan(\beta_2)+4G}{\tan(\beta_2)-4G}\sin(\beta_2)\right) \tag{3.17}$$

$$\beta_{2,min} = \arctan\left(\frac{2}{\sqrt{3}}\cos\left(\frac{\arccos(-12\sqrt{3}G)}{3}+\frac{4\pi}{3}\right)\right) \tag{3.18}$$

$$\beta_{2,max} = \arctan\left(\frac{2}{\sqrt{3}}\cos\left(\frac{\arccos(-12\sqrt{3}G)}{3}\right)\right) \tag{3.19}$$

$$\beta_3 = \frac{\pi}{2} - (\beta_1+\beta_2) \tag{3.20}$$

Since the corner half angles are not uniquely defined between an equilateral triangle with $G = \sqrt{3}/36$ and a slit-like triangle with $G = 0$, the value of β_2 has to be chosen from the range given by Eqs. 3.18 and 3.19.

3.1.4 Nodes and Links

In a network representation of a porous medium pores and throats are represented by nodes and links. A node is a point in the center of the pore body with associated properties. The geometry of the nodes is described by inscribed radius and shape factor. A throat is a connection between pores and is bounded by the edges of the pores. The link is defined as the collective contribution of two pores and one throat, e.g. the conductance of a link is the harmonic average of the conductance of the pores and throat. In Figure 3.4 the link is illustrated by the three arrows, describing its path form the center of pore I to pore J. The length of the link is therefore the length from the center of pore I, through the throat and to the center of pore J, as can be seen from Figure 3.4. In most cases this distance is longer than the straight line (red stippled line) between the nodes. While inspecting some of the data sets in this project, it was indeed found that the link length was longer than the straight line between the nodes for all non-surface nodes (excluding inlet and outlet pores).

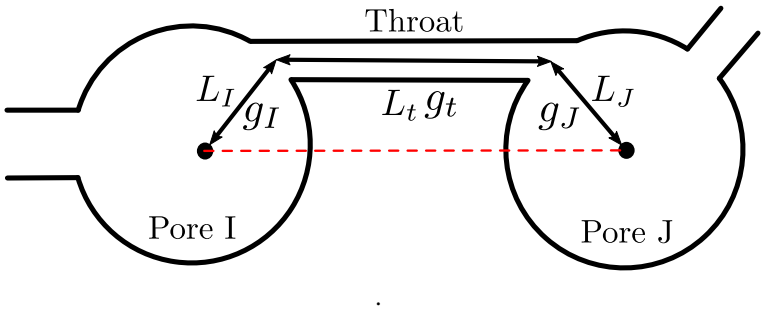

Figure 3.4: Simplified description of two pores connected by a throat in a network model. The lengths and conductances of the pores and throat are L_I, L_J and L_t, and g_I, g_J and g_t respectively. The red line illustrates the straight line between the pore centers. Adapted from Valvatne (2004)

3.1.5 Network Data File Format

The commercial software used in thesis, e-Core, generates several ASCII files to describe the pore network in SI-units. The description of the pores are stored in the `node`-files, while the description of the throats are stored in the `link`-files. This format is also used by Imperial College London, and sometime referred to as the Statoil format. The most used files are described below.

`link1.dat` contains specific information about the throats. The throat's shape factor, length, radius and which nodes the throat connects to is given. `link2.dat` also contains

information about the throats. As described above, a link is composed of a throat and a fraction of two connected pores. The lengths of the link associated with these pores are given in this file. In addition, the volumes and micro-porosity volumes are also included.

The first line in `node1.dat` contains the total number of pores and total length of the system in the x-, y- and z-direction. The successive lines are specific to each of the nodes, containing the coordinates, the coordination number, the indexes of the connected nodes and throats, and if the node is connected to either the inlet or outlet. The geometric information about the nodes is given in `node2.dat`. Each line consists of the specific nodes shape factor, inscribed radius, volume and micro-porosity volume.

Several other data files containing various information about the nodes and links exist. A data file describing the end-point saturation in the nodes, `nodesat_wf.dat`, is used when studying local residual oil saturation and the cluster-size distribution. Another available data file, `link3.dat`, is equivalent to `link2.dat`, but with information specific to electrical potential calculation. This file is not used in this thesis.

A more rigorous explanation is given in Section A.1 in Appendix A. The scripts developed for this thesis are only able to process data files with this specific format.

3.2 Pore Structure

The pore structure of different rocks varies greatly - some rocks are homogeneous and isotropic, while others are highly chaotic, demonstrating great heterogeneity. In this thesis both sandstones and carbonates are evaluated, both displaying different pore structure. Fontainebleau sandstones are clean and non-complex while, the Ketton limestone is almost pure calcite and exhibits a large variation in pore size distribution because of intra-particle porosity.

Linking different measures of pore structure, such as coordination number, constriction factor, aspect ratio and porosity, to flow properties have been attempted by several authors. Chatzis and Dullien (1977) found that the breakthrough time was dependent on the coordination number, where networks with lower average coordination number yielded later breakthrough. Tanino and Blunt (2012) found that residual saturation decreases with decreasing aspect ratio and increases with decreasing coordination number. Nie et al. (2016) proposed that the residual oil saturation in Fontainebleau sandstones are highly correlated

with the aspect ratio between pore and throat radius. Yuan (1981) and Tanino and Blunt (2012) found that the residual saturation decreased with decreasing porosity. Yuan (1981) also found that an increase in porosity yields an increase in average coordination number. According to Sahimi (2011) the coordination number is the dominating property of the porous media when modeling residual oil saturation after primary drainage. He also states that the ratio between the size of the pores and throats (geometry) of the porous media is important.

Some of the most popular measures of pore structure are;

- Coordination number.
- Constriction factor.
- Aspect ratio - both global and local.
- Porosity - total and effective.

In this thesis all of these are calculated from a wide range of networks, with the purpose of finding a correlation with residual oil saturation. The different measures of pore structure are described below.

3.2.1 Coordination Number

The coordination number, Z, is the number of throats connected to a pore. For regular networks in both two and three dimensions, such as illustrated in Figure 3.5, the coordination number is fixed at each node or varies in a periodic manner. However, for networks extracted from real porous media, the coordination number can vary greatly. Jerauld and Salter (1990) states that the coordination number is typically in the range from 3 to 8. The coordination number of some of the networks used in this thesis is presented in Table 3.1. As can be seen, the coordination numbers from real porous media is generally lower than for ideal three dimensional lattices.

3.2.2 Constriction Factor

The constriction factor describes the variation in cross-sectional area and can be considered as a measure of heterogeneity. In a network model the throats can be modeled as circular tubes. For a circular tube with varying cross-section $A(x)$ and length L the variation of cross-sectional area can be measured by the constriction factor, C, by the following

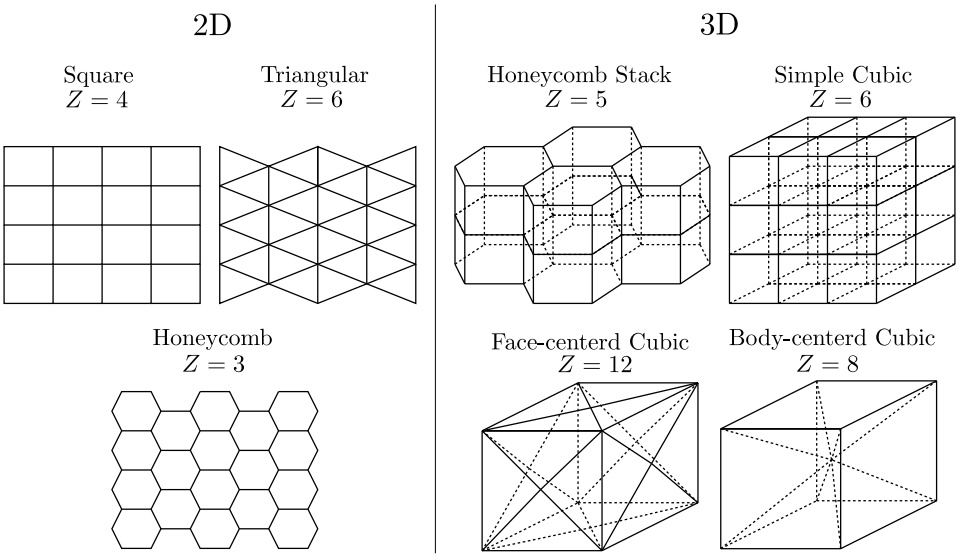

Figure 3.5: Different 2D and 3D network lattices and their coordination number, Z. A unit cell is only illustrated for the BCC and FCC lattices.

Table 3.1: The coordination numbers for some of the networks used in this thesis. Bold numbers are averaged values of multiple networks.

Rock	Z
Fontainebleau	**3.91**
Berea	3.96
Bentheimer	4.41
Doddington	3.99
Estaillades	4.31
Ketton	4.08
Sandstones	**4.07**

relation (Berg, 2014),

$$C = \frac{1}{L^2} \int_0^L A(x)^2 dx \int_0^L \frac{1}{A(x)^2} dx \tag{3.21}$$

The fluid flow is assumed to be approximated by the Hagen-Poiseuille equation, which can be written as,

$$Q = \frac{\Delta P \pi R^4}{8\mu L} \tag{3.22}$$

Where Q is the flow rate, L is the length, ΔP is the pressure difference over L and R is the radius. For a circular pipe, with $\frac{\Delta P}{L} = \nabla p$ and neglecting gravity, Eq. 3.22 can be rewritten to,

$$A^2 = \frac{Q8\mu\pi}{\nabla p} \tag{3.23}$$

The viscosity is constant, and from Eq. 3.23 it can be seen that,

$$A^2 \propto \frac{Q}{\nabla p} \tag{3.24}$$

By combining Eqs. 3.21 and 3.24 we then obtain,

$$C = \frac{1}{L^2} \int_0^L \frac{Q}{\nabla p(x)} dx \int_0^L \frac{\nabla p(x)}{Q} dx \tag{3.25}$$

The fluid is assumed to be incompressible yielding constant flow rate, and Eq. 3.25 is thus simplified to

$$C = \frac{1}{L^2} \int_0^L \frac{1}{\nabla p(x)} dx \int_0^L \nabla p(x) dx \tag{3.26}$$

We want to calculate the constriction factor for the streamlines in the network models. A streamline at any instant is defined as an imaginary curve in a flow field such that it is tangential to the direction of the instantaneous velocity. The streamlines cannot intersect, and in steady-state flow the streamlines will be fixed. A streamtube is bounded by streamlines and have an associated constant volumetric flowrate. In a network model, all streamlines in a given streamtube will be identical. As proposed by Berg (Berg (2012), Berg (2014)), the pressure gradient, in Eq. 3.26, is replaced by the the pressure derivative $\delta p/\delta s = \nabla p \bullet \mathbf{u}/u$ along the streamline, yielding the constriction factor, $C(S)$, for a single streamline as,

$$C(S) = \frac{1}{l_S^2} \int_S \frac{u}{\nabla p \bullet \mathbf{u}} ds \int_S \frac{\nabla p \bullet \mathbf{u}}{u} ds \tag{3.27}$$

where l_S is the streamline length.

3.2.3 Aspect Ratio

The aspect ratio is defined as the ratio of the pore radius to the throat radius (Dong, 2008). In this thesis the global and two different local aspect ratios have been validated, using the inscribed radii given in the network data files. The global aspect ratio is the ratio

between the average pore size and the average throat size. Two types of local aspect ratio is found. The minimum local aspect ratio is the ratio between the pore radius and the largest bounding throat radius, while the average local aspect ratio the ratio between the pore radius and the average of the bounding throat radii.

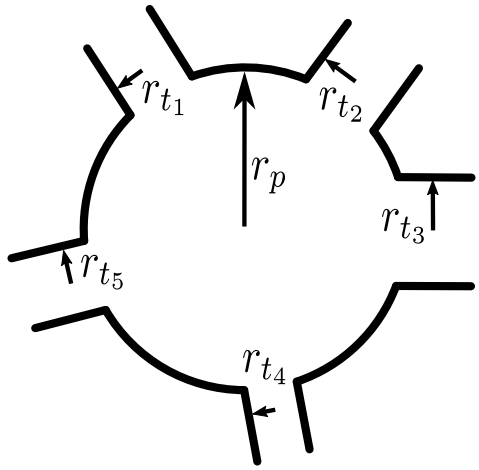

Figure 3.6: A pore bounded by five throats. The average local aspect ratio is $r_p/r_{t_{avg}}$, while the minimum local aspect ratio is r_p/r_{t3} since r_{t3} is the largest bounding throat radius.

3.2.4 Tortuosity

Tortuosity, τ, is a measure of the elongation of streamlines with the respect to the length of the system they span, and can be defined as (Carman, 1956),

$$\tau = \frac{L}{L_e} \tag{3.28}$$

where L is the distance between the ends of a curve with length L_e. In the context of pore network modelling, L_e is the cumulative length the network elements a flow path traverse. Figure 3.7 illustrates flow paths with different tortuosities. For high porosity and well-connected rocks, a high value of tortuosity is expected. While lower tortuosity is common for carbonates, where the flow paths through the rock often are more tortuous. The tortuosity is used to study the differences between streamlines as a result of different implementation techniques.

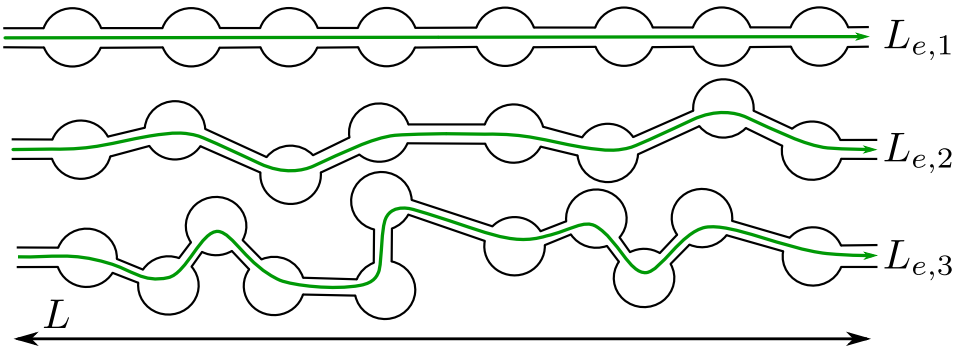

Figure 3.7: The tortuosity increase as the length of the paths (green; $L_{e,1} < L_{e,2} < L_{e,3}$) increase relative to the system length, L.

3.3 Trapping Mechanisms During Imbibition

The process where a reservoir is filled by oil is known as the primary drainage, and can span over millions of years. Because of the duration of this process it is reasonable to assume that the phases have established capillary equilibrium, and the interfacial curvature between them is constant. This is valid for the curvature in the corners and crevices, arc menisci (AM), and terminal menisci (TM) crossing a pore or throat (Blunt, 2017). Figure 3.8 illustrates the fluid distribution of wetting and non-wetting phase at a cross-section of a pore or throat.

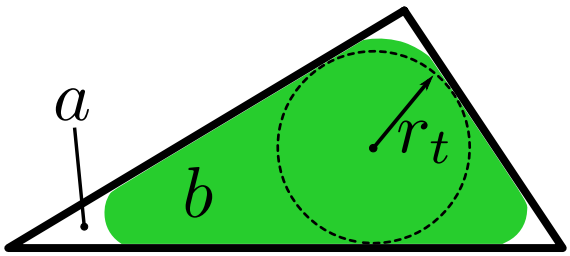

Figure 3.8: Wetting phase (a : white) and non-wetting phase (b : green) at a cross-section of a throat. We can see arc menisci in the corners and the inscribed radius of the throat, r_t.

During waterflooding, water is injected into the reservoir and oil is displaced. This process induce a moderate increase in water pressure and an associated decrease in capillary pressure. For a water-wet sample, the oil is primarily distributed in the center of the the pore bodies, surrounded by layers of water in the corners as illustrated in Figure 3.8. When the capillary pressure decreases, mechanisms like bypassing and snap-off can trap oil, and thus

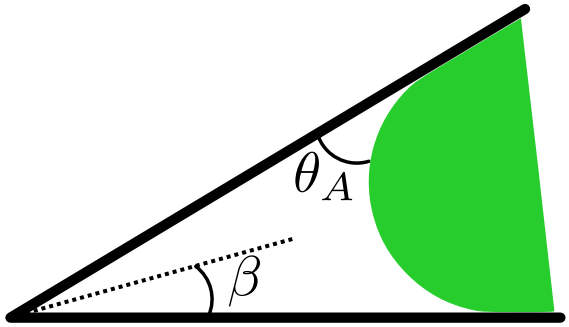

Figure 3.9: Detailed view of the wetting phase in the corner of a pore throat, where the corner half angle, β, and the advancing contact angle, θ_A, are illustrated.

lead to an increase in residual oil saturation (Chatzis et al., 1983). Bypassing describes the phenomena where oil is bypassed when the flow chooses a more favorable path, and thus leaving oil behind.

3.3.1 Snap-Off

As water is injected into the reservoir the water pressure will increase and the wetting layers (a in Figure 3.8) start to swell slowly. The water saturation will increase uniformly everywhere in the porous medium. If displacement of oil by direct pore filling is not possible, the water will continue to swell until the AM's in the corners meet. In a rapid process the water fills the narrow part of the pore throat and two TM's are formed, as illustrated in Figure 3.10. This mechanism is known as snap-off and the water, now present in the throat, can block the displacement of oil. Snap-off can potentially prevent further pore fillings and lead to significantly reduced oil recovery (Blunt, 2017).

For a throat with equal corner half-angles, the threshold capillary pressure at which snap-off occurs, P_c^s, is given by (Blunt, 2017)

$$P_c^s = \frac{\gamma \cos\theta_A}{r_t}(1 - \tan\theta_A \tan\beta) \tag{3.29}$$

where θ_A is the advancing contact angle (the non-wetting phase is being displaced) and r_t is the inscribed radius of the throat, as visualized in Figures 3.8 and 3.9.

After Lenormand et al. (1984) the different pore filling mechanisms are named after how

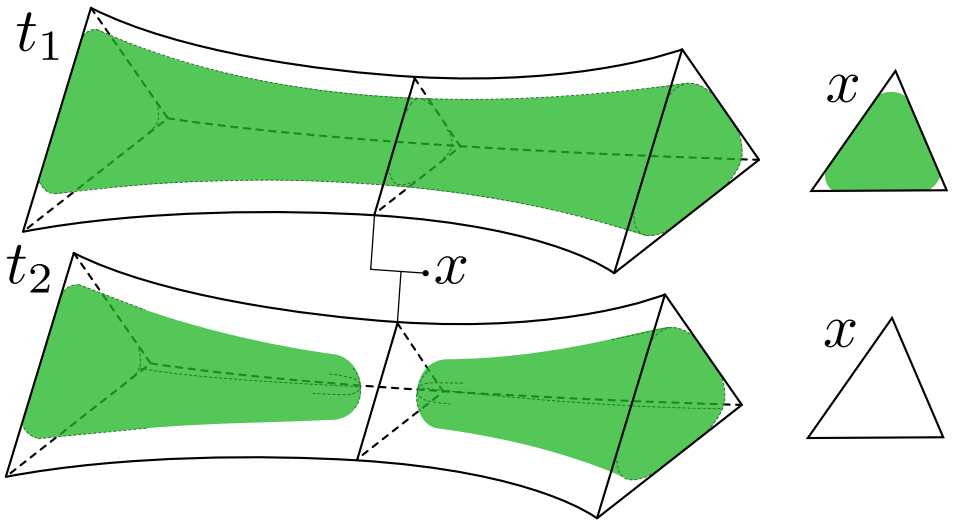

Figure 3.10: Snap-off in a triangular pore throat and cross-sections of the narrowest part x at times $t_1 < t_2$. At t_1 the oil phase (green) is continuous through the throat, while at t_2 two large bodies of oil are separated.

many oil-filled throats are connected to a certain pore. They are labeled I_n where n is the number of oil filled throats. An I_n mechanism will generally occur at a higher capillary pressure than an I_{n+1} mechanism, because of the critical radius of curvature needed to fill the pore (Blunt, 2017). The most favored drainage displacement mechanism is therefore I_1, which has an approximate threshold capillary pressure, P_c, given by (Blunt, 2017),

$$P_c = \frac{C_{Ip}\gamma \cos\theta_A}{r_p} \tag{3.30}$$

where C_{Ip} is a coefficient (related to pore geometry and contact angle), with numerical value ranging from 1 to 2, and r_p is the inscribed radius of the pore. Snap-off will only occur if wetting-layers are present and if it is the most favored displacement process. It is therefore the ratio of threshold capillary pressure, P^I_{cR}, between piston-like pore filling, I_1, and snap-off which controls the amount of trapped non-wetting phase. Dividing Eq. 3.29 by Eq. 3.30 yields

$$P^I_{cR} = \frac{r_p}{C_{Ip} r_t}(1 - \tan\theta_A \tan\beta) \tag{3.31}$$

In any pore, snap-off in the largest bounding throat determines trapping. The ratio above, for a certain pore, should therefore be considered between I_1 and snap-off in the largest

adjacent throat. If the ratio is larger than 1 snap-off will happen (Blunt, 2017). The variables in Eq. 3.31 - contact angle, aspect ratio and corner half-angle - will influence the amount of trapping. Generally, snap-off is more favored when the aspect ratio is high, the corners of the pore are sharp and the contact angle is small (wetting conditions). The amount of trapping will also be larger for a poorly connected pore space, because the oil will have fewer opportunities to be displaced. Because of the presence of water layers connected across the reservoir snap-off will be the dominating trapping mechanism in water-wet systems (Blunt, 2017).

3.4 Percolation Theory

Percolation theory is a mathematical discipline, which can also be used to describe flow through porous media. Its relevance to modelling of flow in porous media was first recognized in the seventies (Larson et al. (1981), Sahimi (2011)). In the following decades, the applications of percolation theory methods to various porous media problems have increased. Through percolation processes the effect of interconnectivity to flow and transport properties of porous media can be quantified. The theory can also be used in combination with standard experimental methods, such as mercury porosimetry, to correctly interpret the results (Sahimi, 2011).

There are two main methods of percolation - bond percolation and site percolation. These two different methods are illustrated in Figures 3.11 and 3.12. In the context of network modelling, the bonds are equivalent to the throats or links, and the sites are equivalent to the nodes or pores.

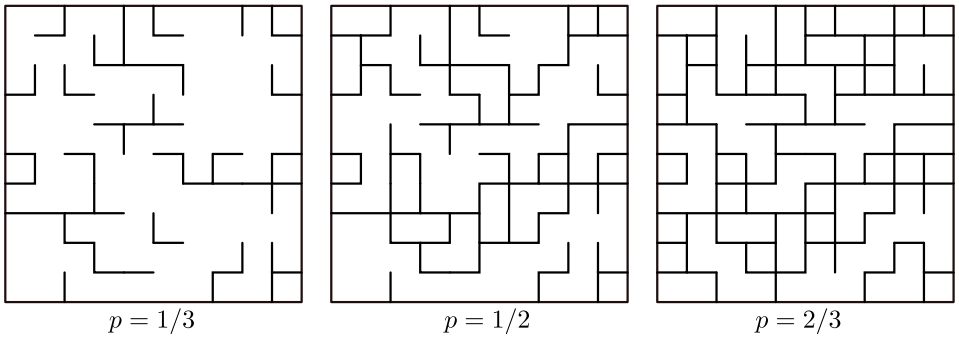

Figure 3.11: Bond percolation in a square network with different values of p. For a infinite square network the bond percolation threshold, p_{cb} is exactly $1/2$.

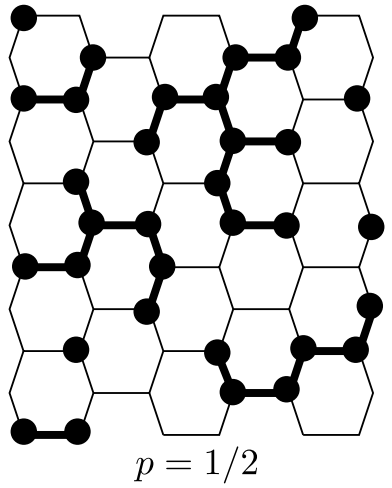

Figure 3.12: Site percolation in a honeycomb 2D network for $p = 1/2$.

In bond percolation the state of the bonds are defined as either open or closed to fluid flow. They are open with a probability p and closed with a probability $1-p$. For large networks, p is equal to a random fraction of all the bonds being open. With this definition, two sites are connected when there is a series of open bonds between them. Site percolation is similar, but in this case the sites are defined as either open or closed. A cluster is defined as a group of connected sites confined by closed bonds, such that no connection to the rest of the network exist. The size of these clusters depends on the value of p (Sahimi, 2011).

3.4.1 Bond Percolation Threshold

The bond percolation threshold, p_{cb}, is defined as the smallest fraction of open bonds necessary to maintain connectivity through the network. At p_{cb} a critical transition of the global connectivity of the network occurs. For $p < p_{cb}$ there is no connected paths of open bonds through the network, while for $p > p_{cb}$ a cluster spanning the whole network exists.

Often, especially for three-dimensional networks, there is no exact value of p_{cb}. Estimates, and some exact values, of the bond percolation threshold for some networks are presented in Table 3.2. An illustration of these networks can be seen in Figure 3.5. Another quantity worth to notice, also presented in Table 3.2, is $B_c = Zp_{cb}$. The value of B_c is dependent on the dimension, d, of the network,

$$B_c \simeq \frac{d}{d-1} \tag{3.32}$$

Table 3.2: Coordination number Z, bond percolation threshold and B_c for different ideal two- and three-dimensional networks (Sahimi, 2011). $(*)$ Exact results.

Network	Z	p_{cb}	B_c
2D			
Honeycomb	3	$1 - 2\sin(\pi/18) \simeq 0.6527*$	1.96
Square	4	$1/2*$	2
Triangular	6	$2\sin(\pi/18) \simeq 0.3473*$	2.08
3D			
Honeycomb Stack	5	0.3093	1.55
Simple Cubic	6	0.2488	1.49
BBC	8	0.1795	1.44
FCC	12	0.119	1.43

thus it should be almost invariant for networks of equal dimensions, with $B_c \simeq 1.5$ for 3D networks. From Table 3.2 it can clearly be seen that the bond percolation threshold is correlated with the coordination number. This is intuitive, as a higher coordination number leads to more redundant paths, and therefore more bonds can be closed while also maintaining connectivity through the system.

3.4.2 Percolation and Snap-Off

During water flooding of a completely water wet system, snap-off can happen in any throat connected through wetting layers. A percolation process where random throats are closed (snapped off) is therefore similar to trapping of oil where snap-off is the dominating trapping mechanism. The closing of a throat is equivalent to it being filled by water. During waterflooding the capillary pressure decreases and oil is displaced. Given that wetting-layers are present throughout the porous medium and snap-off is the favoured trapping process, the throat with the smallest radius will be snapped off first, according to Eq. 3.29. Thus by sorting the links by increasing radius, and removing them in that order, one can find the radius of the link which disconnects the non-wetting phase between the inlet and the outlet. This happens at p_{cb}. If there is little spatial correlation of throat sizes, at least on a larger scale, removing the throats sorted by radius will be similar to randomly removing throats.

Both Melrose and Brandner (1974) and Larson et al. (1981) suggested that a percolation process can be used to model entrapment of a fluid phase in porous media. This is investigated in this thesis, and the process is described in the following chapter.

Chapter 4

Method

To preform calculations, solve various problems and produce results from the pore network models a suite of scripts have been written in Python. The scripts are made as simple and user friendly as possible, but the author recognizes, not being a data scientist, that they neither are optimal nor perfect. As always, improvements and changes can be done to both increase the computational efficiency and the intuitiveness of the scripts.

4.1 The Suite of Scripts

The scripts perform a wide variety of tasks, such as potential calculation, streamline and cluster tracking, network manipulation, percolation and plotting. The executable scripts are written for a specific task and the naming convention gives an indication of their application. This section is meant to give the reader a brief introduction to the different scripts before explaining the details behind the implementation.

Most of the scripts are executable and require access to auxiliary scripts. The auxiliary scripts contain functions which are used within other scripts (e.g. the importing of network data files) and cannot be executed. The interactions among them can be quite complex. The flowchart in Figure 4.1 gives an overview of the scripts and their dependencies.

The script for calculating the potential and flow rates in a network model was written as a part of the specialization project leading into this masters thesis, but has later been adjusted and improved. The script containing the functions for potential calculations is `laplacePN.py`, which is closely linked to `netPotential.py` which imports the

network data files and executes the functions in the former script.

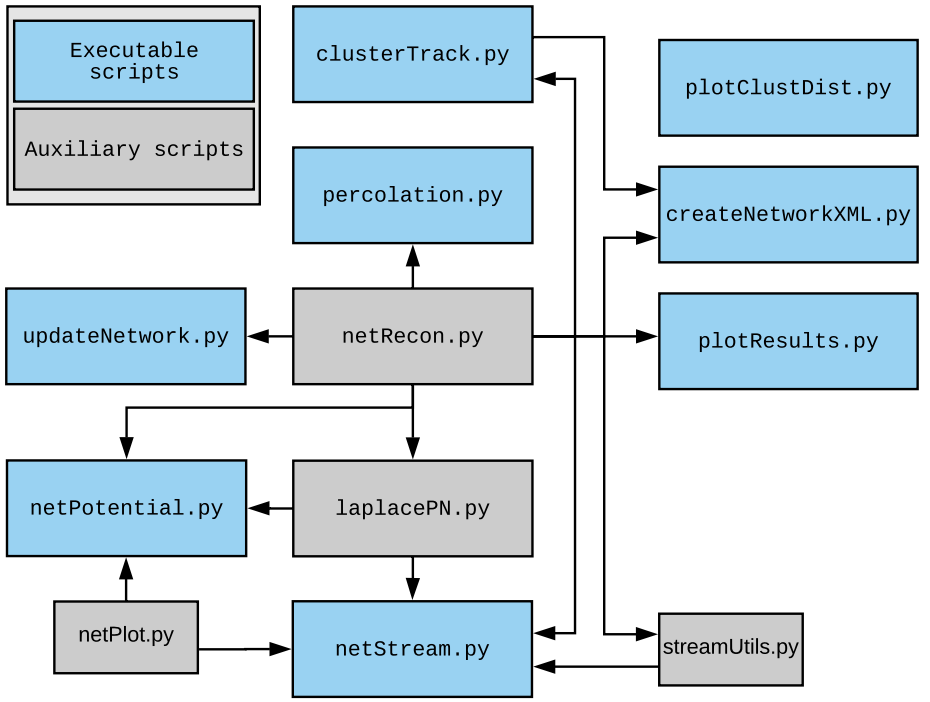

Figure 4.1: The dependencies between the scripts are represented by arrows - e.g. `updateNetwork.py` is dependent on `netRecon.py`. Most of the *executable scripts* are dependent on one or several auxiliary scripts. The *auxiliary scripts* contain several help-function which are used within other scripts. Most of the scripts are dependent on `netRecon.py`, because the functions for importing network data files were implemented in this script.

The main script for tracking streamlines is `netStream.py`. Several of the functions used in this script were written in the auxiliary script `streamUtils.py` in order to improve the readability. The pressure field and flowrates are needed for streamline tracking, and functions from `laplacePN.py` is therefore utilized. The streamlines tracked through a network model can be visualized in two dimensions by functions in `netPlot.py`.

The network data files can contain thousands of lines, which limits the possibilities to perform manual inspection and modifications. We therefore saw the need to create a set of functions to automatically perform necessary manipulations to the network data files. These functions can perform a variety of tasks and were collected in `netRecon.py`. The script contains functions for reading, writing and rearrangement of network data files.

In addition to functions for removal of pores with zero coordination number and non-spanning pores and throats. Several other functions, such as calculation of different pore structure measures, were also implemented in this script. The network manipulation functions can be executed by running `updateNetwork.py`.

Functions for conducting bond percolation processes were implemented in `percolation.py`. The bond percolation threshold was found using the bisection method in order to increase the computational efficiency. Functions which estimate the threshold capillary pressures for snap-off were also implemented in this script. Functions written in `clusterTrack.py` locate the different clusters of trapped oil by utilizing saturation data from waterflooding simulations. Cluster-size statistics, such as average and maximum cluster-size, and the cluster-size distribution can be calculated using this script.

To visualize the network models and the simulation results, several different plotting tools were implemented in various scripts. The two plotting scripts, `plotResults.py` and `plotClustDist.py`, were written specifically to visualize the results obtained in this thesis. Functions to visualize the pore networks in both two and three dimensions, and the streamlines going through them, were implemented in `netPlot.py`.

In this chapter the main equations and solving methods are explained and presented. To get further insight, the reader is recommended to take a look at the read-me file and study the code itself. Most of the scripts are attached in Appendix B.

4.2 Solving for Potential by Sparse Matrix Inversion

The potential in each node is solved for by the method proposed by Øren et al. (1998) and Valvatne (2004). The main equations used in the implementation are given in the following sections. When the potential of each node is found, the flow in each link can be calculated, and the permeability or the formation factor in the flow direction can be approximated.

As mentioned, this script was written as a part of the specialization project, but since then several changes and improvements have been made. The original script was solely ”process-based” and the necessary outputs were saved in order to be used for other applications. Now, the the script is ”method-based”, and its functions can easily be used within other scripts.

4.2.1 Mass Conservation

The total flow rate is obtained by solving for the potential in every pore, imposing mass conservation by Kirchhoff's law

$$\sum_J q_{IJ} = 0 \tag{4.1}$$

where q_{IJ} is the flow rate in link IJ, where J runs over the throats connected to pore I. This yields a set of linear equations, where pressure is solved for using the conjugate gradient method. When electric potential is applied the volumetric flow rate, q, is substituted by current, I. The conjugent gradient method (CG) is an effective method for solving linear sets of equations on the form (Shewchuk, 1994)

$$Ax = b \tag{4.2}$$

In our case the coefficient matrix, A, is a square, symmetric and positive-definite matrix. The vector b is known, while x is unknown. The coefficient matrix is sparse, which means that it contains relatively few non-zero entries. Iterative methods applied to sparse systems are often both memory-efficient and quick, and CG is well suited for this purpose (Shewchuk, 1994). For a more thorough explanation of the CG-method the reader is advised to Shewchuk (1994). In the script the `scipy.sparse.linalg.cg`-function is used.

4.2.2 Boundary Conditions

Every pore body on the inlet side of the network is connected by throats to an imaginary inlet pore with constant potential. Similarly, all pore bodies on the outlet side of the network are connected to an imaginary outlet pore (Bakke et al., 1997). The pores which are not located on the inlet or outlet side of the model are considered as impermable boundaries. The inlet and outlet potential can be specified in the script `netPotential.py`.

4.2.3 Hydraulic Conductance

By utilizing the similarity between a pore throat and a long cylindrical pipe, Poiseuille's law can be used to express the relation between the pressure gradient ΔP and the flow rate q,

$$q = \frac{\Delta P}{L}\frac{\pi R^4}{8\mu} \tag{4.3}$$

where L is length of the pipe and R is the radius of the pipe. Poiseuille law is valid for laminar flow, and thus suitable for describing flow in pores (Xiong et al., 2016). The hydraulic conductance, g_h, is analytically given by Eq. 4.3,

$$g_h = \frac{\pi R^4}{8\mu L} \tag{4.4}$$

In order to approximate the network elements as shapes with either circular, rectangular or triangular cross-section we want to express the hydraulic conductance as a function of the shape factor. It is assumed that most network elements have a triangular cross-section, and the shape factor approximation proposed by Øren et al. (1998) is therefore used,

$$g_h = \frac{3}{5}\frac{A^2 G}{\mu} \tag{4.5}$$

The area in Eq. 4.5 is the total area of a network element. Since only the inscribed radius is readily available we need to relate it to the area. According to Mason and Morrow (1991), the relation between area and inscribed radius is given by Eq. 3.6. Combining Eqs. 4.5 and 3.6 yields,

$$g_h = \frac{3}{80}\frac{r_{insc}^4}{\mu G} \tag{4.6}$$

which is the equation used to calculate the hydraulic conductance of all the pores and throats. Since the permeability is not dependent on the viscosity, the viscosity in Eq. 4.6 can arbitrarily be set to 1

4.2.4 Electrical Conductance and Formation Factor

The close analogy between Poseuille's law and and Ohm's law is used to solve for the electric potential. In Eqs. 4.1 and 4.12 pressure and hydraulic conductance is replaced by voltage and electrical conductance respectively (Øren et al., 1998). Ohm's law can be defined as,

$$I = V\sigma \tag{4.7}$$

where I is the current, V is the voltage and σ is the conductance. The rock matrix is assumed insulating and the electrical conductance, g_e of the network elements is therefore

solely dependent on the geometry, and given by,

$$g_e = \sigma_w A \tag{4.8}$$

where σ_w is the conductivity of the fluid. Combining Eq. 4.8 with the relation, Eq. 3.6, by Mason and Morrow (1991) the following equation is obtained,

$$g_e = \sigma_w \frac{r_{insc}^2}{4G} \tag{4.9}$$

Eq. 4.9 is applied to every element in the network model. Employing Eq. 4.7, where the voltage difference is applied over a length L with associated current through the cross-section A, the specific rock conductance, σ_o, of the saturated rock can be expressed as,

$$\sigma_o = \frac{IL}{AV} \tag{4.10}$$

Figure 4.2 illustrate how the pores and throats are modeled. The formation factor is defined by $F = \sigma_w/\sigma_o$. The effective rock conductance is proportional to the conductivity of the fluid it is saturated with, and will not affect the calculation of the formation factor. The conductivity of the fluid is therefore arbitrarily set to one. By combining the definition of the formation factor with Eq. 4.10 the following equation is obtained,

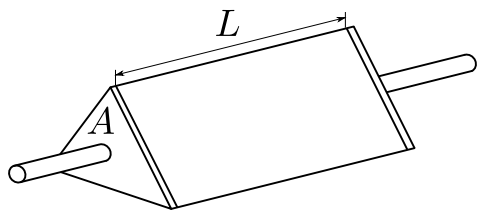

Figure 4.2: A pore network element with applied voltage over a length, L, and current through a cross-section, A.

$$F = \frac{AV}{IL} \tag{4.11}$$

which is the equation used to calculate the formation factor of the network models.

4.2.5 Flow Computation

The flow between pores is assumed to be laminar. The flow rate between pore I and J can then be described by (Øren et al., 1998),

$$q_{IJ} = \frac{g_{IJ}}{L_{IJ}}(p_I - p_J) \tag{4.12}$$

where L_{IJ} is the length between the center of node I and J, p_I and p_J is the pressure in node I and J respectively, g_{IJ} is the conductance of the link, which is defined as the harmonic mean of the conductances of the throat and the associated pore-pair,

$$\frac{L_{IJ}}{g_{IJ}} = \frac{L_t}{g_t} + w_g\left(\frac{L_I}{g_I} + \frac{L_J}{g_J}\right) \tag{4.13}$$

where w_g is the conductance weighting factor, L_t is the length of the pore throat, L_I and L_J is the length of pore I and J respectively, g_I and g_J is the conductance of pore I and J respectively and g_t is the conductance of the throat, as illustrated in Figure 3.4.

In the literature there is not exclusively agreement on how the different conductances should be weighted. Lopez et al. (2003) and Valvatne and Blunt (2004) propose to weight the conductance of the pores and throat equally when calculating the link conductance, while Øren et al. (1998) and Blunt et al. (2002) suggest using $w_g = 0.5$. Applying a weighting factor to the pore conductances can possibly compensate for the fact that the pores are non-cylindrical and that the flow, from one link to another, not necessarily will go through the pore centers. This hypothesis was studied in the specialization project, and it was found that the conductance weighting factor had a notable impact on the results. However, it only shifted the results and it was not possible to conclude which weighting factor should be used for general applications. In this thesis it is therefore chosen to weigh the conductances equally - i.e. applying $w_g = 1$.

Link Conductance at the Boundaries

Since the outlet and inlet pores are imaginary, and therefore do not have an associated conductance, it is necessary to modify Eq. 4.13 to calculate the inlet and outlet link conductance. The imaginary pores have an associated length, which is added to the throat length. For a link connecting the inlet or outlet to a pore, the following equation is then

obtained,

$$\frac{L_{IJ}}{g_{i,IJ}} = \frac{L_t + L_{in/out}}{g_t} + w_g \left(\frac{L_c}{g_c} \right) \tag{4.14}$$

where $L_{in/out}$ is the associated length of the imaginary inlet or outlet pore, and L_c and $g_{i,c}$ is respectively the length and conductance of the pore body connected to the outlet or inlet.

4.3 Tracking Streamlines

In order to calculate local measures of pore structure, a script to track the streamlines through a network model was written. The main script is `netStream.py`, which utilize functions from several other scripts, including `laplacePN.py` and `streamUtils.py`.

4.3.1 Rules to Determine the Path

The streamlines are tracked from inlet to outlet, with several rules which determine the path. A streamline has to be connected to both the inlet and outlet, and therefore cannot progress through nodes which are disconnected from the outlet. This problem is accounted for by removing non-spanning pores and links as described in Section 4.5. Fluid moves from high to low pressure, and a streamline can therefore only traverse through nodes with decreasing pressure. The pressure field is calculated by the method described in Section 4.2. The cumulative flow rate in the streamtubes can not exceed the total flow rate in the link. Because of an average coordination number of $3 - 5$, multiple outlet links can be chosen at each junction (node). In the following section the decision logic is described.

4.3.2 Choosing the Next Link

At each node (or junction) a decision of which link the streamline should proceed to has to be made. In the streamline tracker (`netStream.py`) the next link can be chosen based on either the spatial location of the outlet link or the three-dimensional angle between the inlet link and outlet link. The location-based choosing mechanism was firstly implemented because of its simplicity. Regardless of its less realistic modelling it is not removed from the simulator and remains as an option. To improve the realism and avoid intersecting streamlines, as illustrated in Figure 4.3, it is recommended, and also set to default, to choose links based on the angles between them. The difference between angle-

and location-based streamline tracking is studied, and the results are presented and discussed in Chapter 5.

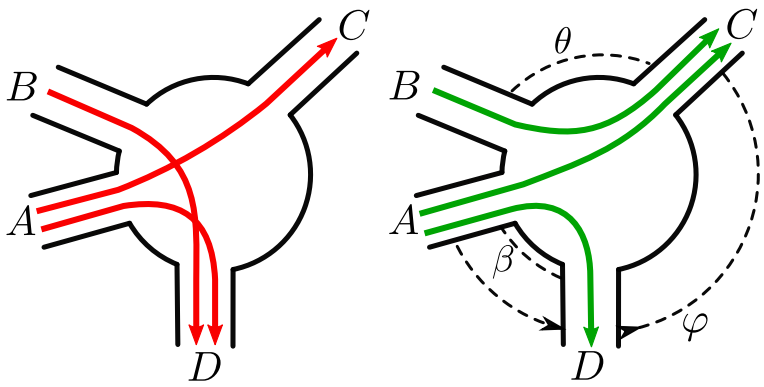

Figure 4.3: The figure to the left illustrates how the next link can be chosen based on spatial location and the resulting intersection of streamlines. In the right figure the next link is chosen based on the angles between inlet and outlet links, resulting in a more realistic flow pattern.

In the example in Figure 4.3, inlet link A has a higher flow rate than the other links. Link B has a lower flow rate than the outlet links C and D. The figure to the left illustrates how the next link can be chosen based on spatial location and the resulting intersection of streamlines. Of the outlet links (C and D), C has the highest location term and is therefore chosen first from A, which is the first inlet link. Its capacity is filled, which results in intersection of streamlines $B \rightarrow D$ and $A \rightarrow C/D$. In the right figure the next link is chosen based on the angles between inlet and outlet links, resulting in a more realistic flow pattern. Since $\beta < \theta < \varphi$, the path $A \rightarrow D$ is first chosen, and the flow capacity of D is reached. Then $\theta < \varphi$ and the path from B to C is chosen. Finally the remaining capacity is filled by $A \rightarrow C$.

Choosing Links Based on Angles

For each node, a set of inlet and outlet links are defined, and the angles between them are calculated by the definition of the dot product of two Euclidean vectors,

$$\theta = \arccos \frac{a \bullet b}{\|a\|\|b\|} \tag{4.15}$$

Vectors a and b are defined as pointing out of the center of the node being validated. The angle θ will then be the angle between the two vectors, as illustrated in Figure 4.4 for

vectors a and b. Starting with the smallest angle, flow rates are then allocated from the inlet links to the outlet links. When the streamlines are tracked in this manner the resulting streamlines will not intersect.

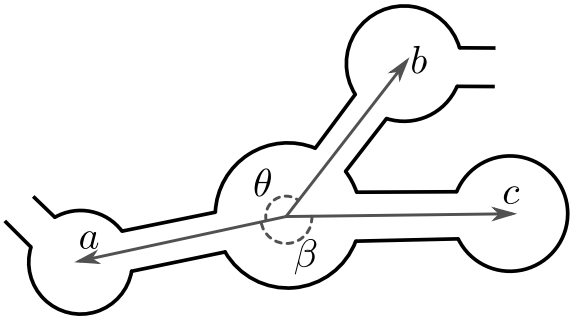

Figure 4.4: Vector a is the inlet link, while vectors b and c are outlet links. The angles θ and β between the inlet link and the outlet links are found by taking the dot product of a and the vectors b and c respectively.

Choosing Links Based on Spatial Location

The other implemented method for choosing which link the streamline should proceed through is by the spatial location of the outlet link. The location term for element i, κ_i, is defined as,

$$\kappa_i = \sqrt{y_i^2 + z_i^2} \tag{4.16}$$

where y_i and z_i are the y- and z-coordinates of node i respectively. The location term of a link, κ_l is defined as the average of the two connected nodes, I and J,

$$\kappa_l = \frac{\kappa_I + \kappa_J}{2} \tag{4.17}$$

The purpose of choosing the next link based on the location is to effectively span the system, while also minimizing the streamline lengths and avoid intersecting streamlines. However, as described previously, and illustrated in Figure 4.3, this might yield intersecting streamlines.

4.3.3 Moving Through the Network

The procedures and logic implemented to track the streamlines in `netStream.py`, can be somewhat complex. The flowchart in Figure 4.5 illustrates the main steps in the script,

and a description is given below. The parenthesized numbers in the text refer to their respective numbers in the figure. Each inlet link (a link conducting flow to a node) has an associated set of potential outlet links, which are stored in a designated list. All these lists are stored in a superior list, `nextLink`, which is used to determine which link the streamline proceeds to from any inlet link [1]. Each of the inner lists are sorted based on either angle or location [2], such that the outlet link at the top of each list is the preferred link. The flowrate assigned to each outlet link is also stored in `nextLink` [3].

At each junction, the preferred outlet link, `nextStep`, is added to the stack of links composing the streamline [4], `stackLinks`, until the streamline can not proceed from the current `nextStep`. The streamline is then tracked to either a dead-end or to the outlet side of the system. If the streamline is tracked to a dead-end, the last chosen `nextStep` is removed as a possible outlet link. If more of the links in the current `stackLinks` make up the same dead-end, they are also removed as possible outlet links [5]. When a streamline is tracked through the model the constriction factor and the tortuosity of the streamline are calculated by Eqs. 4.22 and 3.28 [6]. The flowrate of the associated streamtube and the volume of the nodes and links the streamline consists of are also stored.

The tracking process is repeated until the flow capacity of each link is reached [7]. Then the constriction factor, tortuosity and effective porosity of the system are calculated [8]. By summarizing the flowrates of each streamtube, the total flowrate of the system is found. The fractions of unused nodes and links are also found.

4.3.4 Effective Porosity

A key property of porous media is the effective porosity. The effective porosity is defined as the ratio between the conducting porosity to the total bulk volume (Koponen et al., 1997). The total porosity can be calculated by adding the volume from each pore and pore throat found in network data-files. However, to calculate the effective porosity only the conducting pores and throats must be accounted for. By summarizing the volume of the pores and throats along each streamline the effective porosity is easily calculated.

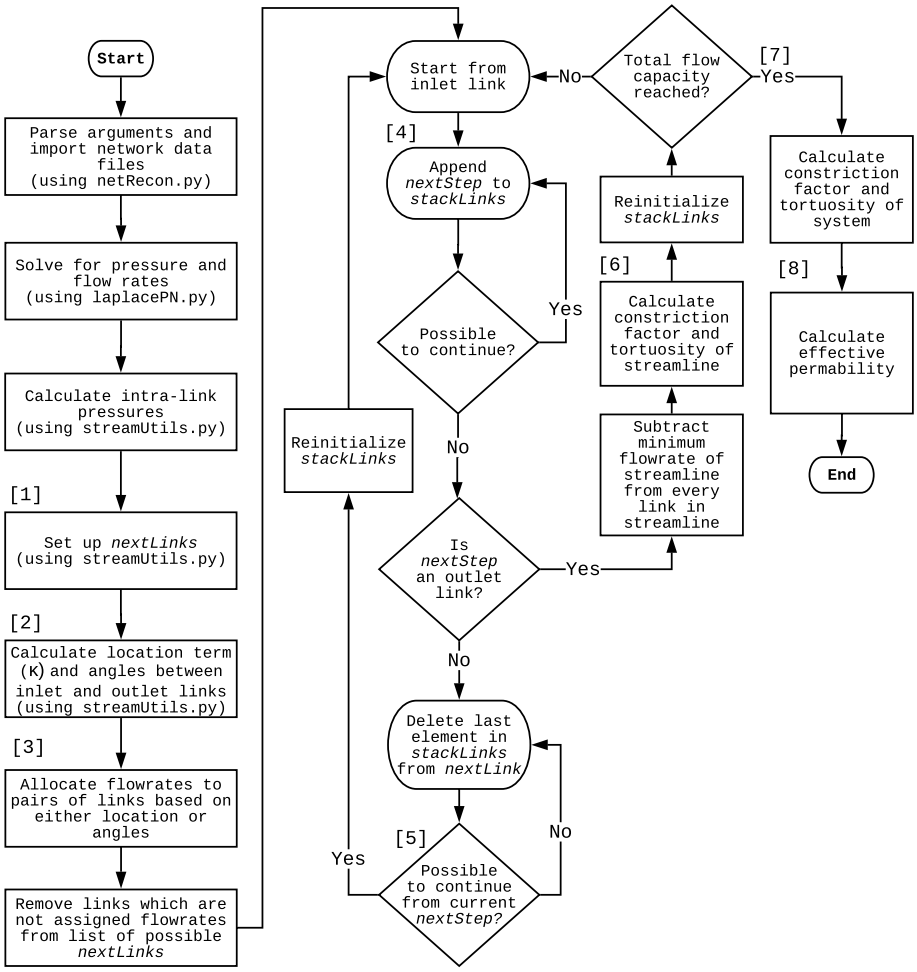

Figure 4.5: The logic and procedures which tracks the streamlines in `netStream.py`. The parenthesized numbers are explained in Section 4.3.3.

4.4 Calculating Measures of Pore Structure

4.4.1 Coordination Number

The network data file `node1.dat` contains information about the coordination number of all the nodes. The coordination numbers are summarized by letting a for-loop run through `node1.dat`. The average coordnation number is found by averageing the cummulative coordination number of the system.

4.4.2 Constriction Factor

In a network model, the pressure gradient and the direction of flow is parallel, simplifying Eq. 3.27 to,

$$C(S) = \frac{1}{l_S^2} \int_S \frac{1}{\nabla p} ds \int_S \nabla p \, ds \tag{4.18}$$

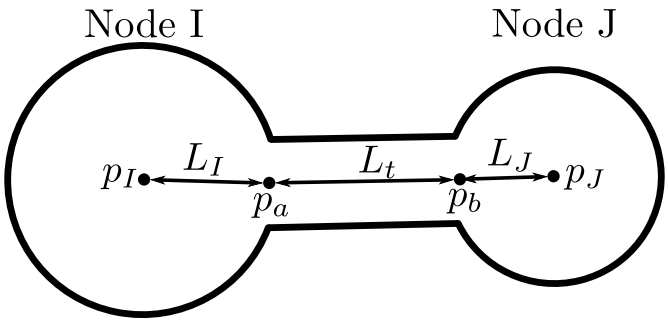

Figure 4.6: The links are composed of a throat and the two connected pores. The intra-link pressures p_a and p_b are defined at the boundary between the throat and the pores. Adapted from Valvatne (2004)

Each streamline consist of a series of links. As illustrated in Figure 4.6, each link can be divided into three parts - one associated with the throat and the two others to the connected pores. Since the flow is incompressible, p_a and p_b, defined in Figure 4.6, can be found by applying Eq. 4.12 to link IJ with flow rate q_{IJ},

$$q_{IJ} = (p_a - p_I)\frac{g_I}{L_I} = (p_b - p_a)\frac{g_t}{L_t} = (p_J - p_b)\frac{g_J}{L_J} \tag{4.19}$$

p_I is known, and p_a can therefore be calculated by,

$$p_a = p_I - \frac{q_{IJ} L_I}{g_I} \tag{4.20}$$

Similarly, p_b is calculated by,

$$p_b = p_a - \frac{q_{IJ} L_t}{g_t} \tag{4.21}$$

where the lengths L_t and L_I, given in `link2.dat`, are the associated lengths of the throat and node I respectively. Since the intra-link elements are the smallest units of measure along a streamline the properties along them are constant, thus Eq. 4.18 can be written as

a sum,

$$C(S) = \frac{1}{l_S^2}\left[\sum_{links} \frac{L_I^2}{\Delta p_1} + \frac{L_t^2}{\Delta p_t} + \frac{L_J^2}{\Delta p_2}\right]\Delta p \tag{4.22}$$

where $\Delta p_1 = p_I - p_a$, $\Delta p_t = p_a - p_b$ and $\Delta p_2 = p_b - p_J$. The constriction factor for each streamtube is calculated by Eq. 4.22. As given by Berg (2014) the hydraulic constriction factor C_s is defined as,

$$C_s = \frac{1}{Q}\int_S C(S) dQ_S \tag{4.23}$$

Since the flow rate in a network model is not infinitesimal, combining Eq. 4.23 with Eq. 4.22 yields,

$$C_s = \frac{\sum q(S)C(S)}{Q} \tag{4.24}$$

where $q(S)$ is the flow rate in streamtube S, and Q is the total flowrate. Eq. 4.24 is used to calculate the constriction factor of the network models.

4.4.3 Aspect Ratio

The global aspect ratio is calculated by a for-loop running through both `node2.dat` and `link2.dat` and storing the radii of all the nodes and links in two separate lists. The global aspect ratio is then found by simply averaging the lists and dividing them by each other.

The local aspect ratios are found by letting a for-loop run through `node1.dat`. For each pore, the radii of the bounding throats are stored in separate lists. To calculate the average local aspect ratio, each pore radius is divided by the average radius of its respective bounding throats. The minimum local aspect ratio is found by dividing each pore radius by the maximum throat radius in its respective list. The local aspect ratios are averaged for the whole network and plotted against the systems residual oil saturation. The minimum local aspect ratio is also plotted against the local residual oil saturation, given in `nodesat_wf.dat`, for every pore with $S_{or} > 0$. These results are presented and discussed in Chapter 5.

4.5 Network Manipulation

Pore networks extracted from real porous media includes all pores and pore throats, regardless if their coordination number is zero or if they are not connected to both the inlet and the outlet of the network (non-spanning).

The removal of dead-end and non-spanning pores and throats will not affect the permeability, formation factor, constriction factor or tortuosity calculations as those parameters are only dependent upon the conducting porosity. Pores not connected throughout the system do not conduct flow, and the pressure drop should therefore be zero. However, because of numerical inaccuracies, the calculation of pressures in dead-end and non-spanning pores can yield different results. Even with a slight difference in pressure, several problems can arise. The potentially most critical problem is related to tracking of streamlines, which can cause the script to crash. The tracking of streamlines can also be computationally demanding with a high amount of dead-end and non-spanning pores. All scripts can be used with both original and updated networks except `netStream.py`, where updated network data files are required due to the problems described above.

Because of the problems mentioned, a script was written to remove all non-spanning pores and throats. To maintain the structure of the original data-files, the program also rearranges the network such that the index of all the pores and throats are consistent with the total number of pores and throats. The removal and rearrangement process is illustrated in Figure 4.7.

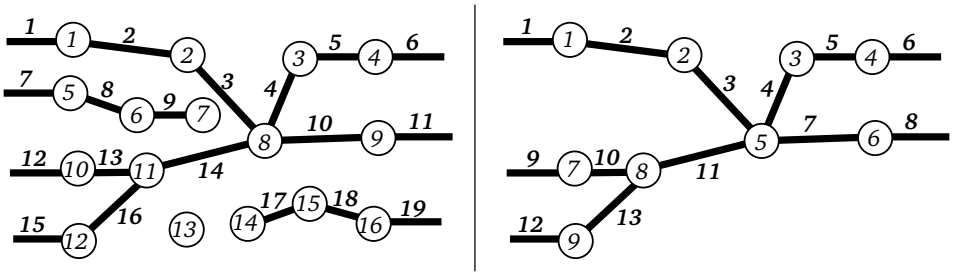

Figure 4.7: The original network (left) contains both a zero-coordination (node 13) pore and several non-spanning pores (i.e. nodes 5 or 15) and links (i.e. links 7 or 19). These are removed in the updated network (right) after the reconstruction and rearrangement process. The indexes of the pores and throats are updated (i.e. node $10 \rightarrow 7$ or throat $16 \rightarrow 13$) to maintain the consistency of the network data files.

The removal and rearrangement process can be computationally demanding, primarily the rearrangement (the re-indexing of pores and throats). The run time increases with increasing number of nodes and links in the network model. However, the process only needs to be preformed once for each network model. The run time for both pressure calculation and streamline tracking is reduced because of fewer pores and throats.

To validate the modification and rearrangement process some of the network properties calculated from the original and updated networks are compared. The permeability, formation factor, average coordination number and global aspect ratio is used in this analysis, and the results are presented in Chapter 5.

4.5.1 Reconstruction and Rearrangement Method

The script to locate the dead-end and zero coordination number pores utilize the information about connected pores in the `node1.dat` network file. The inlet and outlet nodes are first identified. Starting from a given set of nodes, the algorithm moves through the network in a virus-like manner, marking all nodes connected directly or indirectly to the start nodes. The function, `map_connections` in `netRecon.py`, is applied to both the inlet and outlet of the network model. Figure 4.8 illustrates how the algorithm works. The nodes which are not marked from both the inlet and outlet side are removed, together with the associated links, from the temporary data files. In this process both non-spanning and zero coordination number pores and throats are eliminated from the network. However, the temporary data files do not have the original structure because of the mismatch between indexes and the new total number of nodes and links.

This problem is solved by applying another algorithm to the temporary network data-files. Whenever a leap in indexes is encountered in the temporary data files `node1.dat` or `link1.dat`, the associated node or link is given a new index. The update of index is also applied to rest of the data files, maintaining the correct connections between the nodes. As the structure of `node2.dat` and `link2.dat` are similar to the other data files they are updated simultaneously.

4.5.2 Read and Write Network Data Files

The data files describing the pores and throats of the network have specific formats. The function, `get_network_file`, which read the different network data files was included in `netRecon.py`. Depending on the input data file, the function skips the header lines, and stores the data in a nested list, where each inner list corresponds to a certain pore or throat.

When a network is updated the data files are only temporary stored within the program. In order to use the updated data files, the function, `print_network_to_file` (included in `netRecon.py`), was written to write and export the temporary data files to the standard

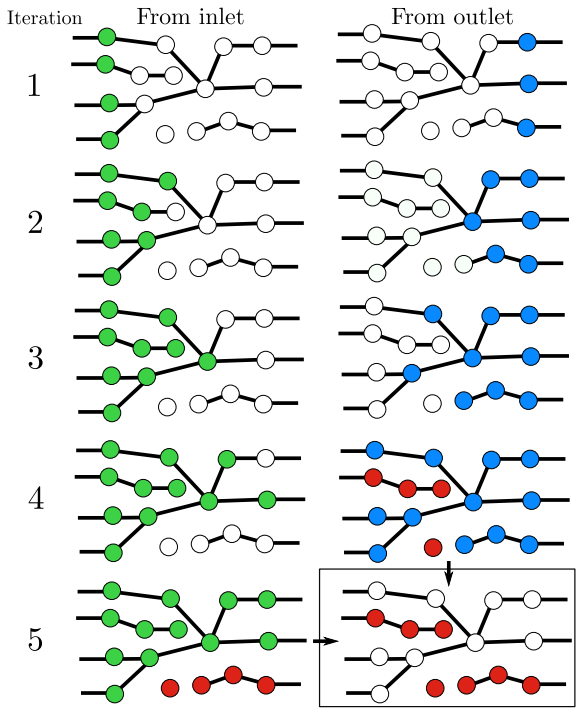

Figure 4.8: The implemented algorithm spreads through the network in a virus-like manner. For each iteration the ”infected” nodes are illustratively marked in green or blue. When the algorithm is applied from both sides of the network, the combined results can be used to identify the nodes and links, marked in red, which are not connected through the network. The nodes and links marked from both directions (green and blue) constitute the updated network.

format. Hence, the network updating process only needs to be conducted once. The original data files are moved to a separate folder.

4.5.3 Updating the Network

All the network manipulation functions are written in `netRecon.py`, and can be accessed from any script. The script `updateNetwork.py` applies several of the functions mentioned above to update the network. This produces updated network data files, which do not contain any non-spanning pores or links, and moves the original data files to a predefined location. This script also outputs some information about the non-spanning pores and throats to the screen.

4.6 The Percolation Process

4.6.1 Radius at the Bond Percolation Threshold

As discussed in Section 3.4.2, by sorting the links by increasing radius, and removing them in that order, the radius of the link which disconnects the inlet from the outlet can be found. Since a network can consist of thousands of throats, removing them one by one is computationally demanding. The bisection method is therefore used to increase the efficiency. Clearly, if all throats are removed, there will not exist any sample-spanning cluster. Likewise, if no throats are removed, there will likely be multiple pathways from the inlet to the outlet. Therefore, initially the upper limit for p_{cb} is set to 1 and the lower limit is set to 0. Then, half of the throats are removed and the function `map_connections` is used to check if there are any paths connecting the inlet and outlet. Depending on the outcome, the lower or upper limit is moved and a new interval for p_{cb} is established. Half of the throats in the new interval is then removed and the connection through the system is checked. This is repeated until the bond percolation threshold, and the associated radius of the largest removed throat, is found. The bond percolation threshold and B_c was found for all the networks studied in this thesis, and the results are presented in Table A.1 in Appendix A.

4.6.2 Threshold Capillary Pressure for Snap-Off

Snap-off can occur by either two menisci meeting, or by the meniscus in the sharpest corner meeting the pinned meniscus. Since it is difficult to know how the menisci will move in the pore throats, it is assumed that two menisci are free to move. In this case the threshold capillary pressure at which snap-off occurs is given by (Valvatne, 2004)

$$P_c^s = \frac{\gamma}{r}\left(\cos(\theta_A) - \frac{2\sin(\theta_A)}{\cot(\beta_1)+\cot(\beta_2)}\right) \tag{4.25}$$

The percolation radius is used to calculate P_c^s, and the interfacial tension and the advancing contact angle are equal to those used in the waterflooding simulations in e-Core. The corner half-angles used in Eq. 4.25 are found based on the average shape factor of the links. The angle β_1 is calculated by Eq. 3.17, and β_2 is the average of Eqs. 3.18 and 3.19.

The threshold capillary pressure for snap-off found by the percolation process is compared to the capillary pressure recorded at $k_{ro} = 0$ during waterflooding in e-Core. At the bond

percolation threshold, the oil-phase is not connected through the network, and by definition the oil relative permeability is zero. The calculated capillary pressure should therefore correspond to the capillary pressure at which the oil relative permeability becomes zero during water flooding. This correspondence is studied for all the networks used in this thesis, and the results are presented and discussed in Chapter 5.

Two different waterflooding simulations, with different advancing contact angles, are conducted for each network model. The associated residual oil saturations are presented in Table A.2 in Appendix A. In the first simulation a span of advancing contact angles from 20° to 60° are used, while in the second simulation a fixed advancing contact angle of 30° is used. Real porous media have a range of advancing contact angles, which makes the results from the first simulation more realistic. However, since Eq. 4.25 is dependent on the advancing contact angle the results from the second simulation are compared to the calculated capillary pressures. In both simulations every pore is modeled as water-wet.

4.7 Cluster-Size Distribution

After imbibition oil is trapped in clusters of varying sizes. To visualize the cluster-size distribution, we follow a similar method to the one presented by Chatzis et al. (1983). Chatzis et al. (1983) defined cluster-size as the number of nodes constituting each cluster. In most networks the number of clusters consisting of one node (singlets) is large. In order to obtain a more continuous distribution, the volume of each cluster is found. The cluster size (volume) is defined as X_i, where i is the number of clusters of volume X_i. The number of clusters of volume X_i is defined as N_i. The total volume, V_{tot}, of the clusters is given by,

$$V_{tot} = \sum X_i N_i \tag{4.26}$$

The volume fraction of oil trapped in a clusters of volume X_i was calculated by,

$$f(X_i) = \frac{X_i N_i}{V_{tot}} \tag{4.27}$$

The cumulative fraction of clusters of size X_i smaller than X_j is given by,

$$P(X_i \leq X_j) = \sum_{i=1}^{j} f(X_i) \tag{4.28}$$

The cumulative cluster-size distribution, $P(X_i \leq X_j)$ is plotted as a function of the clusters relative size, $f(X_i)$. Some of the results are presented in Chapter 5. The plots for the remaining models are included in Section A.7 in Appendix A. Chatzis et al. (1983) found that the the cluster-size distribution was dependent on the average coordination. The number of singlets increased with increasing coordination number. He also observed an effect of snap-off on the cluster-size, where higher aspect ratio networks yielded larger relative fractions of smaller clusters.

The script `clusterTrack.py` was written to identify the different clusters. The script is based on the virus-spreading algorithm presented in Section 4.5.1. The total volume of each cluster is found by summarizing the volume of each node in the cluster. The throat volumes are minor in comparison to the pore body volumes and are therefore neglected.

4.8 e-Core

The electronic rock core laboratory e-Core is used to extract pore networks from micro-CT volumes and to conduct various two-phase flow simulations. e-Core is a project-based software where multiple rocks can be added to each project. All the models used in this thesis were imported into e-Core.

Before a pore network can be extracted, a micro-CT volume must be imported using *Import Micro CT Volume*. The grains are then identified using the option *Grain Recognition*. When the grains are identified, a pore network can be extracted by selecting *New Pore Network*, and different flow simulations, such as primary drainage and waterflooding, can be conducted.

e-Core was also used to digitally generate pore network models of varying sizes. These models were used in the development phase to verify the simulations conducted with the different scripts.

4.9 Network Models

A selection of different network models are used for the simulations in this thesis. The majority of the models are sandstones, but some carbonates and sand packs are also studied. All the pore networks are extracted from published micro-CT volumes using e-Core. Due to the non-uniqueness of pore network models, the results from the simulations in this

thesis can differ from published results. A brief overview of the models, and some of their key properties are presented below.

4.9.1 Original and Updated Networks

As described in this chapter, the network files are manipulated to remove nodes with zero coordination number and non-spanning nodes and links. The updated networks yield different results for several pore structure measures. The two different networks (both describing the same network models) are referred to as *original* and *updated* throughout Chapters 5 and 6.

4.9.2 Network Models from Imperial College London

Imperial College London (ICL) have published several micro-CT volumes of different rock samples (Imperial College London, 2018). Even though the network data files for most of these models are available, all networks were extracted in e-Core to get a better basis for comparison. The networks vary in size, where the smallest have approximately 560 nodes and the largest have 22300 nodes. Most of the models are sandstones, but in order to obtain more generalized results some carbonates and sand packs are also evaluated. The network models and some of their inherent properties are presented in Table 4.1. The acronyms given in Table 4.1 will be used to reference the specific models throughout this thesis.

4.9.3 Fontainebleau Network Models

The other set of network models used for the simulations in this thesis is a series of Fontainebleau sandstones published by Berg (2016). The networks, and some of their properites are presented in Table 4.2. The Fontainebleau sandstone is regarded as a non-complex rock with pure mineral composition, making it a popular choice when studying the variation of petrophysical properties independently from other parameters (Al Saadi et al., 2017). These models are digitally reconstructed in e-Core using the exact same grain distribution, and identical numerical modeling of compaction and diagenese. Thus the models are fairly similar, with porosities ranging from 8% to 25% by varying the degree of quartz cementation. The similarity between the models is of great benefit when analyzing the changes between simulations.

Table 4.1: Properties of network models from Imperial College London. The number of nodes and links are from the original networks.

Network	Number of		Permeability [md]	Formation factor	Total porosity	Effective porosity
	Nodes	Links				
			Sandstones			
S1	1825	3223	2201.9	40.0	14.1 %	12.8 %
S2	2959	6292	15517.3	11.3	24.6 %	23.7 %
S3	9867	16814	489.1	57.4	16.9 %	14.6 %
S4	10426	16389	427.3	75.8	17.1 %	13.3 %
S5	557	1141	18609.4	20.6	21.1 %	19.3 %
S6	776	2099	22013.3	17.4	24.0 %	22.0 %
S7	1547	3636	25856.2	10.4	25.1 %	24.3 %
S8	2485	6483	41597.2	6.5	34.0 %	32.9 %
S9	699	1372	15413.0	21.4	22.2 %	20.0 %
Bentheimer	19587	43723	2469.6	15.9	21.7 %	20.8 %
Berea	8257	16653	4213.6	23.8	19.7 %	18.5 %
Doddington	6344	12838	2332.5	24.7	19.6 %	18.1 %
A1	6814	19976	45451.1	4.0	42.9 %	41.9 %
			Sand Packs			
F42A	6620	16644	52227.7	5.6	32.2 %	31.3 %
F42B	6635	17086	50498.6	5.4	33.3 %	32.4 %
F42C	7084	17834	48591.5	5.4	32.9 %	32.0 %
LV60A	18957	51483	33064.4	4.5	36.3 %	35.7 %
LV60C	21339	59007	31403.6	4.3	36.8 %	36.3 %
			Carbonates			
Estaillades	22300	48389	117.8	212.2	12.7 %	7.2 %
Ketton	7171	14749	3657.9	28.3	13.3 %	12.3 %
C1	5452	13839	6037.4	32.8	23.3 %	18.9 %
C2	7402	14462	398.4	182.9	16.8 %	10.7 %

Table 4.2: Properties of Fontainebleau network models.

Network	Number of		Permeability [md]	Formation factor	Total porosity	Effective porosity
	Nodes	Links				
F8	6548	9892	100.8	297.5	8.6 %	6.0 %
F10	7628	12437	352.8	130.4	10.2 %	8.4 %
F13	9150	16845	1129.6	56.9	12.6 %	11.6 %
F15	10456	21017	3686.2	26.6	15.4 %	14.6 %
F18	11198	23862	6358.0	18.0	17.6 %	17.1 %
F21	11947	27461	11831.2	12.1	20.7 %	20.2 %
F25	11918	29581	21113.8	8.4	24.5 %	24.1 %

Chapter 5

Results

5.1 Validation of the Network Manipulation Process

Certain pore structure properties of selected networks were compared in order to validate the network manipulation process. The numbers of nodes and links removed from the networks studied are shown in Table A.3. In Tables 5.1 and 5.2 the coordination number and the global aspect ratio are compared.

Table 5.1: Difference between network properties of selected original (orig.) and updated (upd.) ICL networks.

Model	Number of				Z		Global aspect ratio	
	Nodes		Links					
	Orig.	Upd.	Orig.	Upd.	Orig.	Upd.	Orig.	Upd.
Bentheimer	19587	18954	43723	43423	4.41	4.53	1.65	1.67
Berea	8257	6017	16653	12653	3.99	4.15	1.57	1.59
Doddington	6344	7969	12838	16505	3.96	4.07	1.62	1.64
Estaillades	22300	15172	48389	40380	4.31	5.30	1.44	1.50

The permeability and formation factor (not shown here) were exactly the same for simulations conducted on the original and updated networks. These calculations are only dependant upon the conducting porosity and should not be altered by removing non-conducting pores and throats. This is therefore the best verification that the network manipulation process was conducted correctly.

Table 5.2: Difference between network properties of original (orig.) and updated (upd.) Fontainebleau networks.

Model	Number of				Z		Global aspect ratio	
	Nodes		Links					
	Orig.	Upd.	Orig.	Upd.	Orig.	Upd.	Orig.	Upd.
F8	6548	5678	9892	9187	2.96	3.19	1.75	1.79
F10	7628	6898	12437	11816	3.20	3.38	1.75	1.78
F13	9150	8848	16845	16649	3.62	3.71	1.72	1.74
F15	10456	10275	21017	20922	3.96	4.02	1.69	1.70
F18	11198	11019	23862	23761	4.20	4.25	1.67	1.68
F21	11947	11847	27461	27404	4.54	4.57	1.64	1.65
F25	11918	11848	29581	29547	4.90	4.92	1.62	1.62

The average coordination number changes as expected. The average coordination number is lower for original networks. This is likely due to the removal of all the zero coordination number pores. The numbers of nodes and links removed should decrease as the networks become more well-connected - i.e. higher average coordination number. This is evident when considering the Fontainebleau-series, where the least amount of nodes are removed from the F25-network, which also has the highest average coordination number. In the manipulation of the Estaillades-network the number of links removed exceeds the number of nodes removed. This could be caused by the number of redundant links (links connecting the same two nodes) being higher for this network, which is also the case for the majority of the carbonate-networks.

5.2 Difference Between Streamline Tracking Methods

Even though, as mentioned in Section 4.3.2, the location-based tracking mechanism may yield less realistic flow pattern through the system, it yields very similar results to the angle-based mechanism. For both implementations, the cumulative flowrate of all the streamtubes are very close to the total flow rate (deviation less than 1%) calculated by the method proposed in Chapter 4. Constriction factor, tortuosity, number of streamlines and fraction of unused nodes (nodes which do not conduct flow) are compared in the cross-plots in Figure 5.1. The coefficient of determination (R^2) is very close to one for all the cross-plotted parameters, which indicates that the two different tracking mechanisms yield streamlines with similar paths through the system.

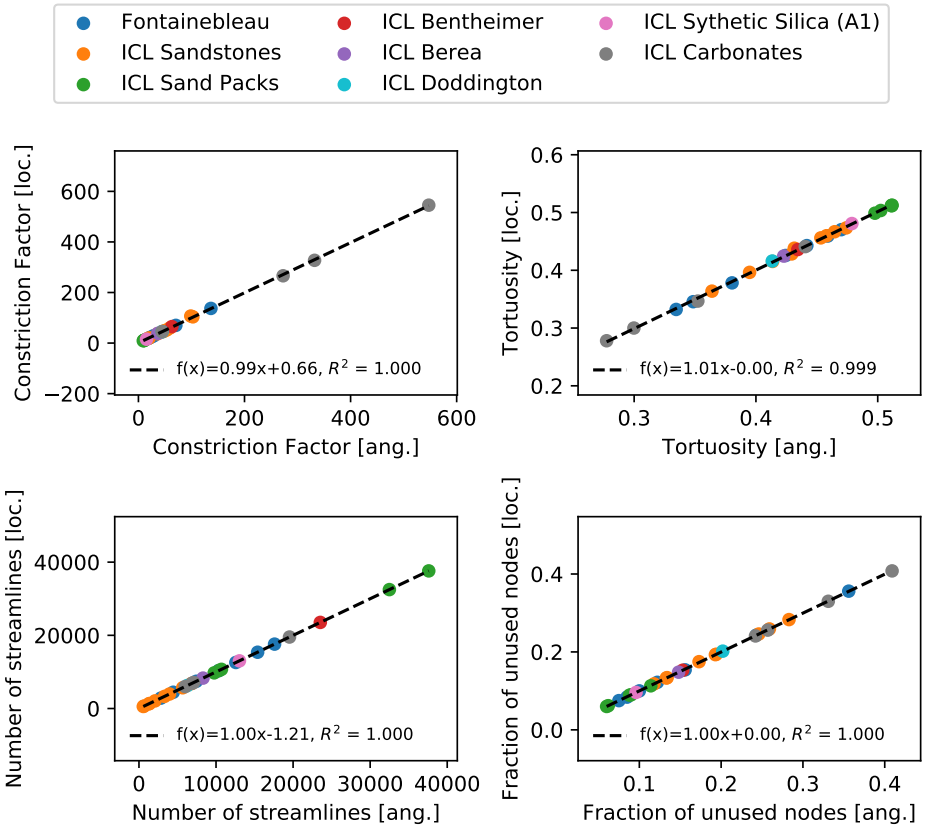

Figure 5.1: Cross-plots of different properties obtained by streamline tracking based on the angles between inlet and outlet links and the location of the outlet links. The correspondence between the the two different tracking mechanisms is very good, yielding R^2-values very close to 1. The sorting mechanism is given by either [loc.] (location) or [ang.] (angles).

The close correspondence between the two tracking mechanisms were not expected, as they are based on two independent criteria. However, at average, the number of links conducting flow from each pore (outlet links) are fairly low. The average coordination number is about 4, which includes both inlet and outlet links. Therefore, when a streamline is tracked to a certain pore, a limited number of outlet links can be chosen. The average number of outlet connections per node for all the networks studied in this thesis is 2.36. The two tracking mechanisms are independent of each other, and given the limited number of outlet links to choose between, it is reasonable to believe that the same outlet links are often chosen at a certain pore.

By forcing the streamlines to increasing yz-location in the location-based tracking mechanism, they will follow relatively straight paths and traverse the system in a parallel manner. The angle-based tracking mechanism will not yield intersecting streamlines, and result in an approximately equal density of streamlines.

5.3 Pore Structure

Two different waterflooding simulations were conducted using e-Core, with advancing contact angles of $\theta_A = 20° - 60°$ and $\theta_A = 30°$ assigned to the water. The interfacial tension in both simulations was set to $0.03\ N/m$. The resulting residual oil saturation from both waterfloods are compared to a wide range of different pore structure measures in Figures 5.3 through 5.11.

The wettability influences both the flow and distribution of fluids in porous media. The residual oil saturation after waterflooding is therefore highly dependent on the wettability of a rock (Anderson, 1987). This dependency was confirmed by conducting three waterflooding simulations with varying wettability to the F8-model, as can be seen in Figure 5.2. The correlations and trends between different measures of pore structure and S_{or} found in this chapter are therefore not necessarily applicable to rocks of different wettability.

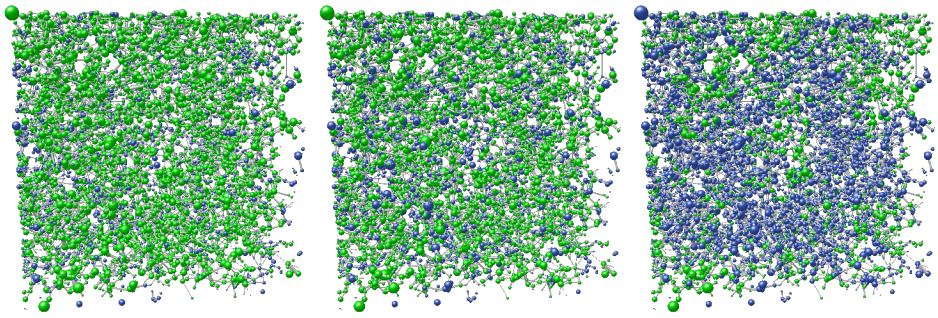

Figure 5.2: Ball and stick plots of the saturation state after waterflooding. The wettability is altered by varying the fraction of oil-wet pores. The figures, from left to right, have 0%, 50% and 100% oil-wet pores respectiviely, resulting in residual oil saturations of 70.2%, 69.6% and 31.7%. The green nodes have residual oil saturation, while the blue nodes are fully water saturated.

5.3.1 Coordination Number

The correlation between coordination number and residual oil saturation, calculated for both original and updated networks, can be seen in Figure 5.3. The correspondence is better for coordination numbers derived from original networks with $R^2 = 0.675$ for water-wetting conditions and 0.705 for the even more water-wetting scenario. The trend

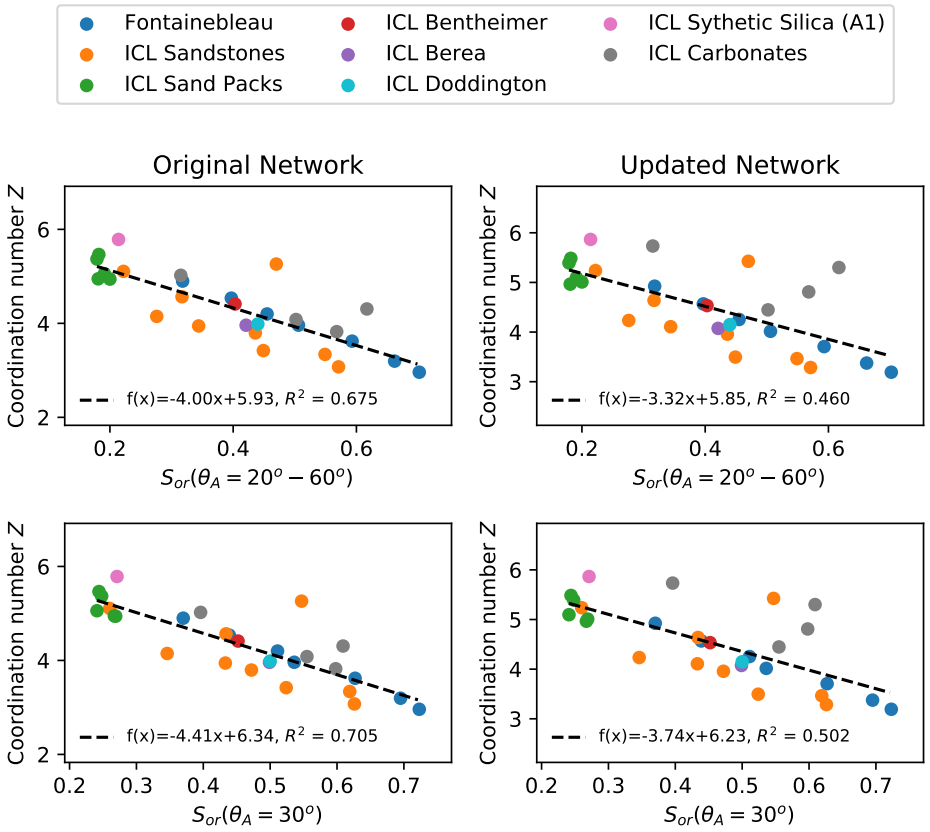

Figure 5.3: Coordination number Z as a function of S_{or}. A better correlation can be seen for simulations conducted on original networks (left), than for updated networks (right). In both cases, the correspondence is better for the simulations conducted with a fixed advancing contact angle of 30°.

between residual oil saturation and coordination number is evident. A higher average coordination number corresponds to a higher number of redundant loops and thus more escape paths for the oil during imbibition. In addition, when a pore has a high coordination number it is likely that the radius of at least one of the bounding throats is large, making other displacement mechanisms more favoured, and thus reduce the residual oil saturation.

A significant difference between R^2 for original and updated networks can be observed from Figure 5.3. This indicates that both the conducting and the non-conducting pore space is important when estimating residual oil saturation.

5.3.2 Constriction Factor

Figure 5.4 shows the constriction factor C (for both angle- and location-based tracking) as a function of the residual oil saturation S_{or} (for both $\theta_A = 20° - 60°$ and $\theta_A = 30°$) for all the networks. The correspondence between constriction factor and residual oil saturation is poor, indicated by low coefficient of determination in all four cross-plots. The outlying carbonates affects this correlation, but even if those are excluded the correspondence is quite poor ($R^2 = 0.56$).

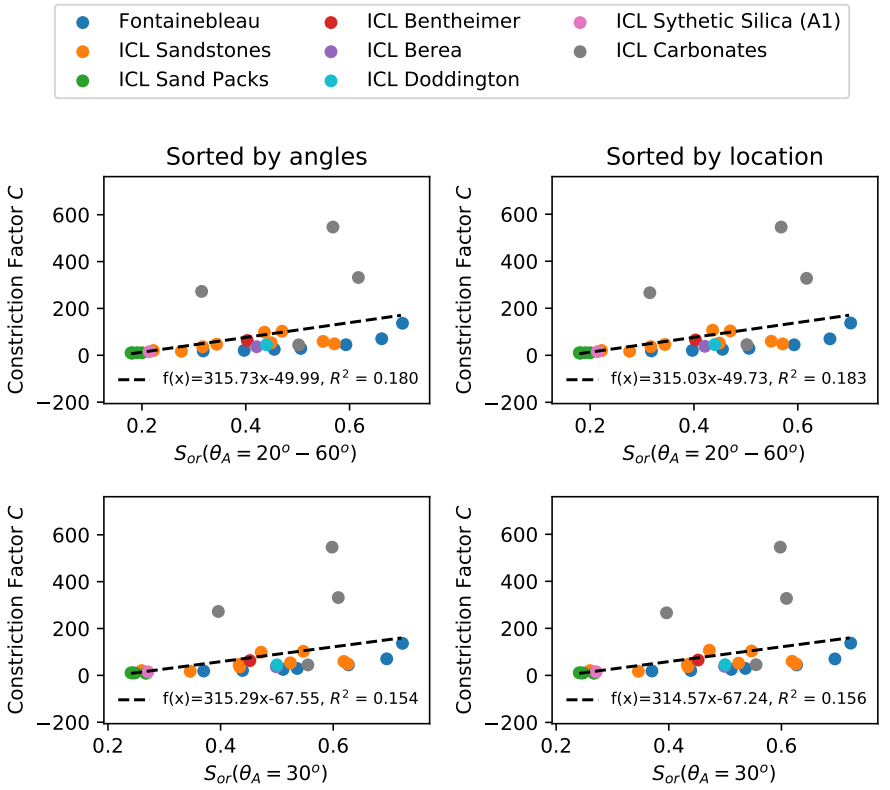

Figure 5.4: Constriction factor C as a function of S_{or}. The results are very similar, and the coefficients of determination are low, for both the angle-based (left) and the location-based (right) tracking mechanisms.

The constriction factor is calculated along the streamlines, thus only the conducting pores and throats are considered. The non-conducting pores and throats will also contribute to residual oil saturation, but those are not accounted for. This could be a reason for the poor correlation.

With significantly higher constriction factors the carbonates deviate from the trendline. Since the hydraulic constriction factor is proportional to A^2 it is very sensitive to large variations in cross-sectional area. Vugs are commonly encountered in carbonates and will induce such large variations. The high values of the constriction factor could be caused by the magnification of the variation of cross-sectional area due to the vugs. In an attempt to reduce this effect the electric constriction factor, which is proportional to A, was calculated. The correlation with the electric constriction factor was slightly better with $R^2 \simeq 0.3$, but the carbonates still yielded significantly higher constriction factors. The better correlation obtained with the electric constriction factor could indicate that the residual oil saturation is more linearly dependent on the cross-sectional area along the conducting pore space. However, it does not seem like the constriction factor, both hydraulic and electric, is suitable for predicting the residual oil saturation for the wide range of models studied in this thesis.

The script, `netStream.py`, failed to calculate the constriction factor for a few streamlines in certain networks due to numerical inaccuracies in the calculation of intra-link pressures. However, this does not have a significant impact on the total constriction factor of the system.

5.3.3 Global Aspect Ratio

Figure 5.5 shows the global aspect ratio as a function of the residual oil saturation S_{or} for both original and updated networks. The correlation is poor, both for original and updated networks and both waterflooding simulations. However, a better correspondence is exhibited for the updated network, where the non-spanning and zero-coordination number nodes are removed.

Snap-off is dependent on the aspect ratio, and a trend between residual oil saturation and aspect ratio is therefore expected. However, as the aspect ratio decreases, the trapping mechanism changes from snap-off to bypassing. Therefore the dominating trapping mechanism in models with low aspect ratios could be bypassing, and they could therefore deviate from the trend. All the models are water-wet, and the change from snap-off to

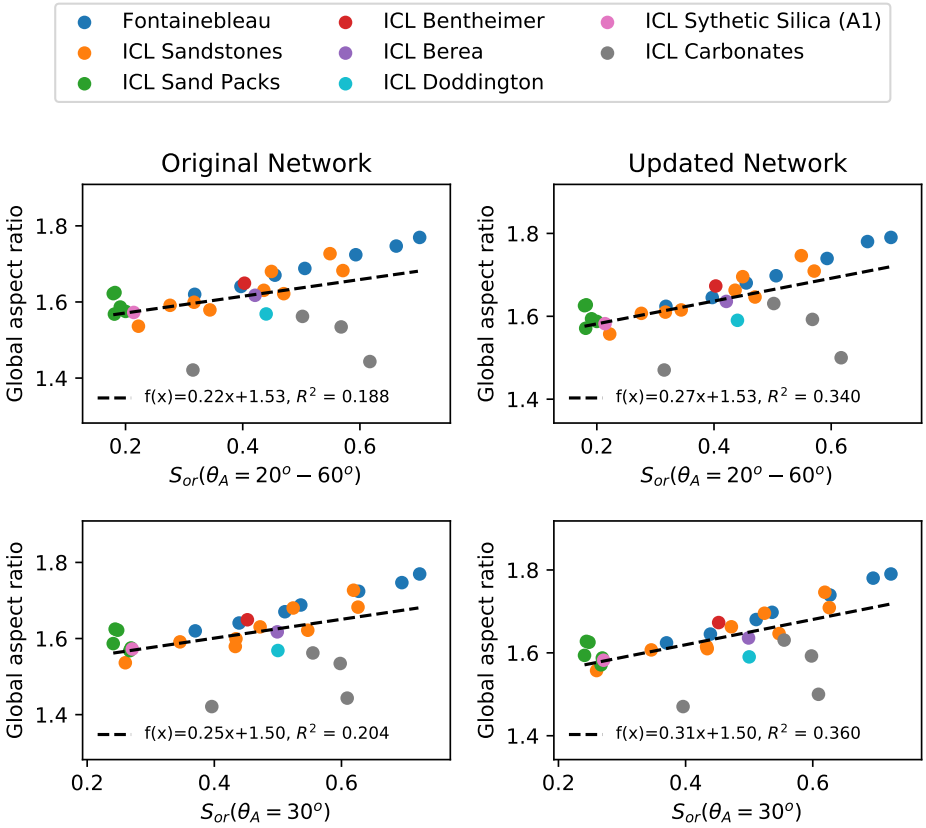

Figure 5.5: Global aspect ratio as a function of S_{or}. Global aspect ratios calculated from the updated networks (right) display better correspondence with the residual oil saturation than those derived from the original networks (left). The R^2-values are low in all cases, and no strong correlation can be seen for the range of networks studied.

bypassing is not observed. In the networks studied in this thesis it is only the carbonates that significantly deviate from the trend. Carbonates usually have large vugs, which would yield high aspect ratios, and snap-off is expected to be the dominating trapping mechanism. A reason for the deviation could be that there are few large vugs compared to smaller pores, and their effect on the global aspect ratio will therefore be minor. As an attempt to account for this, a volume weighted global aspect ratio was calculated. The correlation yielded $R^2 \simeq 0.21$, which is only marginally better. Nevertheless, the global aspect ratio does not seem to be suitable for predicting S_{or} for the wide range off models studied in this thesis.

When a network is updated all the zero-connection pores are removed, and generally more

pores are removed compared to throats. Due to this the average throat size has a larger impact on the global aspect ratio for updated networks, and shifts it to slightly higher values. This could be a reason for the better correlations obtained from the updated networks.

5.3.4 Local Aspect Ratios

Figure 5.6 shows the two different measures of local aspect ratio as a function of the residual oil saturation S_{or} for both waterflooding scenarios. The correlations for the average local aspect ratios are somewhat poor, indicated by low coefficients of determination.

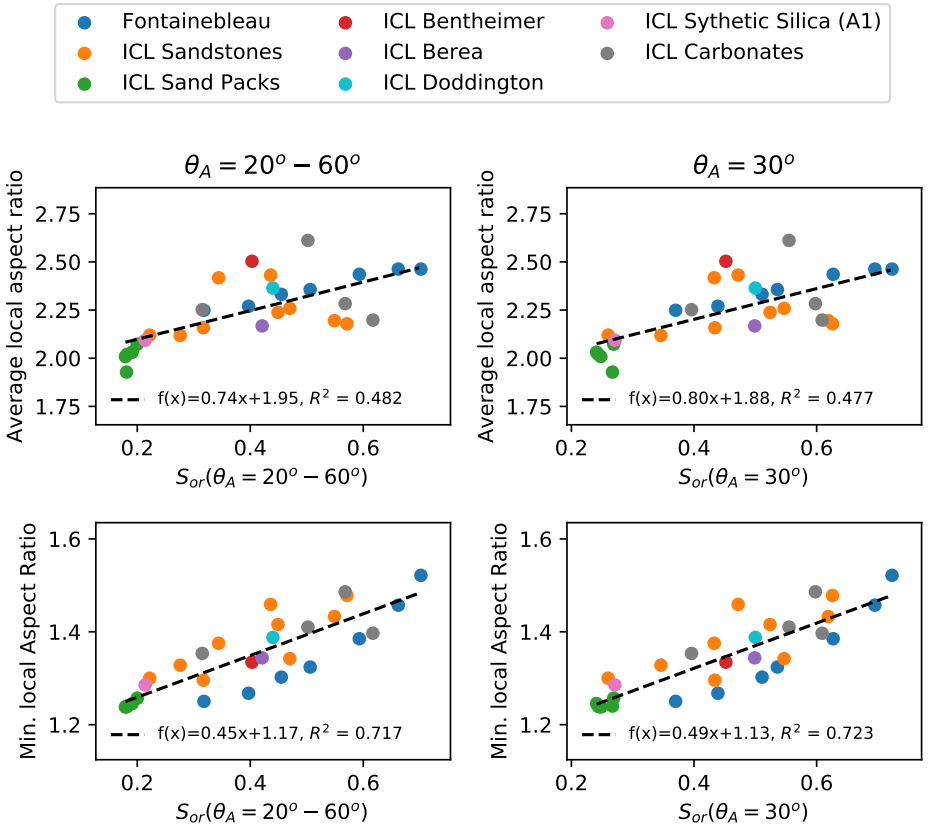

Figure 5.6: Two different local aspect ratios as a function of S_{or}. Both local aspect ratios are derived from the original networks, and exhibit similar correspondence with the residual oil saturation. The R^2-values are fairly low in all cases, and no strong correlation can be seen for the range of networks studied.

The minimum local aspect ratio yields a better correlation, and the carbonates also match better with the other models. Both the local aspect ratios yield better correlations with

residual oil saturation compared to the global aspect ratio.

Snap-off is directly dependent on the aspect ratio. Since a pore can be filled by another displacement mechanism from the largest throat, even though the other throats are snapped-off, it is the largest bounding throat (i.e. the lowest aspect ratio) to any pore which determines if oil is trapped by snap-off. This is probably the reason for why the minimum local aspect ratio correlate better than the average local aspect ratio.

In Figures 5.7 and 5.8 the minimum local aspect ratio is plotted as a function of the local residual oil saturation. The plots for some of the Fontainebleau networks, Figure 5.7, and the carbonates from ICL, Figure 5.8, exhibit no correlation and yields very low coefficients of determination. The plots for the other models are included in Section A.5 in Appendix A.

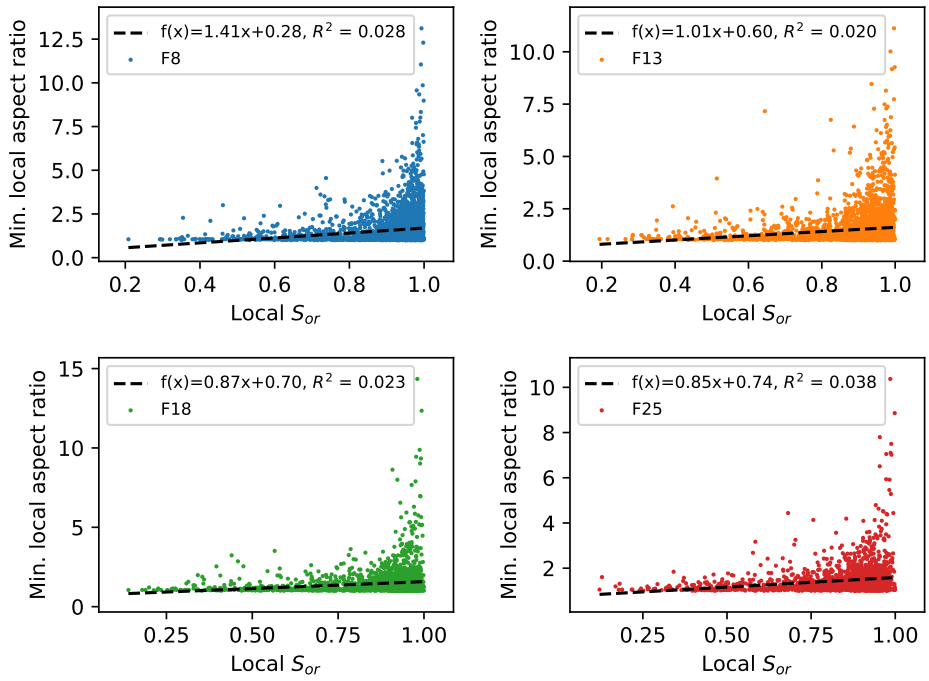

Figure 5.7: Minimum local aspect ratio as a function of local residual oil saturation for F8, F13, F18 and F25. The local S_{or} generally increase with increasing minimum local aspect ratio, but no evident correlation can be seen.

As a general trend, the local S_{or} increases as the local minimum aspect ratio increases.

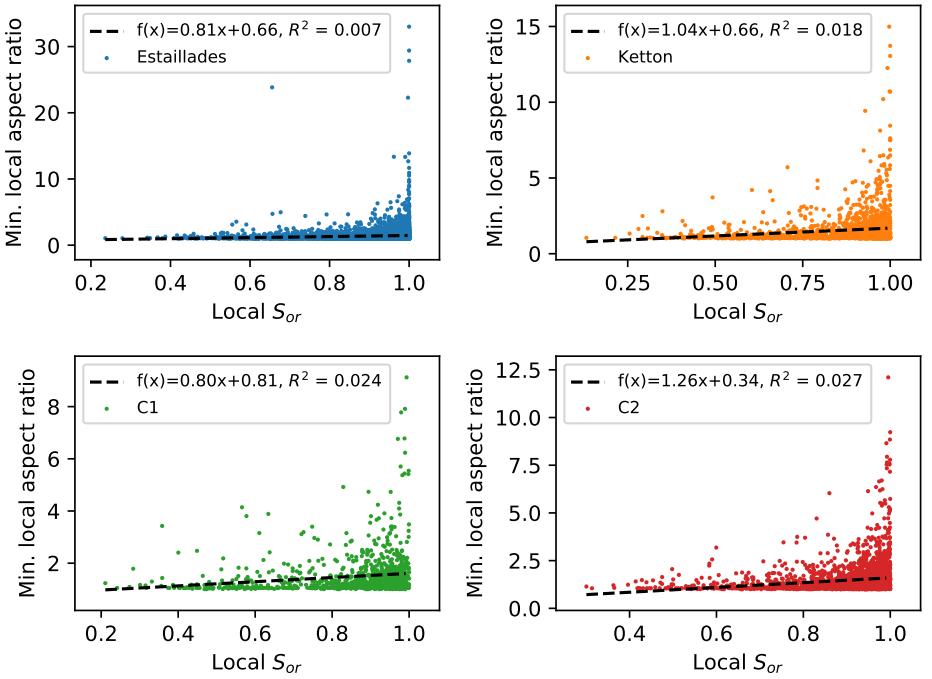

Figure 5.8: Minimum local aspect ratio as a function of local residual oil saturation for Estaillades, Ketton, C1 and C2. The local S_{or} generally increase with increasing minimum local aspect ratio, but no evident correlation can be seen.

The density of the data points are significantly higher for high local residual oil saturations, and the trend can therefore more clearly be seen from the binned scatter plots in Figures 5.9 and 5.10. In most of these plots (except Ketton) the coefficients of determination are fairly high. The number of data points in each bin varies greatly, and the plots can only be regarded as an indication of the observed trend.

Nie et al. (2016) alluded to a correlation between residual oil saturation and aspect ratio. From the results in this thesis this trend is evident, especially for the minimum local aspect ratio. The same trend is also observed for the global aspect ratio and the average local aspect ratio, even though the correlations are somewhat poor.

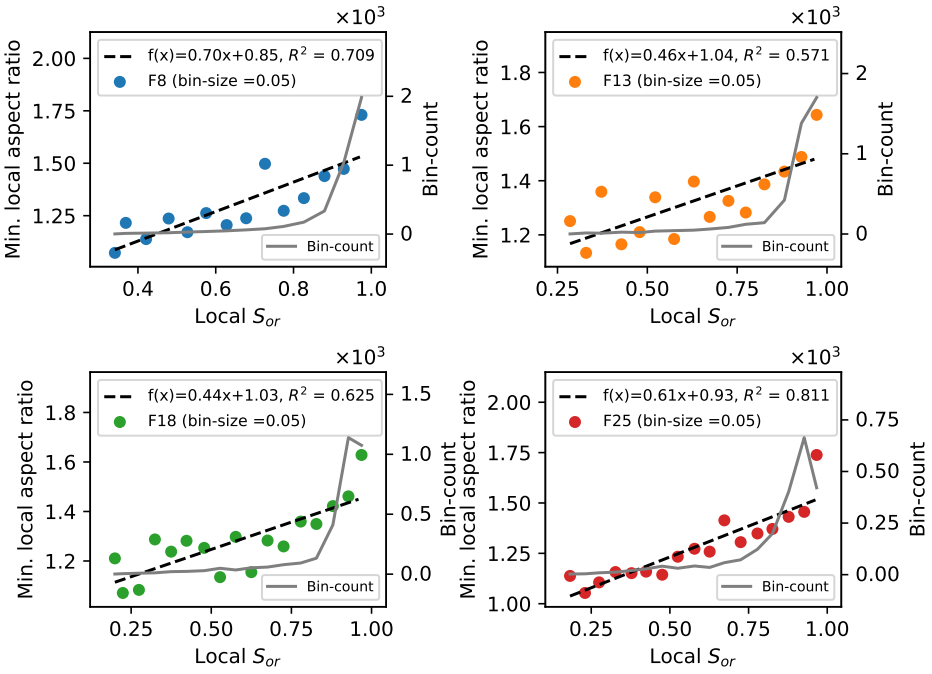

Figure 5.9: Binned plot of minimum local aspect ratio vs. local residual oil saturation for F8, F13, F18 and F25. The bin size is 0.05. The number of data points in each bin (bin-count) is also plotted.

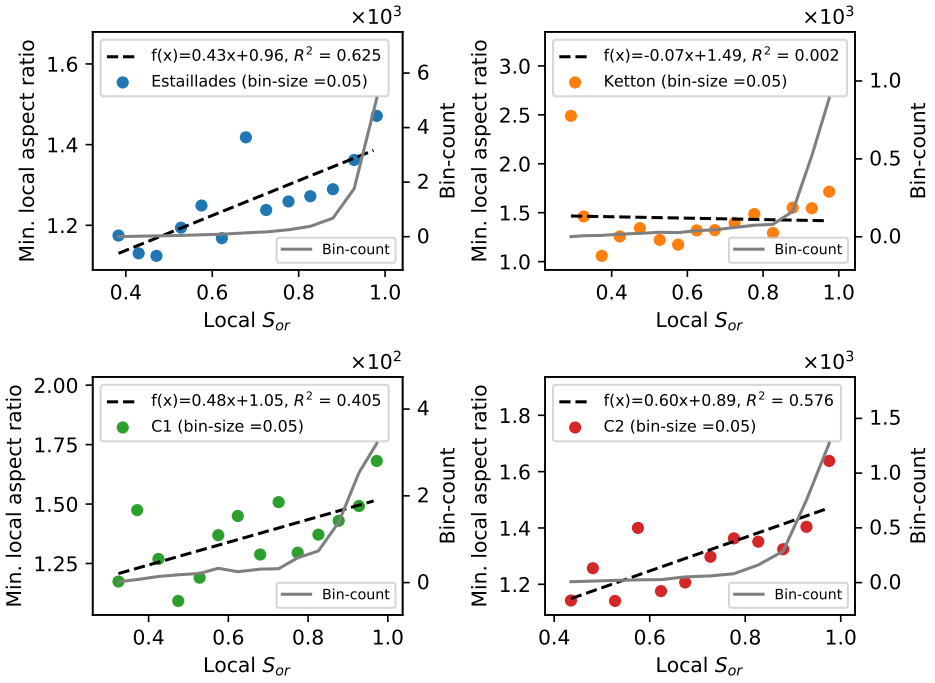

Figure 5.10: Binned plot of minimum local aspect ratio vs. local residual oil saturation for Estaillades, Ketton, C1 and C2. The bin size is 0.05. The number of data points in each bin (bin-count) is also plotted.

5.3.5 Total and Effective Porosity

In Figure 5.11 both total and effective porosity is plotted as a function of residual oil saturation. The correlation is quite good for both porosities, with effective porosity yielding slightly better correspondence.

Yuan (1981) stated that an increase in porosity correlates fairly well with an increase in average coordination number. This was also found for the networks studied in this thesis, and the correlation yielded $R^2 = 0.75$. It is also possible that the porosity is linked to the aspect ratio. It is reasonable to believe that the pore space is cemented fairly equally throughout, which would result in a decrease in porosity associated with an increase in aspect ratio. The increased residual oil saturation associated with the decrease in porosity could therefore be due to the associated decrease in average coordination number and increase in aspect ratio, both of which are conditions favorable to snap-off. As discussed previously, snap-off is less likely to happen in a well-connected pore space, since the oil

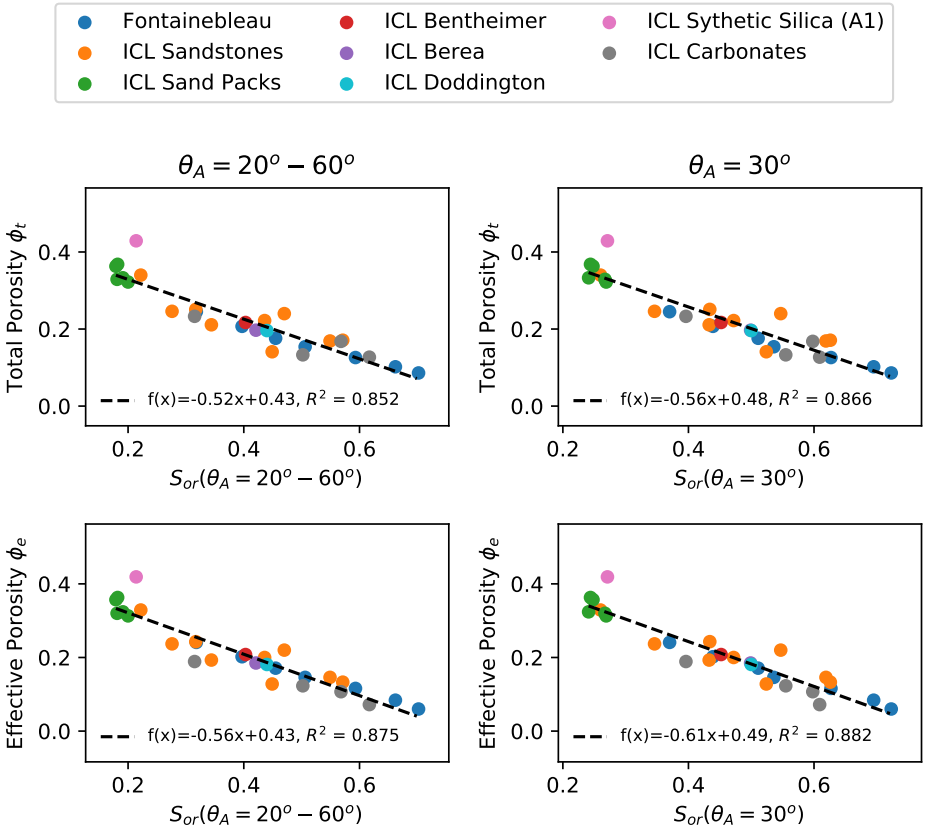

Figure 5.11: Total (ϕ_t) and effective (ϕ_e) porosity as functions of residual oil saturation for both waterflooding scenarios.

has more opportunity to escape, and a higher aspect ratio will also induce more snap-off.

5.4 Percolation Theory and S_{or} End-Effect

When the oil-phase lose connectivity through the network at the bond percolation threshold the oil relative permeability becomes zero. The calculated threshold capillary pressure for snap-off, using the percolation radius, should therefore correspond to the recorded capillary pressure at which the oil relative permeability becomes zero during water flooding. In Figure 5.12 the calculated capillary pressure is compared to the recorded capillary pressure at $k_{ro} = 0$. The correlation is good, indicated by a coefficient of determination of 0.929.

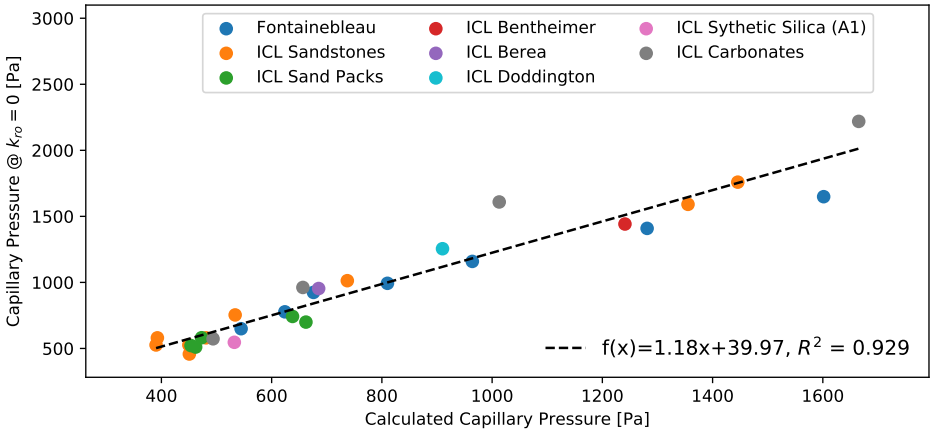

Figure 5.12: Calculated capillary pressure at the bond percolation threshold as function of the recorded capillary pressure at $k_{ro} = 0$.

This correspondence indicates that percolation threshold is the dominating factor for when the oil-phase looses connectivity through the system, at which $k_{ro} = 0$. In the waterflooding simulations conducted using e-Core it was observed that the imbibition continued after the oil-phase lost connectivity. The oil in the pores and throats which are connected to the outlet can still be produced. In pore network modelling, where small samples are considered, this can have a considerable impact on the end-point saturation. Of the total oil produced, 4.9% and 24.4% was on average produced after $k_{ro} = 0$ for the Fontainebleau and Carbonate networks respectively. This end-effect is probably of minor significance in a producing field due to the much larger scale. However, core flooding experiments are more similar to pore network models in scale, and could be influenced by this S_{or} end-effect.

All the simulations are conducted under water-wetting conditions and snap-off is expected to be the dominating trapping mechanism. A bond percolation process is similar to random snap-off in the throats, and the threshold capillary pressure for snap-off was calculated by using the percolation radius. The correspondence obtained in Figure 5.12 is therefore indicative to snap-off being the dominating trapping mechanism. It seems like a percolation process is suitable for modelling entrapment of the residual phase, and the findings are therefore in agreement with the suggestions of Melrose and Brandner (1974) and Larson et al. (1981).

The trend between the percolation threshold and the average coordination number is ev-

ident, with $R^2 = 0.64$ for the networks studied in this thesis. With a high number of redundant paths through the system more bonds can be closed without loosing connectivity over the system. A decrease in the bond percolation threshold seems to be correlated with a decrease in residual oil saturation.

5.5 Cluster-Size Distribution

In Figures 5.13 and 5.14 the cumulative cluster-size distribution $P(X_i \leq X_j)$ is plotted as a function of the relative cluster-size $F(X_i)$, as defined in Section 4.7 in Chapter 4.

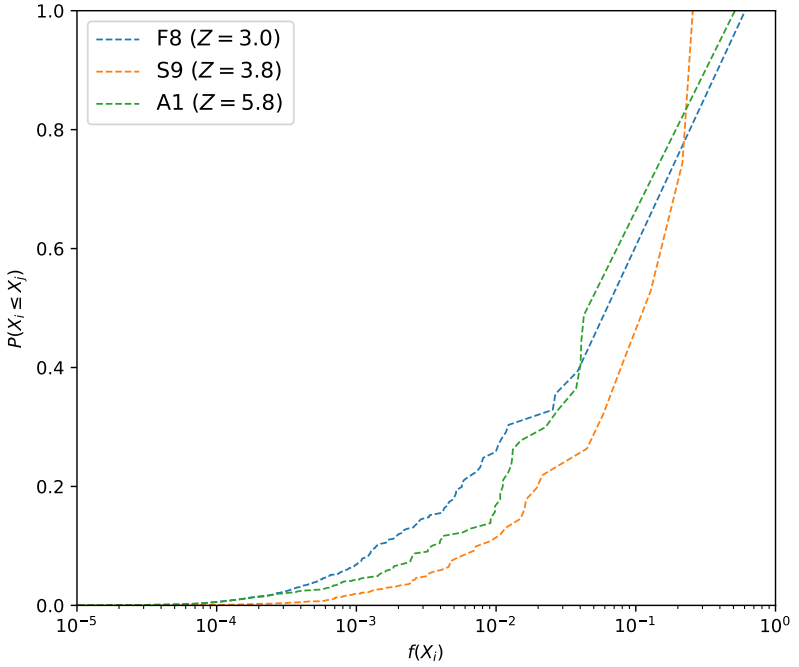

Figure 5.13: Cumulative cluster-size distribution $P(X_i \leq X_j)$ vs. relative cluster-size $F(X_i)$ for selected networks with varying average coordination number.

Networks with varying coordination numbers are plotted in Figure 5.13 in order to see the effect of the coordination number on the cluster-size distribution. In contradiction to the findings of Chatzis et al. (1983), a correlation between coordination number and cluster-size is not evident. Of the three networks, A1 has the highest coordination number and a significantly larger maximum cluster compared to S9. Since the coordination number is a measure of how well-connected the pore space is, one would expect that the lowest

coordination number would yield the largest maximum cluster-size. This is not the case, as can be seen in Figure 5.13.

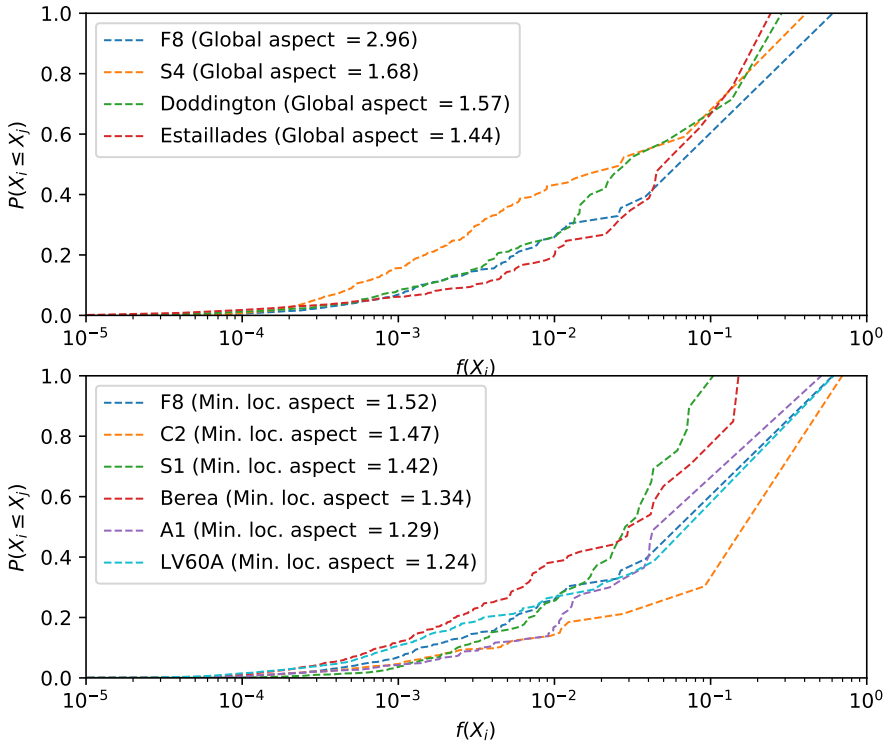

Figure 5.14: Cumulative cluster-size distribution $P(X_i \leq X_j)$ vs. relative cluster-size $F(X_i)$ for selected networks with varying global aspect ratio (top) and varying minimum local aspect ratio (bottom).

In the two plots in Figure 5.14 networks with varying global aspect ratio and minimum local aspect ratio are plotted. A trend between cluster size and local minimum aspect ratio is not evident. The F8 and LV60A networks, in the lower plot in Figure 5.14, have significantly different minimum local aspect ratios and yield relatively similar cluster-size distributions. In the upper plot in Figure 5.14 is seems like there could be a trend between the global aspect ratio and the maximum cluster-size. The relative maximum cluster size increase as the global aspect ratio increases. However, from the findings in this thesis it is not possible to see a clear correlation between cluster-size and either coordination number or aspect ratio.

Chapter 6

Conclusions

The aim of this thesis was to develop suite of scripts to perform simulations and calculate different properties on network models. A single phase pore network solver and streamline tracker were successfully implemented using Python. In this conclusion the most important results are presented and recommendations for further work are given. The residual oil saturation is sensitive to the wettability of the rocks, and the results are therefore not necessarily applicable to rocks with different wettability.

Network manipulation and rearrangement algorithms were successfully implemented. The flow field was not affected by removing all the non-spanning pores and throats. This was confirmed by comparing permeability and formation factor calculations conducted on both networks which yielded equal results. The calculation of pressure in non-spanning pores yielded slight numerical inaccuracies. This enabled the streamlines to traverse through these pores and caused the script, `netStream.py`, to crash. The updated networks eliminated these problems related to numerical inaccuracies when tracking streamlines.

The difference between tracking streamlines based on angles and location was minor. The constriction factor and tortuosity results indicate that the two implementations yield relatively similar paths through the system. This could be due to the limited number of outlet links from each pore. The similar fraction of unused nodes and the total number of streamlines indicate that both methods yield a similar density of streamlines. However, by completely avoiding intersecting streamlines, the angle-based tracking method is assumed to yield more realistic flow patterns through the pore space.

From the results obtained in this thesis it is evident that the pore structure has a measurable impact on the residual oil saturation. However, the trapping mechanisms are complex, and as the pore structure measures are dependent on each other, it is difficult to find good correlations based on one sole descriptor. The effective porosity yielded the best correlation with residual oil saturation. This could be due to the relationship between porosity, aspect ratio and coordination number. The constriction factor, the global aspect ratio and average local aspect ratio does not correlate with the residual oil saturation. This is party caused by the carbonates which deviates from the results obtained from other networks.

Some of the pore structure measures are derived from both original and updated networks. The coordination number yields better correlations for the original networks, while the global aspect ratio yields better results for the updated networks. The reason for this is not determined, however it indicates that both the conducting and the non-conducting pore space are important when estimating the residual oil saturation.

It was found that the bond percolation threshold is the dominating factor for when the oil-phase looses connectivity through the system. By using the percolation radius when calculating the threshold capillary pressure for snap-off, a good correspondence with the recorded capillary pressure at $k_{ro} = 0$ was obtained. This indicates that snap-off is the dominating trapping mechanism in the networks studied in this thesis. The cluster-size distribution of oil after waterflooding was also investigated. However, we did not observe any correlations between the cluster-sizes and the pore structure descriptors.

Recommendations for Further Work

For further studies it is recommended to:

- Conduct simulations of waterflooding at different wetting conditions in order to further investigate the effect of wettability on the correlations between different measures of pore structure and residual oil saturation.
- The correlations found should be further investigated by considering a wider range of network models.
- Find a method to relate the bond percolation threshold to the residual oil saturation.
- Continue the investigation of cluster-size distribution to potentially reveal correlations with pore structure measures.

References

Al Saadi, F., Wolf, K., Kruijsdijk, C., 2017. Characterization of fontainebleau sandstone: Quartz overgrowth and its impact on pore-throat framework. Journal of Petroleum & Environmental Biotechnology 8 (3), 1–12.

Anderson, W. G., 1987. Wettability literature survey-part 6: the effects of wettability on waterflooding. Journal of petroleum technology 39 (12), 1–605.

Andrä, H., Combaret, N., Dvorkin, J., Glatt, E., Han, J., Kabel, M., Keehm, Y., Krzikalla, F., Lee, M., Madonna, C., et al., 2013. Digital rock physics benchmarks—part i: Imaging and segmentation. Computers & Geosciences 50, 25–32.
URL http://www.sciencedirect.com/science/article/pii/S0098300412003147

Bakke, S., Øren, P.-E., et al., June 1997. 3-D pore-scale modelling of sandstones and flow simulations in the pore networks. SPE Journal 2 (02), 136–149.

Berg, C. F., Oct 2012. Re-examining archie's law: Conductance description by tortuosity and constriction. Physical Review E 86 (4).
URL https://link.aps.org/doi/10.1103/PhysRevE.86.046314

Berg, C. F., Jul 2014. Permeability description by characteristic length, tortuosity, constriction and porosity. Transport in Porous Media 103 (3), 381–400.
URL https://doi.org/10.1007/s11242-014-0307-6

Berg, C. F., 2016. Fontainebleau 3D models. http://www.digitalrocksportal.org/projects/57.

Berg, C. F., Lopez, O., Berland, H., 2017. Industrial applications of digital rock technology. Journal of Petroleum Science and Engineering 157, 131–147.
URL http://www.sciencedirect.com/science/article/pii/S0920410517305600

Blunt, M. J., 2017. Multiphase flow in permeable media: a pore-scale perspective. Cambridge University Press.

Blunt, M. J., Bijeljic, B., Dong, H., Gharbi, O., Iglauer, S., Mostaghimi, P., Paluszny, A., Pentland, C., 2013. Pore-scale imaging and modelling. Advances in Water Resources 51, 197–216.
URL http://www.sciencedirect.com/science/article/pii/S0309170812000528

Blunt, M. J., Jackson, M. D., Piri, M., Valvatne, P. H., 2002. Detailed physics, predictive capabilities and macroscopic consequences for pore-network models of multiphase flow. Advances in Water Resources 25 (8), 1069 – 1089.
URL http://www.sciencedirect.com/science/article/pii/S0309170802000490

Bryant, S., Blunt, M., Aug 1992. Prediction of relative permeability in simple porous media. Phys. Rev. A 46, 2004–2011.
URL https://link.aps.org/doi/10.1103/PhysRevA.46.2004

Carman, P. C., 1956. Flow of gases through porous media. Academic press.

Chatzis, I., Dullien, F. A. L., January 1977. Modelling pore structure by 2-D and 3-D networks with applicationto sandstones. Journal of Canadian Petroleum Technology 16 (01).

Chatzis, I., Morrow, N. R., Lim, H. T., et al., 1983. Magnitude and detailed structure of residual oil saturation. Society of Petroleum Engineers Journal 23 (02), 311–326.

Dong, H., 2008. Micro-CT imaging and pore network extraction. Ph.D. thesis, Department of Earth Science and Engineering, Imperial College London.

Dong, H., Blunt, M. J., Sep 2009. Pore-network extraction from micro-computerized-tomography images. Physical review E 80 (3).
URL https://link.aps.org/doi/10.1103/PhysRevE.80.036307

Dong, H., Fjeldstad, S., Alberts, L., Roth, S., Bakke, S., Øren, P.-E., 2008. Pore network modelling on carbonate: a comparative study of different micro-CT network extraction

methods. In: International Symposium of the Society of Core Analysts.
URL http://jgmaas.com/SCA/2008/SCA2008-31.pdf

Fatt, I., 1956. The network model of porous media. Petroleum Transactions, AIME 207, 144–181.

FEI, 2018. Lithicon is now part of FEI. https://www.fei.com/lithicon-is-now-fei/, (Accessed on 13/12/2017).

Imperial College London, 2018. Micro-CT images and networks. (Accessed on 07/10/2017).
URL http://www.imperial.ac.uk/earth-science/research/research-groups/perm/research/pore-scale-modelling/micro-ct-images-and-networks/

Jerauld, G. R., Salter, S. J., April 1990. The effect of pore-structure on hysteresis in relative permeability and capillary pressure: Pore-level modeling. Transport in Porous Media 5 (2), 103–151.
URL https://doi.org/10.1007/BF00144600

Koponen, A., Kataja, M., Timonen, J., Sep 1997. Permeability and effective porosity of porous media. Phys. Rev. E 56, 3319–3325.
URL https://link.aps.org/doi/10.1103/PhysRevE.56.3319

Larson, R., Scriven, L., Davis, H., 1981. Percolation theory of two phase flow in porous media. Chemical Engineering Science 36 (1), 57 – 73.
URL http://www.sciencedirect.com/science/article/pii/0009250981800486

Lenormand, R., Zarcone, C., et al., 1984. Role of roughness and edges during imbibition in square capillaries. In: SPE annual technical conference and exhibition. Society of Petroleum Engineers.

Lindquist, W. B., Lee, S.-M., Coker, D. A., Jones, K. W., 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).
URL http://dx.doi.org/10.1029/95JB03039

Lopez, X., Valvatne, P. H., Blunt, M. J., 2003. Predictive network modeling of single-phase non-newtonian flow in porous media. Journal of Colloid and Interface Science 264 (1), 256 – 265.

URL http://www.sciencedirect.com/science/article/pii/S0021979703003102

Mason, G., Morrow, N. R., 1991. Capillary behavior of a perfectly wetting liquid in irregular triangular tubes. Journal of Colloid and Interface Science 141 (1), 262–274.

Melrose, J., Brandner, C., 1974. Role of capillary forces in detennining microscopic displacement efficiency for oil recovery by waterflooding. Journal of Canadian Petroleum Technology 13 (04).

Nie, X., Gundepalli, V., Mu, Y., Sungkorn, R., Toelke, J., 2016. Numerical investigation of oil-water drainage and imbibition in digitized sandstones. Mechanics & Industry 17 (2).

Øren, P.-E., Bakke, S., Arntzen, O. J., et al., 1998. Extending predictive capabilities to network models. SPE journal 3 (04), 324–336.

Patzek, T., Silin, D., 2001. Shape factor and hydraulic conductance in noncircular capillaries: I. one-phase creeping flow. Journal of Colloid and Interface Science 236 (2), 295 – 304.
URL http://www.sciencedirect.com/science/article/pii/S0021979700974137

Sahimi, M., 2011. Flow and transport in porous media and fractured rock: from classical methods to modern approaches. John Wiley & Sons.

Shewchuk, J. R., 1994. An introduction to the conjugate gradient method without the agonizing pain.

Tanino, Y., Blunt, M. J., 2012. Capillary trapping in sandstones and carbonates: Dependence on pore structure. Water Resources Research 48 (8).
URL https://agupubs.onlinelibrary.wiley.com/doi/abs/10.1029/2011WR011712

Valvatne, P. H., 2004. Predictive pore-scale modelling of multiphase flow. Ph.D. thesis, Department of Earth Science and Engineering, Imperial College London.

Valvatne, P. H., Blunt, M. J., 2004. Predictive pore-scale modeling of two-phase flow in mixed wet media. Water Resources Research 40 (7), w07406.
URL http://dx.doi.org/10.1029/2003WR002627

Wildenschild, D., 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.
URL http://www.sciencedirect.com/science/article/pii/S0309170812002060

Xiong, Q., Baychev, T. G., Jivkov, A. P., 2016. Review of pore network modelling of porous media: experimental characterisations, network constructions and applications to reactive transport. Journal of Contaminant Hydrology 192, 101–117.
URL http://www.sciencedirect.com/science/article/pii/S016977221630122X

Yuan, H. H., 1981. The influence of pore coordination on petrophysical parameters. In: SPE Annual Technical Conference and Exhibition. Society of Petroleum Engineers.

Appendices

Appendix A

Attachments

A.1 The Structure of the Network Data Files

e-Core generates several ASCII files to describe the pore network in SI-units. The format of these files are used by Imperial College London, and are sometimes referred to as the Statoil format. There exist several network data files, but only those used in this thesis are described below. The data for the pores are stored in the node-files, while the data for the throats are stored in the link-files. In the following example the model contains N throats and M pores.

- link1.dat
- link2.dat
- node1.dat
- node2.dat

link1.dat

Each throat is connected to two pores, below referenced as Pore A and Pore B. The first line contains the total number of throats, N. The following N lines in the ”link1.dat”-file is structured as below:

1. Throat index (1 to N)
2. Index Pore A

3. Index Pore B
4. Inscribed radius of throat
5. Shape factor of throat
6. Length of link (from center of Pore A to center of Pore B).

link2.dat

This file contains N lines each containing the properties as structured below.

1. Throat index (1 to N)
2. Index Pore A
3. Index Pore B
4. Length of Pore A
5. Length of Pore B
6. Length of throat
7. Throat volume
8. Micro-porosity volume in throat

node1.dat

This file contains $M + 1$ lines. The first line contains the total number of pores, M, total length in the x-, y- and z-direction. The following M lines each contain the properties as structured below.

1. Pore index (1 to M)
2. x-coordinate
3. y-coordinate
4. z-coordinate
5. Number of connected pores
6. For a pore with C connections:
    (a) The following C entries are the indices of the connected pores.

(b) Entry $(C + 1)$ is the inlet-flag: 1 if the pore is connected to the inlet, 0 if not.

(c) Entry $C + 2$ is the outlet-flag: 1 if the pore is connected to the outlet, 0 if not.

(d) The following C entries are the indices of the connected throats.

A pore which is flagged as connected to a inlet/outlet, is connected to the inlet/outlet reservoir by a pore throat. The index of the inlet/outlet reservoir is -1 (inlet) or 0 (outlet).

node2.dat

This file contains M lines each containing the properties as structured below.

1. Pore index (1 to M)
2. Volume of pore
3. Inscribed radius of pore
4. Shape factor of pore
5. Micro-porosity volume in pore

A.2 Network Percolation Results.

Table A.1: Coordination number Z, bond percolation threshold and B_c for the network models used in this thesis.

Model	Z	p_{cb}	B_c
F8	2.96	0.63	1.88
F10	3.20	0.55	1.76
F13	3.62	0.41	1.49
F15	3.96	0.34	1.34
F18	4.20	0.25	1.04
F21	4.54	0.23	1.04
F25	4.90	0.18	0.87
S1	3.42	0.37	1.27
S2	4.15	0.17	0.69
S3	3.34	0.50	1.68
S4	3.07	0.52	1.61
S5	3.94	0.23	0.92
S6	5.26	0.21	1.11
S7	4.57	0.21	0.96
S8	5.11	0.17	0.86
S9	3.80	0.28	1.08
F42A	4.94	0.18	0.90
F42B	5.06	0.18	0.93
F42C	4.95	0.19	0.93
LV60A	5.37	0.15	0.80
LV60C	5.47	0.14	0.78
Bentheimer	4.41	0.20	0.87
Berea	3.96	0.22	0.86
Doddington	3.99	0.25	0.99
A1	5.79	0.13	0.74
Estaillades	4.31	0.42	1.79
Ketton	4.08	0.06	0.26
C1	5.02	0.15	0.76
C2	3.82	0.47	1.78

A.3 Waterflooding Results from e-Core

Table A.2: The residual oil saturation in the networks after two different ($\theta_A = 20^\circ - 60^\circ$ and $\theta_A = 30^\circ$) waterflood simulations in e-Core.

	S_{or}	
	θ_A	
Model	20°-60°	30°
F8	70.2 %	72.3 %
F10	66.2 %	69.5 %
F13	59.3 %	62.7 %
F15	50.6 %	53.6 %
F18	45.5 %	51.1 %
F21	39.7 %	43.9 %
F25	31.8 %	37.0 %
S1	44.9 %	52.4 %
S2	27.6 %	34.6 %
S3	54.9 %	61.9 %
S4	57.1 %	62.6 %
S5	34.4 %	43.3 %
S6	47.0 %	54.7 %
S7	31.7 %	43.4 %
S8	22.2 %	26.0 %
S9	43.6 %	47.2 %
F42A	20.0 %	26.9 %
F42B	19.1 %	24.1 %
F42C	18.1 %	26.7 %
LV60A	17.9 %	24.8 %
LV60C	18.2 %	24.4 %
Bentheimer	40.3 %	45.2 %
Berea	42.1 %	49.9 %
Doddington	44.0 %	50.0 %
A1	21.4 %	27.1 %
Estaillades	61.7 %	60.9 %
Ketton	50.2 %	55.5 %
C1	31.5 %	39.6 %
C2	56.8 %	59.8 %

A.4 Network Modification Results

Table A.3: Number and percentage of nodes and links removed from the networks.

	# removed		% removed	
Model	Nodes	Links	Nodes	Links
F8	870	705	13.3 %	7.1 %
F10	730	621	9.6 %	5.0 %
F13	302	196	3.3 %	1.2 %
F15	181	95	1.7 %	0.5 %
F18	179	101	1.6 %	0.4 %
F21	100	57	0.8 %	0.2 %
F25	70	34	0.6 %	0.1 %
S1	54	34	3.0 %	1.1 %
S2	79	47	2.7 %	0.7 %
S3	525	342	5.3 %	2.0 %
S4	1063	758	10.2 %	4.6 %
S5	29	18	5.2 %	1.6 %
S6	29	22	3.7 %	1.0 %
S7	28	13	1.8 %	0.4 %
S8	79	50	3.2 %	0.8 %
S9	37	25	5.3 %	1.8 %
F42A	112	63	1.7 %	0.4 %
F42B	61	33	0.9 %	0.2 %
F42C	31	17	0.4 %	0.1 %
LV60A	99	44	0.5 %	0.1 %
LV60C	91	51	0.4 %	0.1 %
Bentheimer	633	300	3.2 %	0.7 %
Berea	288	148	3.5 %	0.9 %
Doddington	327	185	5.2 %	1.4 %
A1	110	63	1.6 %	0.3 %
Estaillades	7128	8009	32.0 %	16.6 %
Ketton	888	675	12.4 %	4.6 %
C1	1370	2045	25.1 %	14.8 %
C2	2550	2639	34.5 %	18.2 %

A.5 Local S_{or} vs. Minimum Local Aspect Ratio.

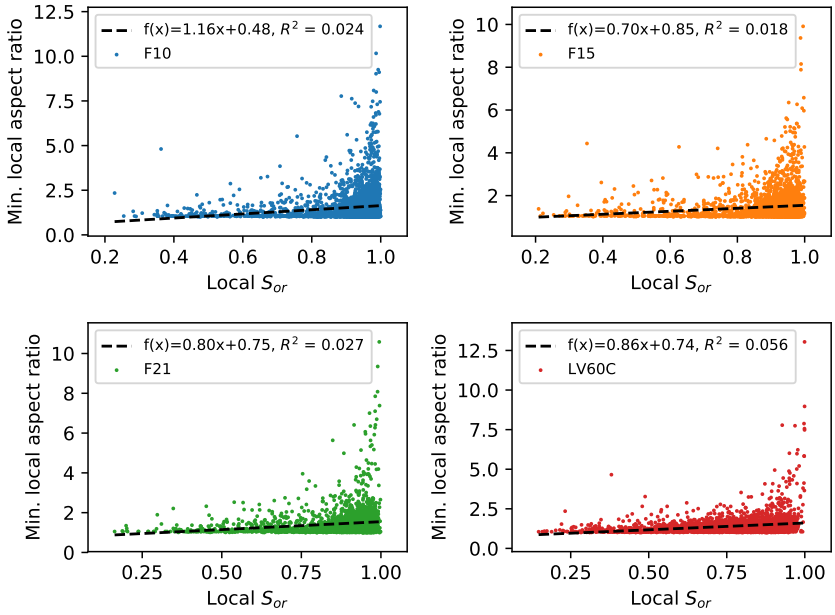

Figure A.1: Local S_{or} vs. minimum local aspect ratio for F10, F15, F21 and LV60C.

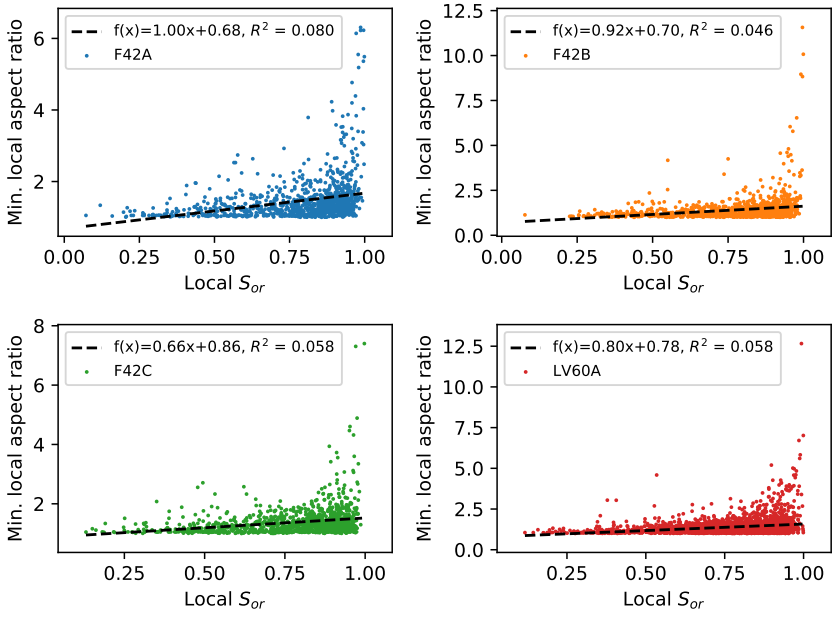

Figure A.2: Local S_{or} vs. minimum local aspect ratio for F42A, F42B, F42C and LV60A.

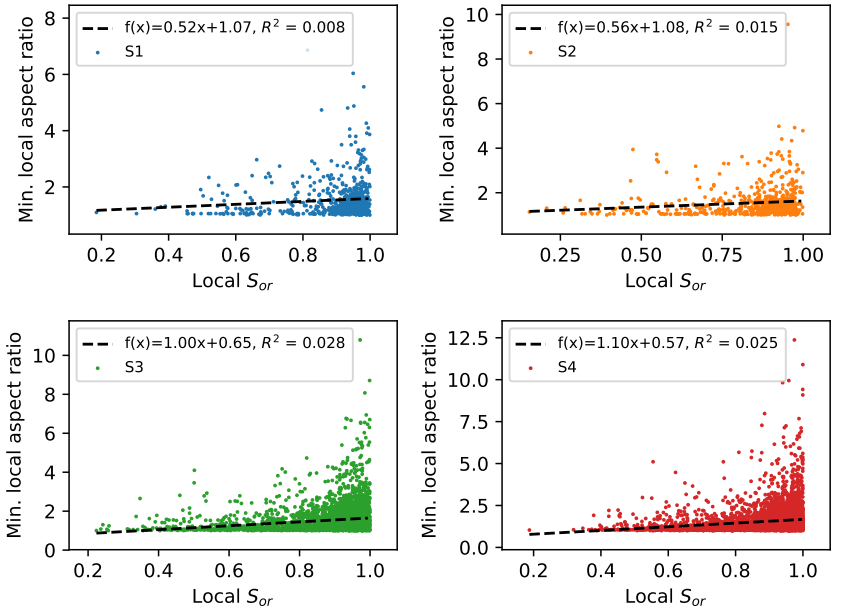

Figure A.3: Local S_{or} vs. minimum local aspect ratio for S1, S2, S3 and S4.

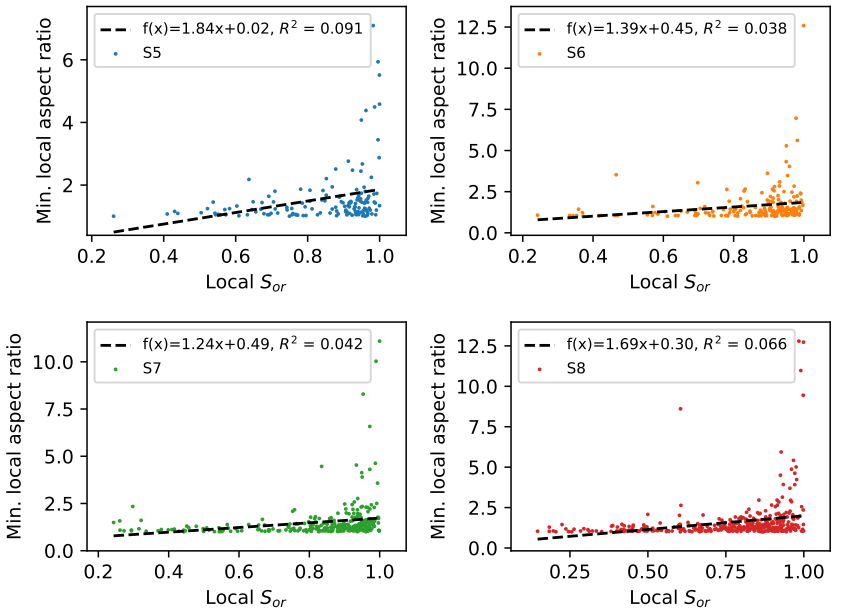

Figure A.4: Local S_{or} vs. minimum local aspect ratio for S5, S6, S7 and S8.

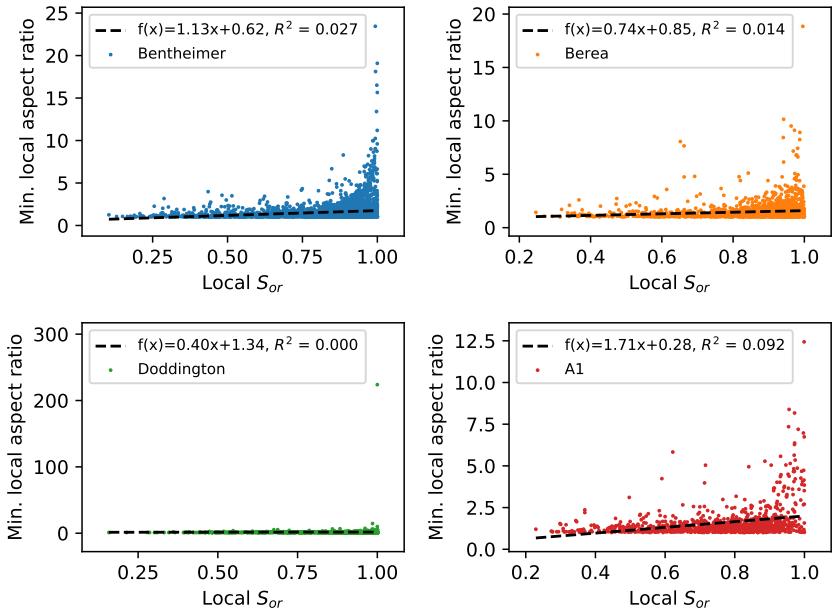

Figure A.5: Local S_{or} vs. minimum local aspect ratio for Bentheimer, Berea, Doddington and A1.

A.6 Oil Production After $k_{ro} = 0$

Table A.4: Oil produced after $k_{ro} = 0$ during waterflooding simulations using e-Core. The percentage of oil produced after $k_{ro} = 0$ is calculated by $(S_o[@k_{ro} = 0] - S_{or})/(S_{oi} - S_{or})$

Model	S_{oi}	S_{or}	S_o @ $k_{ro} = 0$	Oil produced after $k_{ro} = 0$	Average
F8	93.3 %	72.3 %	73.3 %	4.61 %	
F10	96.4 %	69.5 %	71.2 %	6.18 %	
F13	98.9 %	62.7 %	64.3 %	4.34 %	
F15	99.4 %	53.6 %	56.4 %	6.06 %	
F18	99.6 %	51.1 %	53.6 %	5.02 %	
F21	99.7 %	43.9 %	46.0 %	3.85 %	
F25	99.8 %	37.0 %	39.6 %	4.06 %	4.87 %
S1	99.0 %	52.4 %	58.8 %	13.76 %	
S2	99.5 %	34.6 %	40.4 %	8.96 %	
S3	97.0 %	61.9 %	62.9 %	2.91 %	
S4	94.9 %	62.6 %	63.7 %	3.19 %	
S5	99.7 %	43.3 %	54.9 %	20.60 %	
S6	99.5 %	54.7 %	56.8 %	4.80 %	
S7	99.6 %	43.4 %	44.7 %	2.22 %	
S8	99.7 %	26.0 %	37.1 %	15.04 %	
S9	99.6 %	47.2 %	62.4 %	29.02 %	11.17 %
F42A	99.8 %	26.9 %	28.5 %	2.17 %	
F42B	99.8 %	24.1 %	26.2 %	2.75 %	
F42C	99.9 %	26.7 %	28.4 %	2.27 %	
LV60A	99.7 %	24.8 %	25.8 %	1.31 %	
LV60C	99.8 %	24.4 %	27.2 %	3.70 %	2.44 %
Bentheimer	99.6 %	45.2 %	47.3 %	3.95 %	
Berea	99.0 %	49.9 %	51.3 %	2.81 %	
Doddington	99.0 %	50.0 %	52.9 %	5.90 %	
A1	99.7 %	27.1 %	30.6 %	4.75 %	4.35 %
Estaillades	80.5 %	60.9 %	63.7 %	14.18 %	
Ketton	98.5 %	55.5 %	59.2 %	8.71 %	
C1	89.0 %	39.6 %	56.8 %	34.81 %	
C2	78.7 %	59.8 %	67.3 %	39.74 %	24.36 %

A.7 Cluster-Size Distribution Results

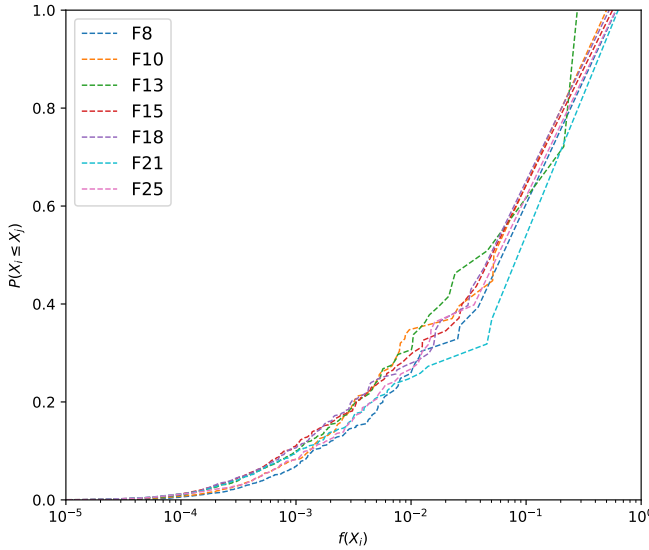

Figure A.6: Cumulative cluster-size distribution $P(X_i \leq X_j)$ vs. relative cluster-size $F(X_i)$ for the Fontainebleau networks.

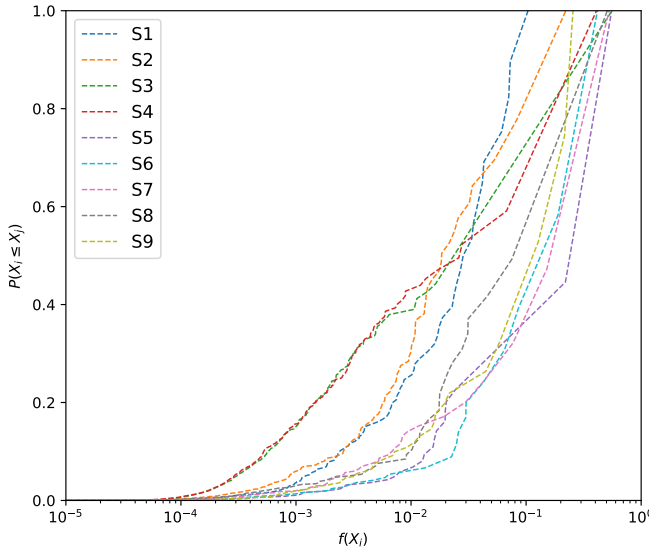

Figure A.7: Cumulative cluster-size distribution $P(X_i \leq X_j)$ vs. relative cluster-size $F(X_i)$ for the ICL Sandstone networks.

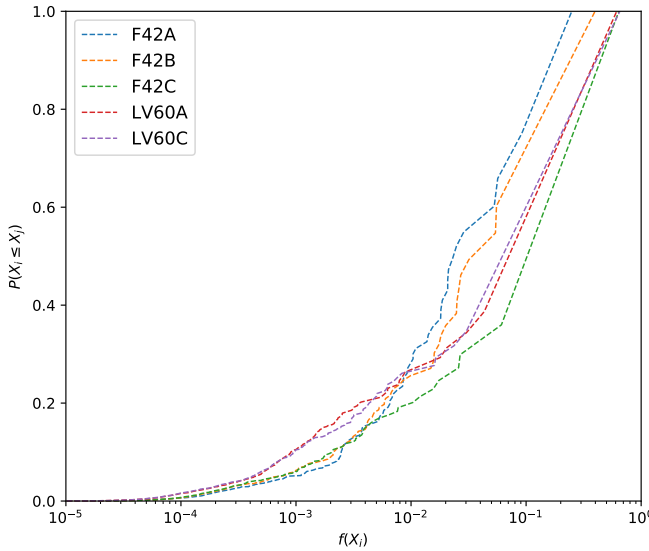

Figure A.8: Cumulative cluster-size distribution $P(X_i \leq X_j)$ vs. relative cluster-size $F(X_i)$ for the ICL Sand Pack networks.

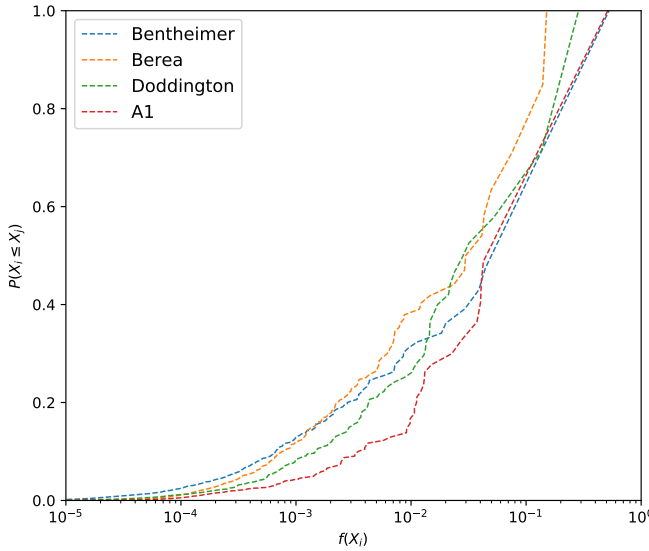

Figure A.9: Cumulative cluster-size distribution $P(X_i \leq X_j)$ vs. relative cluster-size $F(X_i)$ for different ICL sandstone networks.

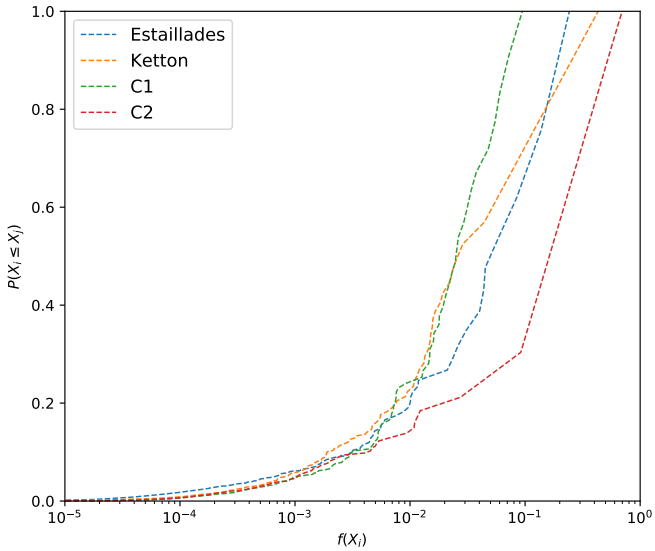

Figure A.10: Cumulative cluster-size distribution $P(X_i \leq X_j)$ vs. relative cluster-size $F(X_i)$ for ICL Carbonate networks.

Appendix B

Scripts

B.1 readme.txt

```
##########################################################################
This is the readme-file for the suite of scripts developed to conduct
calculations on network models.

All scripts has to be located in the same folder, and called from a
folder with network data files.

Anders Torland, anderstorland@gmail.com, 2018
##########################################################################

--------------------------------------------------------------------------------

CONTENTS
--------------------------------------------------------------------------------


1. SCRIPTS.
        1.1.  laplacePN.py
        1.2.  netPotential.py
        1.3.  netStream.py
        1.4.  streamUtils.py
        1.5.  netPlot.py
        1.6.  netRecon.py
        1.7.  updateNetwork.py
        1.8.  percolation.py
        1.9.  clusterTrack.py
```

---

1. SCRIPTS

1.1. laplacePN.py :

This script constists of functions used to to calculate the solution of the Laplace equation on a network model by sparse matrix inversion. The functions in this script is used in 'netPotential.py' and in 'netStream.py'.

1.2. netPotential.py :

This script is executable and use the functions in 'laplacePN.py'. The script has an argument parser with several optional and mandatory arguments, given below:

```
-h, --help: show this help message and exit
-initialguess INITIALGUESS: Set initial guess: Either 'linear' or
    'zero'
-solvefor SOLVEFOR:     Solve for permeability or formation factor
     ('perm' or 'FF').
-plot2D: Plot 2D pore network (Z vs. Y)
-plot3D: Plot 3D pore network
-plotAspect: Plot aspect ratio vs. Sor
-output: If text output is wanted.
-network NETWORK: Specify if calculations should be conducted on '
    original' or 'updated' network data files.
```

Execution example: 'python netPotential.py -solvefor perm -output -plot2D -network original'

This would calculate the pressure in each pore and det permeability of the original network which the script is executed from. A 2D plot of the nodes and links would also be produced and opend, and a text output of the most used properties would be written to the terminal.

.....

1.3. netStream.py :

.....

This script is executable and tracks the streamlines/tubes in a network model. In order to avoid numerical instabilities and problems during the tracking it is recommended to run this script on updated network files (see 'netRecon.py' and 'updateNetwork.py' below). The plotting is performed in 'netPlot.py'.

Arguments:

-h, --help: show this help message and exit
-plot: Plot streamlines on 2D model.
-sort SORT: Sort links by either 'location' or 'angles'

Execution example: 'python netStream.py -sort angles - plot'

This would track the streamlines based on the angle between the inlet link and the outlet link, and plot the streamlines in 2D.

.....

1.4. streamUtils.py :

.....

A set of help-functions used in 'netStream.py'

.....

1.5. netPlot.py :

.....

A script with functions to plot the streamlines in 2D through a network model and the full network model in either 2D or 3D.

.....

1.6 netRecon.py :

.....

The script contains several different functions to extract information and modify network data files.
Some of the functions are given and explained below:

```
- get_network_file
        Reads the network data files ('node1.dat', 'node2.dat', 'link1.dat
            ' and 'link2.dat'). Automatically skips the headers.
- get_surface_nodes
        Returns the index of the inlet and outlet nodes.
- get_dimensions
        Returns the dimensions of the network model (length, width, depth)
- get_zero_coord_nodes
        Return the number of nodes with zero coordination number
- map_connections
        A "viurs-spreading" algorithm to map connectivity between nodes
            from a given set of starting nodes.
- remove_nonSpanning
        This function utilize multiple other functions in the same script
            to remove all nodes and links which are not connected to both
            the inlet and the outlet (i.e. non-spanning).
- print_network_to_file
        This function will print a network file to a new '.dat' file. Used
             when networks are updated and saved. The original data files
            are automatically moved to 'originalNetworks/' by the function
             'move_network_files'.
- rearrange_network
        When the network data files are updated this function will
            renumber all nodes and links such that the numbering
            convention is consitent with 1 to N and 1 to M (N=total number
             of nodes, M=total number of links).
- get_coordination_number
        Returns the average coordination number
- get_redundant_links:
        Finds the number of redundant links which connect the same two
            pores.
        Not used and computationally demanding. Shound be rewritten
            similarly to how the effective porosity or number of unused
            nodes are calculate in 'netStream.py'.
- get_local_aspect_ratio
        Returns the average local aspect ratio and the minimum local
            aspect ratio.
        The script currently only extracts these quantities from the
            original network. Can be rewritten to calculate values from
            updated networks ('nodesat_wf.dat' has to be reduced to only
            contain the nodes in the updated network data files.)
- output_network_mod_file
        Print some of the network modification parameters to a text-file
            after a network is updated.
- print network_statistics
```

```
        This function calculates the most important network statistics
            such as:
                - Number of nodes & links (and inlet/outlet nodes/links)
                - Global aspect ratio
                - Average local aspect ratio (of original network)
                - Minimum local aspect ratio (of original network)
                - Average connection number
                - Node and link radius metrics.
- get_coord
        Returns all the coordinates of all the nodes.

.....................................................................

1.7. updateNetwork.py                          :
.....................................
This script is executable and should be executed before tracking
    streamlines with 'netStream.py'. The script only need so be executed
    once for each network. This script use the functions in 'netRecon.py'
    to update (remove non-spanning), rearrange and print new network files
    . The script has to be run from a folder with original data files.

.....................................................................

1.8. percolation.py                             :
.....................................
This script is executable and finds the bond percolation treshold. The
    bonds (links) are sorted from smallest to largest by radius, and
    removed in acending order until the connection between outlet and
    inlet is found. The function 'map_connections' (in 'netRecon.py') is
    used to check the connectivity of the network. A triangular pore
    throat shape is assumed (% of non-triangular pores is outputted) and
    the angles of the triangle are calculated by method propesed by Patzek
     and Silin (2001). A interfacial tension and contact angle has to be
    specified within the scripts. The threshold capillary pressure for
    snap-off for the percolation radius is calculated (the threshold
    capillary pressure for snap-off assuming an equilateral triangle is
    also calculated).

In order to reduce the runtime the bisection method is used.

.....................................................................

1.9. clusterTrack.py                            :
.....................................
This script is semi-executable. Since it was developed towards the end of
    my master's thesis, it is not completely finished. The script contains
```

```
     several functions (described below), whereas some of them are used
    with in other scripts (from 'createNetworkXML.py'). To execute
    functions in this script the functions has to be manually called from
    the bottom of the script.

This script has two main functions:
        1) This script locates the oil clusters after waterflooding in a
            network model using the e-Core data file 'nodesat_wf.py'. The
            script counts all oil saturated nodes (So>0), locates all the
            clusters (with the index of each node in it), the average
            cluster size, number of singlet clusters ('clusters' with only
             one node) and the size of the maximum cluster. It also
            calculates the relative size of the maximum cluster and the
            oil saturated nodes.
                Functions:
                - get_cluster_size()
                - locate_clusters()
                - print_cluster_stats(...)

        2) The script calculates the cumulative cluster-size distribution.
                Functions:
                - get_cumulative_clust_dist()
                - plot_cumulative_clust_dist(...)
                - print_cumulative_dist(...)

.........................................................................

1.10. calcPoreStruct.py                        :
.................................
Can plot local aspect ratio vs. local Sor. Uses the e-Core data file '
    nodesat_wf.py' after waterflooding. Two methods of data-binning are
    used: equal count binning and equal interval binning. The size of the
    bins can be specified in the script. Linear regression is also
    conducted on the data-sets. The plots are saved in '<network X>/output
    '.

The script is used with in 'plotResults.py'.

.........................................................................

1.11. createNetworkXML.py                      :
.................................
This script is written to produce ball and stick models using the Windows
    10 application 'Ball&Stick' by 'fourelem'. The script write XML-files
    files. If no argument is given all the nodes and links will be plotted
     using the classical red-and-white layout.
```

```
The script has an argument parser with the following arguments:
        -h, --help:             show this help message and exit
        -colorResidual  Color nodes with residual oil saturation.
        -onlyResidual   Only show nodes with residual oil saturation.
        -onlyWater      Only show nodes with water saturation = 1.

Execution example:      'python createNetworkXML.py -colorResidual

This would create an XML-file of the network where only the oil filled
    nodes are colored. (The water bearing nodes are also plotted to
    maintain the size of the models. They are plotted with a white color
    so they are not visible when using a white background.)

.....................................................................

1.12. plotResults.py                         :
.....................................
The script plots the results for my master thesis. The results are loaded
    from text files. The script is written specifically for these results,
     and can not easily be used for other purposes.

.....................................................................

1.13. plotCumDist.py                         :
.....................................
The script plots the results for cumulative cluster-size distribution for
    my master thesis. The results are loaded from text files. The script
    is written specifically for these results, and can not easily be used
    for other purposes.

---------------------------------------------------------------------------

2. SPECIAL PACKAGES
---------------------------------------------------------------------------

.....................................................................

12.1. tqdm                                                  :
.....................................
'tqdm' is a packaged used to display a progress bar in the terminal window
    . It is used to keep track, and estimate the remaining time, of time-
    consuming processes (for-loops).

If you do not wish to install/use 'tqdm' you can delete the import command
     and the use of it in the scripts. This will not affect the results.
```

```
Installation and usage:
For mac users (like myself) 'tqdm' can be installed via pip (if you do not
     have pip; pip can be installed by running 'sudo easy_install pip' in
    the terminal window) with the command: 'pip install tqdm'. Once
    installed it needs to be imported into the scripts where its used by
    writing: 'from tqdm import *'.

To use 'tqdm' with a for-loop, simply write tqdm(...) around the range of
    the loop. Example: 'for i in tqdm(range(1,10)):'

--------------------------------------------------------------------------------

3. STRUCTURE OF DIRECTORY
--------------------------------------------------------------------------------

<network-X>
        \output (1)
        \originalNetworks (2)

        (1) The output folder must be created manually. This is where most
             of the plots and text outputs are stored.
        (2) This folder will be created by running 'updateNetwork.py'.
            Most of the scripts check if this folder exists to check if
            updated networks are created. If it exists and is empty some
            the scripts will not work. Therefore if updated networks are
            removed and replaced by original networks it is recommended to
             delete or rename the folder.

--------------------------------------------------------------------------------

4. NETWORK DATA FILES
--------------------------------------------------------------------------------

Required network data files:
        - 'node1.dat'
        - 'node2.dat'
        - 'link1.dat'
        - 'link2.dat'

Optional files (required for 'clusterTrack.py' and the argument '
    plotAspect' in 'netPotentialpy'):
        - 'nodesat_wf.dat'

--------------------------------------------------------------------------------
```

Some scripts use the updated networks, while others use the original. Even though the user, in most cases, is notified when the wrong network is used, it is highly recommended to update all networks by running 'updateNetwork.py'. This way both networks can be accessed by the scripts.

B.2 netPotential.py

```
1  import sys
2  import os
3  import argparse
4  import numpy as np
5  import scipy
6  import scipy.sparse as sparse
7  import scipy.sparse.linalg
8  import time
9  import csv
10 import matplotlib.pyplot as plt
11 import math
12
13 # Selfwritten scripts
14 import netRecon
15 import laplacePN
16 import netPlot
17
18 parser = argparse.ArgumentParser(prog='laplacePN.py', description='A␣
       program␣to␣calculate␣permeability␣and␣flowrate␣in␣a␣pore␣network.')
19 parser.add_argument("-initialguess", type=str, help="Set␣initial␣guess:␣
       Either␣'linear'␣or␣'zero'", default='zero')
20 parser.add_argument("-solvefor", type=str, help="Solve␣for␣permeability␣or
       ␣formation␣factor␣('perm'␣or␣'FF')." , default='perm')
21 parser.add_argument("-plot2D", action='store_true', help="Plot␣2D␣pore␣
       network␣(Z␣vs.␣Y)", default=0)
22 parser.add_argument("-plot3D", action='store_true', help="Plot␣3D␣pore␣
       network", default=0)
23 parser.add_argument("-output", action='store_true', help="If␣text␣output␣
       is␣wanted.")
24 parser.add_argument("-network", type=str, help="Specify␣if␣calculations␣
       should␣be␣conducted␣on␣'original'␣or␣'updated'␣network␣data␣files.",
       default='original')
25 args = parser.parse_args()
26
```

```
start = time.clock() # Stating the clock

t1 = time.clock()
print "Importing␣network␣files..."
dest = os.getcwd()+'/originalNetworks'
if args.network == 'original':
  if os.path.exists(dest):
    node1 = netRecon.get_network_file('originalNetworks/node1')
    node2 = netRecon.get_network_file('originalNetworks/node2')
    link1 = netRecon.get_network_file('originalNetworks/link1')
    link2 = netRecon.get_network_file('originalNetworks/link2')
  else:
    node1 = netRecon.get_network_file('node1')
    node2 = netRecon.get_network_file('node2')
    link1 = netRecon.get_network_file('link1')
    link2 = netRecon.get_network_file('link2')
elif args.network == 'updated':
  if os.path.exists(dest):
    print "*"*80, "WARNING:␣Network␣statistic␣calculations␣should␣be␣
        conducted␣on␣original␣network"
    print "*"*80
    node1 = netRecon.get_network_file('node1')
    node2 = netRecon.get_network_file('node2')
    link1 = netRecon.get_network_file('link1')
    link2 = netRecon.get_network_file('link2')
  else:
    print "*"*80
    print "ERROR:␣␣␣Updated␣network␣not␣created.␣Run␣'updateNetwork.py'␣"
    print "␣to␣remove␣non-spanning␣nodes␣and␣links."
    print "*"*80
    sys.exit()
else:
  print "*"*80
  print "ERROR:␣Specify␣valid␣network:␣'original'␣or␣'updated'."
  print "*"*80
  sys.exit()
t2 = time.clock()
print "Time␣to␣import␣network␣files:␣%.2f␣sec." % (t2-t1)

p_in = 2
p_out = 1

xvec, g_nodes, g_throats, gL_link, g_inlet, g_outlet, in_nodes, out_nodes,
     cum_pore_radius, cum_throat_radius, link_length, in_link, out_link =
    laplacePN.calc_potential(node1, node2, link1, link2, 2,1, args.
    solvefor)
```

```
69  inflow, outflow, inlet_counter, outlet_counter = laplacePN.
        calc_surface_flowrates(g_inlet, g_outlet, in_nodes, out_nodes, p_in,
        p_out, xvec)
70  perm_mdarcy = laplacePN.calc_permeability(p_in, p_out, inflow)
71  avg_pore_radius, max_pore_radius, min_pore_radius, avg_link_radius,
        avg_con_num, porosity = laplacePN.calc_network_statistics(node1, node2
        ,link1,link2)
72  q_vec = laplacePN.calc_flowrate(link1, gL_link, p_in, p_out, xvec)
73
74  # Output of results.
75  if args.output:
76    print "-+-+ Results (",args.network, "network )", "-+"*23
77    if args.solvefor == 'perm':
78      print "Permeability = %.2f mD" % perm_mdarcy
79    if args.solvefor == 'FF':
80      # Calculation of formation factor
81      FF = laplacePN.calc_formation_factor(p_in, p_out, inflow)
82      print "Formation factor: %.2f" % FF
83      print "F poro. approx.: %.2f" % (1/(porosity/100)**2)
84    print "Flowrate - Inlet: %.4E m3" % inflow
85    print "Flowrate - Outlet: %.4E m3" % outflow
86    print " "
87    netRecon.print_network_statistics(node1, node2, link1, link2, args.
          network)
88
89  if args.plot2D+args.plot3D!=0:
90    if args.plot2D:
91      netPlot.plot_network('2D')
92    elif args.plot3D:
93      netPlot.plot_network('3D')
```

B.3 laplacePN.py

```
1   ######################################################################
2   #
3   # A code to calculate the solution of the Laplace equation on a
4   # network model by sparse matrix inversion.
5   # Anders Torland, anderstorland@gmail.com, 2018
6   #
7   import numpy as np
8   import scipy
9   import scipy.sparse as sparse
10  import scipy.sparse.linalg
11  import time
12  import math
```

```
13
14  # Selfwritten scripts
15  import netRecon
16
17  start = time.clock() # Stating the clock
18  def calc_potential(node1, node2, link1, link2, p_in, p_out, solvefor, **
        saturation):
19    # Calculating conductivity, g, for all pores and throats
20    #Using A=r_inc**2/(4*G) from Mason and Morrow 1990 and Eq. 18 from Oren
          1998
21    visc = 1    #IF CHANGED: CHANGE IN 'calc_permeability(..)'
22    def calc_g_nodes(node2, argument):
23      g_nodes=[]
24      cum_pore_radius = []
25      for i in range(len(node2)):
26        if argument == 'perm':
27          g_nodes.append((3*node2[i][2]**4)/(80*node2[i][3]*visc))
28          #g_nodes.append((3*(math.pow((math.pi*node2[i,2]**2),2))*node2[i
                ,3])/(5*1))
29        elif argument == 'FF':
30           #Using A=r_inc**2/(4*G) from Mason and Morrow 1990
31          g_nodes.append(node2[i][2]**2/(4*node2[i][3]))
32          if 'bodySaturation' in saturation:
33            g_nodes.append(saturation['bodySaturation'][i]*node2[i
                  ][2]**2/(4*node2[i][3]))
34          else:
35            g_nodes.append(node2[i][2]**2/(4*node2[i][3]))
36          #g_nodes.append(math.pi*node2[i,2]**2)
37        cum_pore_radius.append(node2[i][2])
38      return g_nodes, cum_pore_radius
39
40    def calc_g_throats(link1, argument):
41      g_throats=[]
42      cum_throat_radius = []
43      for i in range(len(link1)):
44        if argument == 'perm':
45          g_throats.append((3*link1[i][3]**4)/(80*link1[i][4]*visc))
46          #g_throats.append((3*(math.pow((math.pi*link1[i,3]**2),2))*link1[i
                ,4])/(5*1))
47        elif argument == 'FF':
48          #Using A=r_inc**2/(4*G) from Mason and Morrow 1990
49          if 'throatSaturation' in saturation:
50            g_throats.append(saturation['throatSaturation'][i]*link1[i
                  ][3]**2/(4*link1[i][4]))
51          else:
52            g_throats.append(link1[i][3]**2/(4*link1[i][4]))
```

```
53          #g_throats.append(math.pi*link1[i,3]**2)
54        cum_throat_radius.append(link1[i][3])
55      return g_throats, cum_throat_radius
56
57    def calc_g_link(link1,link2,g_nodes, g_throats):
58      # Calculating link-conductivity (Oren, 1998)
59      gL_link=[]
60      gL_link_solv=[]
61      denominator=0
62      link_length=[]
63      g_weigth=1
64      # for-loop: Running through the total number of links.
65      for i in range(len(link1)):
66        if g_throats[i]<=0.0:
67          gL_link.append(0.0)
68          # The link-cond/length was scaled by 1e15. Not needed anymore.
69          gL_link_solv.append(0.0)
70        else:
71          # if: Checking if link is flagged as "inlet".
72          if int(link2[i][1])==-1:
73            if g_nodes[int(link2[i][2])-1]<=0.0:
74              gL_link.append(0.0)
75              gL_link_solv.append(0.0)
76            else:
77              denominator=(((link2[i][5]+link2[i][3])/g_throats[i])+g_weigth
                    *(link2[i][4]/g_nodes[int(link2[i][2])-1]))
78              gL_link.append(1/denominator)
79              gL_link_solv.append(1/denominator)
80          # if: Checking if link is flagged as "outlet".
81          elif int(link2[i][2])==0:
82            if g_nodes[int(link2[i][1])-1]<=0.0:
83              gL_link.append(0.0)
84              gL_link_solv.append(0.0)
85            else:
86              denominator=(((link2[i][5]+link2[i][4])/g_throats[i])+g_weigth
                    *(link2[i][3]/g_nodes[int(link2[i][1])-1]))
87              gL_link.append(1/denominator)
88              gL_link_solv.append(1/denominator)
89          else:
90            if g_nodes[int(link2[i][1])-1]<=0.0 or g_nodes[int(link2[i][2])
                  -1] <= 0.0:
91              gL_link.append(0.0)
92              gL_link_solv.append(0.0)
93            else:
94              denominator=((link2[i][5]/g_throats[i])+g_weigth*((link2[i
                    ][3]/g_nodes[int(link2[i][1])-1])+(link2[i][4]/g_nodes[int
```

```
                (link2[i][2])-1])))
            gL_link.append(1/denominator)
            gL_link_solv.append(1/denominator)
        link_length.append(link1[i][5])
    return gL_link_solv, gL_link, link_length

def create_equation_matrix(node1, gL_link, gL_link_solv, p_in, p_out):
    num_node=int(node1[-1][0])
    # Defining different indexing vectors.
    row=[]
    col=[]
    data=[]
    b_vec=[]
    g_inlet=[]
    g_outlet=[]
    in_nodes=[]
    in_link=[]
    out_nodes=[]
    out_link=[]

    # for-loop: Running through the total number of nodes.
    for i in range(num_node):

      num_con=int(node1[i][4])
      # Setting up b-vector.
      # if: True if inlet.
      if int(node1[i][5+num_con]) == 1:
        ind=node1[i].index(-1)
        b_vec.append(-p_in*gL_link_solv[int(node1[i][ind+num_con+2])-1])
        row.append(i)
        col.append(i)
        data.append(-gL_link_solv[int(node1[i][ind+num_con+2])-1])

        ## For rate calculations over inlet
        g_inlet.append(gL_link[int(node1[i][ind+num_con+2])-1])
        in_nodes.append(i+1)
        in_link.append(int(node1[i][ind+num_con+2]))
      # if: True if outlet
      elif int(node1[i][6+num_con]) == 1:
        ind=node1[i].index(0)
        b_vec.append(-p_out*gL_link_solv[int(node1[i][ind+num_con+2])-1])
        row.append(i)
        col.append(i)
        data.append(-gL_link_solv[int(node1[i][ind+num_con+2])-1])

        ## For rate calculations over outlet
```

```
140           g_outlet.append(gL_link[int(node1[i][ind+num_con+2])-1])
141           out_nodes.append(i+1)
142           out_link.append(int(node1[i][ind+num_con+2]))
143
144         else:
145           b_vec.append(0)
146         # for-loop: Running "connection number"-times through each line,
147         # finding all connected nodes to current node, and setting up p-
              matrix.
148         for j in range(num_con):
149           # if: True if connection number for current node > 0
150           # AND connected node is NOT inlet/outlet.
151
152           if int(node1[i][5+j])>0 and int(node1[i][4])>0:
153             # For the connecting nodes
154             row.append(i)
155             col.append(int(node1[i][5+j])-1)
156             data.append(gL_link_solv[int(node1[i][7+num_con+j])-1])
157             # the node itself
158             row.append(i)
159             col.append(i)
160             data.append(-gL_link_solv[int(node1[i][7+num_con+j])-1])
161
162     # Constructing the sparse matrix from col,row and data
163     mtx = sparse.csc_matrix((data, (row, col)), shape=(num_node,num_node))
164     b_v=np.asarray(b_vec)
165     return mtx, b_v, g_inlet, g_outlet, in_nodes, out_nodes, in_link,
            out_link
166
167   def create_initial_guess(node1,guess, p_in, p_out):
168     num_node=int(node1[-1][0])
169     def linear_initial_guess(num_node,p_in,p_out):
170       length, width, depth = netRecon.get_dimensions()
171       p_L=(p_in-p_out)/length
172       for i in range(num_node):
173         x0vec[i]=p_in-node1[i][1]*p_L
174       return x0vec
175
176     x0vec = np.zeros(num_node,dtype=np.float)
177     if guess == 'linear':
178       x0vec = linear_initial_guess(num_node,p_in,p_out)
179
180   def CG_potential_solver(mtx, b_v, initial_guess, tolerance):
181     #Solving by CG-method.
182     tic=time.clock()
183     # Solving for pressures with CG.
```

```
184       xvec = scipy.sparse.linalg.cg(mtx, b_v, initial_guess, tol=tolerance)
185       toc=time.clock()
186       CG_runtime=toc-tic
187       return xvec[0]
188
189     g_nodes, cum_pore_radius = calc_g_nodes(node2, solvefor)
190     g_throats, cum_throat_radius = calc_g_throats(link1, solvefor)
191     gL_link_solv, gL_link, link_length = calc_g_link(link1,link2,g_nodes,
            g_throats)
192     mtx, b_v, g_inlet, g_outlet, in_nodes, out_nodes, in_link, out_link =
            create_equation_matrix(node1, gL_link, gL_link_solv, p_in, p_out)
193     x0vec = create_initial_guess(node1,'linear', p_in, p_out)
194     xvec = CG_potential_solver(mtx, b_v, x0vec, 1E-15)
195
196     return xvec, g_nodes, g_throats, gL_link, g_inlet, g_outlet, in_nodes,
            out_nodes, cum_pore_radius, cum_throat_radius, link_length, in_link,
             out_link
197
198 def calc_surface_flowrates(g_inlet, g_outlet, in_nodes, out_nodes, p_in,
        p_out, xvec):
199 # Calculating flowrate at inlet and outlet
200     inflow=0
201     outflow=0
202     inlet_counter=0
203     outlet_counter=0
204     # for-loop: Running through all inlet pores.
205     # Calculating flowrate at current inlet-pore.
206     for i in range(len(in_nodes)):
207       inflow+=(g_inlet[i]*(p_in-xvec[int(in_nodes[i])-1]))
208       inlet_counter+=1
209       # for-loop: Running through all outlet pores.
210       # Calculating flowrate at current outlet-pore.
211     for i in range(len(out_nodes)):
212       outflow+=(g_outlet[i]*(p_out-xvec[int(out_nodes[i])-1]))
213       outlet_counter+=1
214     return inflow, outflow, inlet_counter, outlet_counter
215
216 def calc_permeability(p_in, p_out, inflow):
217     # Permeability calculation
218     length, width, depth = netRecon.get_dimensions()
219
220     dP=p_in-p_out            # Pressure difference over inlet.
221     area=width*depth           # Inlet area:     total_y_length *
            total_z_length
222     visc=1               # Viscosity:    Defined as 1 throughout.
223     perm=(inflow*length*visc)/(area*dP) # Permeability: By Darcys Law (in SI
```

```
        -unit: m^2)
224   perm_mdarcy=perm*1.01*10**15        # Permeability: In Darcy-unit: mD
          (1.01*12^-12 m^2 = 1 mD)
225
226   return perm_mdarcy
227
228 def calc_formation_factor(p_in, p_out, inflow):
229   length, width, depth = netRecon.get_dimensions()
230   area=width*depth
231   FF=((area*(p_in-p_out))/(inflow*length))
232   return FF
233
234 def calc_network_statistics(node1,node2,link1,link2):
235   length, width, depth = netRecon.get_dimensions()
236   area=width*depth
237   num_node=int(node1[-1][0])
238   # Porosity calculation and link/node statistics
239   pore_volume=0
240   pore_radius=[]
241   link_radius=0
242   total_connections=0
243   # for-loop: Running through all links.
244   for i in range(len(link2)):
245     pore_volume+=link2[i][6]+link2[i][7]
246     link_radius+=link1[i][3]
247   # for-loop: Running through all pores.
248   for i in range(len(node2)):
249     pore_volume+=node2[i][1]+node2[i][4]
250     pore_radius.append(node2[i][2])
251     total_connections+=node1[i][4]
252   avg_pore_radius=(sum(pore_radius)/num_node)*10**6
253   max_pore_radius=max(pore_radius)*10**6
254   min_pore_radius=min(pore_radius)*10**6
255   avg_link_radius=(link_radius/int(link1[-1][0]))*10**6
256   avg_con_num=total_connections/num_node
257   porosity=(pore_volume*100)/(length*area)
258   return avg_pore_radius, max_pore_radius, min_pore_radius,
          avg_link_radius, avg_con_num, porosity
259
260 def calc_flowrate(link1, gL_link, p_in, p_out, xvec):
261   # Set up q in each link
262   q_vec=[]
263   # for-loop: Running through all inlet pores.
264   for i in range(int(link1[-1][0])):
265     c1=int(link1[i][1])
266     c2=int(link1[i][2])
```

```
    if c1==-1:
      q_vec.append(gL_link[i]*(p_in-xvec[c2-1]))
    elif c2==0:
      q_vec.append(gL_link[i]*(xvec[c1-1]-p_out))
    else:
      q_vec.append(gL_link[i]*(xvec[c2-1]-xvec[c1-1]))
  q_vec=np.asarray(q_vec)
  q_vec=abs(q_vec)

  return q_vec
```

B.4 netStream.py

```
#######################################################################
#
# A code to trace the network-streamlines and then calculate
# tortuosity and constriction factor.
# Anders Torland, anderstorland@gmail.com, 2018
#
import sys
import os
import argparse
import numpy as np
import math
from numpy import linalg as LA
import time
import matplotlib.pylab as plt
from tqdm import *
from operator import itemgetter

#Selfwritten scripts
import netRecon
import netPlot
import laplacePN
import streamUtils

parser_stream = argparse.ArgumentParser(prog='netStream.py', description='
    A␣script␣to␣track␣streamlines␣in␣a␣pore␣network.')
parser_stream.add_argument("-plot", action='store_true', help="Plot␣
    streamlines␣on␣2D␣model.")
parser_stream.add_argument("-sort", type=str, help="Sort␣links␣by␣either␣'
    location'␣or␣'angles'", default='angles')
parser_stream.add_argument("-solvefor", type=str, help="Solve␣for␣
    permeability␣or␣formation␣factor␣('perm'␣or␣'FF').", default='perm')
parser_stream.add_argument("-phaseConductancePD", action='store_true',
```

```
        help="Calcualate␣conductance␣on␣the␣water-phase␣only.␣Need␣
        additionally␣the␣files␣linksat_pd.dat␣and␣nodesat_pd.dat.")
29  parser_stream.add_argument("-saveResultsToFile", action='store_true', help
        ="Save␣most␣important␣results␣to␣the␣text␣file␣output.txt.")
30  args_stream = parser_stream.parse_args()
31
32  mainTic = time.clock()
33  foldername = os.path.basename(os.getcwd())
34  filename = os.path.basename(__file__)
35  print ":::::␣Running␣%s␣on␣network␣model:␣%s␣:::::" % (filename,
        foldername)
36  print args_stream.phaseConductancePD
37
38  dest = os.getcwd()+'/originalNetworks'
39  if not os.path.exists(dest):
40    print "*"*80
41    print "WARNING:\tModified␣network␣not␣created,␣and␣may␣contain␣non-
          spanning\n\t\tnodes␣and␣links.␣Run␣'updateNetwork.py'␣to␣delete␣\n\t
          \tunreachable␣nodes␣and␣links."
42    print "*"*80
43
44  link1 = netRecon.get_network_file('link1')
45  link2 = netRecon.get_network_file('link2')
46  node2 = netRecon.get_network_file('node2')
47  node1 = netRecon.get_network_file('node1')
48
49  # Using laplacePN.py to solve for the pressure.
50  print "Solving␣for␣potential...",
51  sys.stdout.flush()
52  tic = time.clock()
53  p_in = 2
54  p_out = 1
55  if args_stream.phaseConductancePD:
56    throatSaturationPD=netRecon.get_saturation_file('link','pd')
57    bodySaturationPD=netRecon.get_saturation_file('node','pd')
58    pres, g_nodes, g_throats, gL_link, g_inlet, g_outlet, in_nodes,
          out_nodes = laplacePN.calc_potential(node1, node2, link1, link2,
          p_in, p_out, args_stream.solvefor,throatSaturation=
          throatSaturationPD,bodySaturation=bodySaturationPD)[0:8]
59  else:
60    pres, g_nodes, g_throats, gL_link, g_inlet, g_outlet, in_nodes,
          out_nodes = laplacePN.calc_potential(node1, node2, link1, link2,
          p_in, p_out, args_stream.solvefor)[0:8]
61  q_vec = laplacePN.calc_flowrate(link1, gL_link, p_in, p_out, pres)
62  inflow_laplace = laplacePN.calc_surface_flowrates(g_inlet, g_outlet,
        in_nodes, out_nodes, p_in, p_out, pres)[0]
```

```
63  presDiff = [p_in, p_out]
64  q_vec=np.asarray(q_vec)
65  toc = time.clock()
66  print "     [DONE] - %.2f sec." % (toc-tic)
67
68  print "Setting up for streamline tracking...",
69  sys.stdout.flush()
70  tic = time.clock()
71  numberOfNodes = int(node2[-1][0])
72  # Setting pressures at inlet/outlet to 1 and 2
73  pres = np.insert(pres, 0, presDiff[1])
74  pres = np.insert(pres, len(pres), presDiff[0])
75
76  ## Setting up vectors for connected links to the current link.
77  #-> Also creating vectors for the "location"-term
78  linkLocTerm, inletLinks, outletLinks = streamUtils.get_locationTermVector
        ()
79
80  locTerm_inlet = [linkLocTerm[i-1] for i in inletLinks]
81  inletLinks = [x for y, x in sorted(zip(locTerm_inlet, inletLinks))]
82  inletLinks.reverse()
83
84  link_connectedNodes, link_lengths, link_nodeLengths, link_pressure,
        link_nodePressure = streamUtils.ConstrictionLinkPressures(pres,
        g_nodes, g_throats, q_vec)
85
86  # Finding the out and in-links of each node
87  node_out_links = []
88  node_in_links = []
89  avg_out_links = 0.0
90  for i in range(len(node1)):
91    numOfConnections = int(node1[i][4])
92    temp_node_out_links = []
93    temp_node_in_links = []
94    for j in range(numOfConnections):
95      if (pres[int(node1[i][0])] >= pres[int(node1[i][5+j])]):
96        temp_node_out_links.append(int(node1[i][7+numOfConnections+j]))
97      else:
98        temp_node_in_links.append(int(node1[i][7+numOfConnections+j]))
99    node_out_links.append(temp_node_out_links)
100   node_in_links.append(temp_node_in_links)
101   avg_out_links += len(temp_node_out_links)
102 avg_out_links = avg_out_links/numberOfNodes
103
104 nextLink = streamUtils.get_nextLink(link_connectedNodes, linkLocTerm, pres
        , q_vec)
```

```
105
106 angles, out_links = streamUtils.get_LinkAngles(link_connectedNodes,
        link_nodePressure, pres)
107
108 q_vec_in  = np.copy(q_vec) # making copies of q_vec to keep track of in
        and out rates
109 q_vec_out = np.copy(q_vec)
110 q_vec_in_loc  = np.copy(q_vec) # making copies of q_vec to keep track of
        in and out rates
111 q_vec_out_loc = np.copy(q_vec)
112
113 sort_key = args_stream.sort
114
115 for i in range(numberOfNodes):
116   SortVec = []
117   for j in range(len(node_in_links[i])):
118     if  len(node_out_links[i]) > 0:
119       for k in range(len(angles[node_in_links[i][j]-1])):
120         if sort_key == 'angles':
121           SortVec.append([node_in_links[i][j], out_links[node_in_links[i][
                j]-1][k], angles[node_in_links[i][j]-1][k]])
122         elif sort_key == 'location':
123           SortVec.append([node_in_links[i][j], out_links[node_in_links[i][
                j]-1][k], linkLocTerm[out_links[node_in_links[i][j]-1][k
                ]-1]])
124   SortVec = sorted(SortVec, key=itemgetter(2))
125   # Associating volumes to each link
126   for x in range(len(SortVec)):
127     from_link = SortVec[x][0]
128     to_link = SortVec[x][1]
129
130     if q_vec_in[from_link-1]>q_vec_out[to_link-1]:
131       nextLink[from_link-1][2][out_links[from_link-1].index(to_link)]=
              q_vec_out[to_link-1]
132       q_vec_in[from_link-1]-=q_vec_out[to_link-1]
133       q_vec_out[to_link-1]-=q_vec_out[to_link-1]
134     else:
135       nextLink[from_link-1][2][out_links[from_link-1].index(to_link)]=
              q_vec_in[from_link-1]
136       q_vec_out[to_link-1]-=q_vec_in[from_link-1]
137       q_vec_in[from_link-1]-=q_vec_in[from_link-1]
138
139 for i in range(len(nextLink)):
140   k=0
141   while 0 in nextLink[i][2]:
142     if nextLink[i][2][k] == 0:
```

```
143         nextLink[i][0].pop(k)
144         nextLink[i][1].pop(k)
145         nextLink[i][2].pop(k)
146       else:
147         k+=1
148     if ((i+1) in outletLinks):
149       nextLink[i][0].append(0)
150       nextLink[i][1].append(0)
151
152 # Sorting each line in nextLink_ang from high to low flow rates.
153 for i in range(len(nextLink)):
154     nextLink[i][0] = [x for y, x in sorted(zip(nextLink[i][2], nextLink[i
          ][0]), reverse= True)]
155     nextLink[i][2] = sorted(nextLink[i][2], reverse=True)
156
157 toc = time.clock()
158 print "␣␣␣[DONE]␣-␣%.2f␣sec." % (toc-tic)
159
160 totalQ = 0.0
161 constrictionSum = 0.0
162 totalTortVolume=0.0
163 tortSum = 0.0
164
165 listOfStacks = []
166 listOfNodes = []
167 streamlineLength = []
168 length, width, depth = netRecon.get_dimensions()
169
170 tic = time.clock()
171 print "Tracking␣streamlines...␣(from␣%s␣inlet␣links)" % (len(inletLinks))
172
173 for i in tqdm(range(0,len(inletLinks))):
174   # Possible links avaiable
175   while len(nextLink[inletLinks[i]-1][0]) > 0:
176
177     qStack = []
178     stack = []
179     stackLinks = []
180     nextStep = inletLinks[i]
181     stackLinks.append(nextStep)
182
183     while len(nextLink[nextStep-1][0]) > 0 and nextStep not in outletLinks
          : #true
184       if nextLink[nextStep-1][1][0] != 0:
185         stack.append(nextLink[nextStep-1][1][0])
186       stackLinks.append(nextLink[nextStep-1][0][0])
```

```
187         qStack.append(nextLink[nextStep-1][2][0])
188         nextStep = nextLink[nextStep-1][0][0]
189 
190       if nextStep not in outletLinks:
191         stackLinks.pop(-1)
192         while len(stackLinks) > 0 and len(nextLink[nextStep-1][0]) == 0: #
              lengde av vec
193 
194           nextLink[stackLinks[-1]-1][0].pop(0)
195           nextLink[stackLinks[-1]-1][1].pop(0)
196           nextLink[stackLinks[-1]-1][2].pop(0)
197           nextStep = stackLinks[-1]
198           stackLinks.pop(-1)
199 
200       else:
201         smallestQ = np.min(qStack)
202         totalQ += smallestQ
203         listOfStacks.append(stackLinks)
204         listOfNodes.append(stack)
205         pathLength = []
206         orgStack = []
207 
208         pathConst = 0.0
209 
210         for j in range(0,len(stackLinks)-1):
211           link = stackLinks[j]
212           pathLength.append(link1[stackLinks[j]-1][5])
213           nextLink[stackLinks[j]-1][2][0] -= smallestQ
214 
215           if nextLink[stackLinks[j]-1][2][0] <= 0 and nextLink[stackLinks[j
                ]-1][0][0] != 0:
216             nextLink[stackLinks[j]-1][0].pop(0)
217             nextLink[stackLinks[j]-1][1].pop(0)
218             nextLink[stackLinks[j]-1][2].pop(0)
219 
220           if ((link_nodePressure[link-1][0]-link_pressure[link-1][0]) == 0
                or (link_pressure[link-1][0]-link_pressure[link-1][1]) == 0 or
                 (link_pressure[link-1][1]-link_nodePressure[link-1][1]) == 0)
                :
221             print "*"*80
222             print "WARNING:␣Zero-division␣encountered␣in␣calculation."
223             print "Current␣streamline␣(#%i)␣constriction␣factor␣ignored." %
                    i
224             print "*"*80
225           else:
226             pathConst += (link_nodeLengths[link-1][0]**2)/(link_nodePressure
```

```
                [link-1][0]-link_pressure[link-1][0]) + (link_lengths[link
                -1]**2)/(link_pressure[link-1][0]-link_pressure[link-1][1])
                + (link_nodeLengths[link-1][1]**2)/(link_pressure[link
                -1][1]-link_nodePressure[link-1][1])
227
228         pathConst*=1/(sum(pathLength)**2)
229         pathConst*=smallestQ
230         streamlineLength.append(sum(pathLength))
231         tortSum += length/sum(pathLength)
232         constrictionSum+=pathConst
233
234 toc=time.clock()
235 print "Time␣to␣track␣streamlines:␣%.3f␣sec.␣\n" % (toc-tic)
236
237 #effective porosity calculation
238 poro_nodes = []
239 poro_links = []
240
241 for stack in listOfNodes:
242   poro_nodes.extend(stack)
243 poro_nodes = set(poro_nodes)
244 for stack in listOfStacks:
245   poro_links.extend(stack)
246 poro_links = set(poro_links)
247
248 eff_pore_volume = 0
249 for node in poro_nodes:
250   eff_pore_volume += node2[node-1][1]+node2[node-1][4]
251 for link in poro_links:
252   eff_pore_volume += link2[link-1][6]+link2[link-1][7]
253 eff_porosity=(eff_pore_volume/(length*width*depth))*100.0
254
255 if args_stream.phaseConductancePD:
256   eff_phase_volume = 0
257   for node in poro_nodes:
258     eff_phase_volume += (node2[node-1][1]+node2[node-1][4])*
          bodySaturationPD[node-1]
259   for link in poro_links:
260     eff_phase_volume += (link2[link-1][6]+link2[link-1][7])*
          throatSaturationPD[link-1]
261   eff_phase_porosity=(eff_phase_volume/(length*width*depth))*100.0
262
263   phase_volume = 0
264   for i in range(0,len(node2)):
265     phase_volume += (node2[i][1]+node2[i][4])*bodySaturationPD[i]
266   for i in range(0,len(link2)):
```

```
267       phase_volume += (link2[i][6]+link2[i][7])*throatSaturationPD[i]
268     phase_porosity=(phase_volume/(length*width*depth))*100.0
269
270 por_volume = 0
271 for i in range(0,len(node2)):
272     por_volume += (node2[i][1]+node2[i][4])
273 for i in range(0,len(link2)):
274     por_volume += (link2[i][6]+link2[i][7])
275 por_porosity=(por_volume/(length*width*depth))*100.0
276
277 ## Check for unused nodes
278 numUnusedNodes = len(node1)-len(poro_nodes)
279 numUnusedLinks = len(link1)-len(poro_links)
280 numStr_node = (((float(numUnusedNodes)/numberOfNodes)*100), numUnusedNodes
        , numberOfNodes)
281 numStr_link = (((float(numUnusedLinks)/len(link1))*100), numUnusedLinks,
        len(link1))
282
283 mainToc = time.clock()
284 print "* Streamline tracking statistics:"
285 print "Streamlines tracked based on *%s* of links" % sort_key
286 print "Number of streamlines:   %s" % len(listOfStacks)
287 print "Unused nodes:    %.1f%% (%s/%s)" % numStr_node
288 print "Unused links:    %.1f%% (%s/%s)\n" % numStr_link
289 print "* Pore structure measures:"
290 print "Constriction factor:   %.2f" % (constrictionSum/totalQ)
291 print "Tortuosity:    %.4f" % (tortSum/len(listOfStacks))
292 print "Effective porosity:  %.2f %% " % eff_porosity
293 print "Average outlet links:    %.2f \n" % avg_out_links
294 if args_stream.phaseConductancePD:
295     print "Effective phase poro:  %.2f %%" % eff_phase_porosity
296     print "Phase poro:  %.2f %%" % phase_porosity
297     print "Network poro:  %.2f %%" % por_porosity
298     print "Effective saturation:  %.2f %% \n" % (100.0*eff_phase_porosity/
          phase_porosity)
299 print "* Flowrate calculations:"
300 print "Streamline flowrate:   %.7E" % totalQ
301 print "Steady-state flowrate:   %.7E" % inflow_laplace
302 print "- Flowrate difference: %.7E" % (inflow_laplace-totalQ)
303 print "- Error:   %.2f%%      Elapsed time: %.2f sec." % (((inflow_laplace
        -totalQ)/inflow_laplace)*100, (mainToc-mainTic))
304 if args_stream.solvefor == 'FF':
305     FF = laplacePN.calc_formation_factor(p_in, p_out, inflow_laplace)
306     print "Formation factor:  %.2f" % FF
307 else:
308     perm_mdarcy = laplacePN.calc_permeability(p_in, p_out, inflow_laplace)
```

```
309   print "Permeability = %.2f mD" % perm_mdarcy
310
311 if args_stream.saveResultsToFile:
312   outfile=open('output/netStream_output.txt','w')
313   if args_stream.solvefor == 'FF':
314     outfile.write("%.4f\n" % FF)
315   else:
316     outfile.write("%.4f\n" % perm_mdarcy)
317   outfile.write("%.4f\n" % (constrictionSum/totalQ))
318   outfile.write("%.4f\n" % (tortSum/len(listOfStacks)))
319   outfile.write("%.4f\n" % (por_porosity/100.0))
320   outfile.write("%.4f\n" % (eff_porosity/100.0))
321   if args_stream.phaseConductancePD:
322     outfile.write("%.4f\n" % (eff_phase_porosity/100.0))
323     outfile.write("%.4f\n" % (phase_porosity/100.0))
324     outfile.write("%.4f\n" % (eff_phase_porosity/phase_porosity))
325   outfile.close()
326
327 if args_stream.plot:
328   netPlot.plot_streamline(listOfNodes)
```

B.5 streamUtils.py

```
1  import numpy as np
2  import math
3  from numpy import linalg as LA
4  import time
5  import netRecon
6
7  link1 = netRecon.get_network_file('link1')
8  link2 = netRecon.get_network_file('link2')
9  node2 = netRecon.get_network_file('node2')
10 node1 = netRecon.get_network_file('node1')
11
12 def get_locationTermVector():
13   #Calculates the location-term for each link
14   inletLinks = []
15   outletLinks = []
16   locationTerm = [np.sqrt(item[2]**2 + item[3]**2) for item in node1]
17   linkLocTerm = []
18   for i in range(len(link1)):
19     if link1[i][1] != -1 and link1[i][2] != 0:
20       linkLocTerm.append((locationTerm[int(link1[i][1])-1]+locationTerm[
           int(link1[i][2])-1])/2)
21     elif link1[i][1] == -1:
```

```
22        linkLocTerm.append(locationTerm[int(link1[i][2])-1])
23        inletLinks.append(int(link1[i][0]))
24      elif link1[i][2] == 0:
25        linkLocTerm.append(locationTerm[int(link1[i][1])-1])
26        outletLinks.append(int(link1[i][0]))
27    return linkLocTerm, inletLinks, outletLinks
28
29  def ConstrictionLinkPressures(pres, g_nodes, g_throats, q_vec):
30    # Calculating the pressures needed for constriction factor calculation.
31    link_connectedNodes = []
32    link_nodeLengths = []
33    link_nodeConductivity = []
34    link_nodePressure = []
35    link_lengths = []
36    link_conductivity = []
37    link_pressure = []
38
39    for i in range(len(link1)):
40      tempCon = [int(link1[i][1]), int(link1[i][2])]
41      tempPressures = [pres[tempCon[0]], pres[tempCon[1]]]
42      tempLengths = [link2[i][3], link2[i][4]]
43      if -1 in tempCon or 0 in tempCon:   # Finding the pressures between
            each object.
44        g1 = g_nodes[max(tempCon)-1]    # l1     l1    l2
45        g2 = g1                #   g1     gl    g2
46      else:                  #  __         __
47        g1 = g_nodes[tempCon[0]-1]      # /1 \--------/2 \
48        g2 = g_nodes[tempCon[1]-1]      # \__/--------\__/
49      tempConduc = [g1,g2]          #  p1  pa      pb  p2
50
51      link_lengths.append(link2[i][5])
52      link_conductivity.append(g_throats[i])
53
54      link_connectedNodes.append([x for y, x in sorted(zip(tempPressures,
            tempCon))])
55      link_connectedNodes[-1].reverse()
56      link_nodeLengths.append([x for y, x in sorted(zip(tempPressures,
            tempLengths))])
57      link_nodeLengths[-1].reverse()
58      link_nodeConductivity.append([x for y, x in sorted(zip(tempPressures,
            tempConduc))])
59      link_nodeConductivity[-1].reverse()
60      link_nodePressure.append(sorted(tempPressures))
61      link_nodePressure[-1].reverse()
62
63      Q = q_vec[i]
```

```
64      if -1 in tempCon:
65        pb = (Q*link_nodeLengths[-1][1])/link_nodeConductivity[-1][1] +
              link_nodePressure[-1][1]
66        pa = pb+(Q*link_lengths[-1])/link_conductivity[-1]
67      else:
68        pa = link_nodePressure[-1][0]-(Q*link_nodeLengths[-1][0])/
              link_nodeConductivity[-1][0] # NB! q=h*(p1-pa)
69        pb = pa-(Q*link_lengths[-1])/link_conductivity[-1]
70
71      p2check = pb- (Q*link_nodeLengths[-1][1])/link_nodeConductivity[-1][1]
72      link_pressure.append([pa,pb])
73    return link_connectedNodes, link_lengths, link_nodeLengths,
          link_pressure, link_nodePressure
74
75  def get_nextLink(link_connectedNodes, linkLocTerm, pres, q_vec):
76    nextLink = []
77    for i in range(len(link1)):
78      prefNode = link_connectedNodes[i][-1]
79      nextNode_q = q_vec[i]
80      nextLink_temp = []
81      nextNode = []
82      if prefNode > 0: # If the link in flow direction is connected by links
            to other nodes.
83        prefNodeInfo = node1[prefNode-1] # storing the connected nodes info
              in list.
84        numOfConnections = int(prefNodeInfo[4])
85        for j in range(numOfConnections):
86          # Checking if the connected node has lower potential and that it
                is not an inlet node
87          if (pres[int(prefNodeInfo[0])] > pres[int(prefNodeInfo[5+j])] and
                int(prefNodeInfo[5+j]) not in [0,-1]):
88            nextNode.append(int(prefNodeInfo[0]))
89            nextLink_temp.append(int(prefNodeInfo[7+numOfConnections+j]))
90          if int(prefNodeInfo[5+j]) == 0:
91            nextNode.append(int(prefNodeInfo[0]))
92            nextLink_temp.append(int(prefNodeInfo[7+numOfConnections+j]))
93      elif prefNode == 0:
94        nextNode.append(0)
95        nextLink_temp.append(0)
96      nextLink.append([nextLink_temp, nextNode, [0]*len(nextNode)])
97    return nextLink
98
99  def get_LinkAngles(link_connectedNodes, link_nodePressure, pres):
100   # Calculates the angles between inlet link and possible outlet links
101   # Format:   [[list of angles from link 1 to link 1 to N],...
102   #       ,[list of angles from link N to link 1 to N]]]
```

```
103   in_links = []
104   out_links = []
105   in_nodes = []
106   out_nodes = []
107   out_nodes_coord = []
108   angles = []
109   coord_all, coord_in, coord_out, coord_nonsurf = netRecon.get_coord(node1
          )
110   inletPrefCount = 0
111
112   for i in range(len(link1)):
113
114     if -1 in link_connectedNodes[i]:
115       prefNode = link_connectedNodes[i][-1]
116       center_node = prefNode
117       in_vec = coord_all[center_node-1]-np.array([0.01,0,0])-coord_all[
              center_node-1] #coord_all[center_node-1]-np.array([0.01,0,0])-
              coord_all[center_node-1]
118     else:
119       prefNode = link_connectedNodes[i][-1]
120       center_node = prefNode
121       in_node = link_connectedNodes[i][0]
122       in_vec = coord_all[in_node-1]-coord_all[center_node-1]
123     in_link = i+1
124     nextLink_temp = []
125     nextNode = []
126
127     if prefNode > 0: # If the link in flow direction is connected by links
            to other nodes.
128       prefNodeInfo = node1[prefNode-1] # storing the connected nodes info
              in list.
129       numOfConnections = int(prefNodeInfo[4])
130       temp_in_links = []
131       temp_out_links = []
132       temp_in_nodes = []
133       temp_out_nodes = []
134       temp_out_nodes_coord = []
135
136       for j in range(numOfConnections):
137         if (pres[int(prefNodeInfo[0])] > pres[int(prefNodeInfo[5+j])]):
138           temp_out_links.append(int(prefNodeInfo[7+numOfConnections+j]))
139           temp_out_nodes.append(int(prefNodeInfo[5+j]))
140           if int(prefNodeInfo[5+j]) == 0:
141             temp_out_nodes_coord.append(coord_all[center_node-1]+np.array
                    ([0.01,0,0]))
142           else:
```

```
143             temp_out_nodes_coord.append(coord_all[int(prefNodeInfo[5+j])
                    -1])
144       out_links.append(temp_out_links)
145       out_nodes.append(temp_out_nodes)
146       out_nodes_coord.append(temp_out_nodes_coord)
147
148       temp_angles = []
149       in_vec_norm = in_vec/LA.norm(in_vec)
150       for k in range(len(out_links[-1])):
151         out_vec = out_nodes_coord[-1][k]-coord_all[center_node-1]
152         out_vec_norm = out_vec/LA.norm(out_vec)
153         dot_prod = np.dot(in_vec_norm, out_vec_norm)
154         if abs(dot_prod) > 1:
155           print "*"*80
156           print "WARNING: Absolute value of dot product is over 1 and is"
157           print "therefore set to %i. (Dot product: %.16f)" % (int(
                  dot_prod), dot_prod)
158           print "*"*80
159           dot_prod = int(dot_prod)
160         angle = math.acos(dot_prod)*(180/math.pi)
161         temp_angles.append(angle)
162       angles.append(temp_angles)
163
164     elif prefNode == 0:
165       out_nodes.append([])
166       out_links.append([])
167       angles.append([])
168     elif prefNode == -1:
169       inletPrefCount += 1
170       out_nodes.append([])
171       out_links.append([])
172       angles.append([])
173   return angles, out_links
174
175 def print_nextLink_ang(nextLink_ang):
176   for i in range(len(nextLink_ang)):
177     print "Link %s | links: %s | nodes: %s | rates: %s" % ((i+1),
          nextLink_ang[i][0], nextLink_ang[i][1], nextLink_ang[i][2])
```

B.6 netPlot.py

```
1 import numpy as np
2 import time
3 from tqdm import *
4
```

```
5  import matplotlib.pylab as plt
6  import matplotlib.ticker as ticker
7  from mpl_toolkits.mplot3d import Axes3D
8
9  import netRecon
10
11
12 link1 = netRecon.get_network_file('link1')
13 link2 = netRecon.get_network_file('link2')
14 node2 = netRecon.get_network_file('node2')
15 node1 = netRecon.get_network_file('node1')
16
17 def get_streamline_vec(listOfNodes, coord_all):
18   streamline_vec = []
19   for i in range(len(listOfNodes)):
20     x_coord = []
21     z_coord = []
22     for j in range(len(listOfNodes[i])):
23       x_coord.append(coord_all[listOfNodes[i][j]-1][0])
24       z_coord.append(coord_all[listOfNodes[i][j]-1][2])
25     streamline_vec.append([x_coord,z_coord])
26   return streamline_vec
27
28 def plot_streamline(listOfNodes):
29   tic = time.clock()
30   coord_all, coord_in, coord_out, coord_nonsurf = netRecon.get_coord(node1
         )
31   streamline_vec = get_streamline_vec(listOfNodes, coord_all)
32
33   scatter_size = -0.0014*len(coord_all)+15.4
34   fig = plt.figure()
35   plt.scatter(coord_in[:,0], coord_in[:,2], alpha=1,label='Inlet␣pores', s
         =scatter_size, marker="o")
36   plt.scatter(coord_out[:,0], coord_out[:,2], alpha=1, label='Outlet␣pores
         ', s=scatter_size, marker="o")
37   plt.scatter(coord_nonsurf[:,0], coord_nonsurf[:,2], alpha=1, label='Non-
         surface␣pores', s=scatter_size, marker="o")
38
39   print "Plotting␣streamlines..."
40   for i in tqdm(range(len(streamline_vec))):
41     plt.plot(streamline_vec[i][0],streamline_vec[i][1], c='r',alpha=0.1,
           linewidth=0.7)
42
43   plt.legend(scatterpoints=1)
44   fig.tight_layout()
45   axes = plt.gca()
```

```
46    axes.set_xlim([0-max(coord_all[:,0])/10,max(coord_all[:,0])+max(
          coord_all[:,0])/10])
47    axes.set_ylim([0-max(coord_all[:,1])/10,max(coord_all[:,1])+max(
          coord_all[:,1])/10])
48    toc=time.clock()
49    print "Time␣to␣plot␣streamlines:", "%.3f" % (toc-tic)
50    plt.show()
51
52  def plot_network(dim):
53    start_plt = time.time() # Stating the clock
54  # Importing data
55    coord_all, coord_in, coord_out, coord_nonsurf = netRecon.get_coord(node1
          )
56
57    if dim == '3D':
58      fig = plt.figure()
59      ax = fig.add_subplot(111, projection='3d')
60      ax.scatter(coord_in[:,0],coord_in[:,1],coord_in[:,2], c='r', marker='^
            ', label='Inlet␣pores')
61      ax.scatter(coord_out[:,0],coord_out[:,1],coord_out[:,2], c='r', marker
            ='o', label='Outlet␣pores')
62      ax.scatter(coord_nonsurf[:,0],coord_nonsurf[:,1],coord_nonsurf[:,2], c
            ='b', marker='o', label='Non-surface␣pores')
63
64      for i in range(int(link1[-1][0])):
65        if (int(link2[i][1])+int(link2[i][2]))>0:
66          c1=int(link1[i][1])
67          p1=coord_all[c1-1,:]
68          c2=int(link1[i][2])
69          p2=coord_all[c2-1,:]
70          ax.plot([p1[0], p2[0]],[p1[1], p2[1]],[p1[2], p2[2]],c='k',alpha
                =0.4)
71
72      ax.legend(scatterpoints=1)
73      #ax.set_axis_off()
74      time_plt = float(time.time()-start_plt)
75      print "Time␣elapsed␣to␣plot␣network␣=␣", "%.4f" %  time_plt, "sec"
76      plt.show()
77    elif dim == '2D':
78      fig = plt.figure()
79      plt.scatter(coord_in[:,0], coord_in[:,2], alpha=0.5,label='Inlet␣pores
            ')
80      plt.scatter(coord_out[:,0], coord_out[:,2], alpha=0.5, label='Outlet␣
            pores')
81      plt.scatter(coord_nonsurf[:,0], coord_nonsurf[:,2], alpha=0.5, label='
            Non-surface␣pores')
```

```
82
83      for i in range(int(link1[-1][0])):
84        if (int(link2[i][1])>0 and int(link2[i][2]))>0:
85          c1=int(link1[i][1])
86          p1=coord_all[c1-1,:]
87          c2=int(link1[i][2])
88          p2=coord_all[c2-1,:]
89          plt.plot([p1[0], p2[0]],[p1[2], p2[2]], c='k',alpha=0.5)
90
91      plt.legend(scatterpoints=1)
92      fig.tight_layout()
93      axes = plt.gca()
94      axes.set_xlim([0-max(coord_all[:,0])/10,max(coord_all[:,0])+max(
            coord_all[:,0])/10])
95      axes.set_ylim([0-max(coord_all[:,1])/10,max(coord_all[:,1])+max(
            coord_all[:,1])/10])
96      plt.show()
97
98    else:
99        print "Please␣select␣either␣'2D'␣or␣'3D'␣in␣the␣variable␣'dim='"
```

B.7 updateNetwork.py

```
1  ######################################################################
2  #
3  # A code to calculate the solution of the Laplace equation on a
4  # network model by sparse matrix inversion.
5  # Anders Torland, anderstorland@gmail.com, 2018
6  #
7  import numpy as np
8  import scipy
9  import scipy.sparse as sparse
10 import scipy.sparse.linalg
11 import time
12 import math
13
14 # Selfwritten scripts
15 import netRecon
16
17 start = time.clock() # Stating the clock
18 def calc_potential(node1, node2, link1, link2, p_in, p_out, solvefor, **
       saturation):
19   # Calculating conductivity, g, for all pores and throats
20   #Using A=r_inc**2/(4*G) from Mason and Morrow 1990 and Eq. 18 from Oren
         1998
```

```
21  visc = 1   #IF CHANGED: CHANGE IN 'calc_permeability(..)'
22  def calc_g_nodes(node2, argument):
23    g_nodes=[]
24    cum_pore_radius = []
25    for i in range(len(node2)):
26      if argument == 'perm':
27        g_nodes.append((3*node2[i][2]**4)/(80*node2[i][3]*visc))
28        #g_nodes.append((3*(math.pow((math.pi*node2[i,2]**2),2))*node2[i
              ,3])/(5*1))
29      elif argument == 'FF':
30         #Using A=r_inc**2/(4*G) from Mason and Morrow 1990
31        g_nodes.append(node2[i][2]**2/(4*node2[i][3]))
32        if 'bodySaturation' in saturation:
33          g_nodes.append(saturation['bodySaturation'][i]*node2[i
                ][2]**2/(4*node2[i][3]))
34        else:
35          g_nodes.append(node2[i][2]**2/(4*node2[i][3]))
36        #g_nodes.append(math.pi*node2[i,2]**2)
37      cum_pore_radius.append(node2[i][2])
38    return g_nodes, cum_pore_radius
39
40  def calc_g_throats(link1, argument):
41    g_throats=[]
42    cum_throat_radius = []
43    for i in range(len(link1)):
44      if argument == 'perm':
45        g_throats.append((3*link1[i][3]**4)/(80*link1[i][4]*visc))
46        #g_throats.append((3*(math.pow((math.pi*link1[i,3]**2),2))*link1[i
              ,4])/(5*1))
47      elif argument == 'FF':
48        #Using A=r_inc**2/(4*G) from Mason and Morrow 1990
49        if 'throatSaturation' in saturation:
50          g_throats.append(saturation['throatSaturation'][i]*link1[i
                ][3]**2/(4*link1[i][4]))
51        else:
52          g_throats.append(link1[i][3]**2/(4*link1[i][4]))
53        #g_throats.append(math.pi*link1[i,3]**2)
54      cum_throat_radius.append(link1[i][3])
55    return g_throats, cum_throat_radius
56
57  def calc_g_link(link1,link2,g_nodes, g_throats):
58    # Calculating link-conductivity (Oren, 1998)
59    gL_link=[]
60    gL_link_solv=[]
61    denominator=0
62    link_length=[]
```

```
63      g_weigth=1
64      # for-loop: Running through the total number of links.
65      for i in range(len(link1)):
66        if g_throats[i]<=0.0:
67          gL_link.append(0.0)
68          # The link-cond/length was scaled by 1e15. Not needed anymore.
69          gL_link_solv.append(0.0)
70        else:
71          # if: Checking if link is flagged as "inlet".
72          if int(link2[i][1])==-1:
73            if g_nodes[int(link2[i][2])-1]<=0.0:
74              gL_link.append(0.0)
75              gL_link_solv.append(0.0)
76            else:
77              denominator=(((link2[i][5]+link2[i][3])/g_throats[i])+g_weigth
                    *(link2[i][4]/g_nodes[int(link2[i][2])-1]))
78              gL_link.append(1/denominator)
79              gL_link_solv.append(1/denominator)
80          # if: Checking if link is flagged as "outlet".
81          elif int(link2[i][2])==0:
82            if g_nodes[int(link2[i][1])-1]<=0.0:
83              gL_link.append(0.0)
84              gL_link_solv.append(0.0)
85            else:
86              denominator=(((link2[i][5]+link2[i][4])/g_throats[i])+g_weigth
                    *(link2[i][3]/g_nodes[int(link2[i][1])-1]))
87              gL_link.append(1/denominator)
88              gL_link_solv.append(1/denominator)
89          else:
90            if g_nodes[int(link2[i][1])-1]<=0.0 or g_nodes[int(link2[i][2])
                  -1] <= 0.0:
91              gL_link.append(0.0)
92              gL_link_solv.append(0.0)
93            else:
94              denominator=((link2[i][5]/g_throats[i])+g_weigth*((link2[i
                    ][3]/g_nodes[int(link2[i][1])-1])+(link2[i][4]/g_nodes[int
                    (link2[i][2])-1])))
95              gL_link.append(1/denominator)
96              gL_link_solv.append(1/denominator)
97          link_length.append(link1[i][5])
98      return gL_link_solv, gL_link, link_length
99
100   def create_equation_matrix(node1, gL_link, gL_link_solv, p_in, p_out):
101     num_node=int(node1[-1][0])
102     # Defining different indexing vectors.
103     row=[]
```

```
104     col=[]
105     data=[]
106     b_vec=[]
107     g_inlet=[]
108     g_outlet=[]
109     in_nodes=[]
110     in_link=[]
111     out_nodes=[]
112     out_link=[]
113
114     # for-loop: Running through the total number of nodes.
115     for i in range(num_node):
116
117       num_con=int(node1[i][4])
118       # Setting up b-vector.
119       # if: True if inlet.
120       if int(node1[i][5+num_con]) == 1:
121         ind=node1[i].index(-1)
122         b_vec.append(-p_in*gL_link_solv[int(node1[i][ind+num_con+2])-1])
123         row.append(i)
124         col.append(i)
125         data.append(-gL_link_solv[int(node1[i][ind+num_con+2])-1])
126
127         ## For rate calculations over inlet
128         g_inlet.append(gL_link[int(node1[i][ind+num_con+2])-1])
129         in_nodes.append(i+1)
130         in_link.append(int(node1[i][ind+num_con+2]))
131       # if: True if outlet
132       elif int(node1[i][6+num_con]) == 1:
133         ind=node1[i].index(0)
134         b_vec.append(-p_out*gL_link_solv[int(node1[i][ind+num_con+2])-1])
135         row.append(i)
136         col.append(i)
137         data.append(-gL_link_solv[int(node1[i][ind+num_con+2])-1])
138
139         ## For rate calculations over outlet
140         g_outlet.append(gL_link[int(node1[i][ind+num_con+2])-1])
141         out_nodes.append(i+1)
142         out_link.append(int(node1[i][ind+num_con+2]))
143
144       else:
145         b_vec.append(0)
146       # for-loop: Running "connection number"-times through each line,
147       # finding all connected nodes to current node, and setting up p-
              matrix.
148       for j in range(num_con):
```

```
149         # if: True if connection number for current node > 0
150         # AND connected node is NOT inlet/outlet.
151
152         if int(node1[i][5+j])>0 and int(node1[i][4])>0:
153           # For the connecting nodes
154           row.append(i)
155           col.append(int(node1[i][5+j])-1)
156           data.append(gL_link_solv[int(node1[i][7+num_con+j])-1])
157           # the node itself
158           row.append(i)
159           col.append(i)
160           data.append(-gL_link_solv[int(node1[i][7+num_con+j])-1])
161
162     # Constructing the sparse matrix from col,row and data
163     mtx = sparse.csc_matrix((data, (row, col)), shape=(num_node,num_node))
164     b_v=np.asarray(b_vec)
165     return mtx, b_v, g_inlet, g_outlet, in_nodes, out_nodes, in_link,
          out_link
166
167   def create_initial_guess(node1,guess, p_in, p_out):
168     num_node=int(node1[-1][0])
169     def linear_initial_guess(num_node,p_in,p_out):
170       length, width, depth = netRecon.get_dimensions()
171       p_L=(p_in-p_out)/length
172       for i in range(num_node):
173         x0vec[i]=p_in-node1[i][1]*p_L
174       return x0vec
175
176     x0vec = np.zeros(num_node,dtype=np.float)
177     if guess == 'linear':
178       x0vec = linear_initial_guess(num_node,p_in,p_out)
179
180   def CG_potential_solver(mtx, b_v, initial_guess, tolerance):
181     #Solving by CG-method.
182     tic=time.clock()
183     # Solving for pressures with CG.
184     xvec = scipy.sparse.linalg.cg(mtx, b_v, initial_guess, tol=tolerance)
185     toc=time.clock()
186     CG_runtime=toc-tic
187     return xvec[0]
188
189   g_nodes, cum_pore_radius = calc_g_nodes(node2, solvefor)
190   g_throats, cum_throat_radius = calc_g_throats(link1, solvefor)
191   gL_link_solv, gL_link, link_length = calc_g_link(link1,link2,g_nodes,
        g_throats)
192   mtx, b_v, g_inlet, g_outlet, in_nodes, out_nodes, in_link, out_link =
```

```
        create_equation_matrix(node1, gL_link, gL_link_solv, p_in, p_out)
193   x0vec = create_initial_guess(node1,'linear', p_in, p_out)
194   xvec = CG_potential_solver(mtx, b_v, x0vec, 1E-15)
195
196   return xvec, g_nodes, g_throats, gL_link, g_inlet, g_outlet, in_nodes,
          out_nodes, cum_pore_radius, cum_throat_radius, link_length, in_link,
           out_link
197
198 def calc_surface_flowrates(g_inlet, g_outlet, in_nodes, out_nodes, p_in,
      p_out, xvec):
199 # Calculating flowrate at inlet and outlet
200   inflow=0
201   outflow=0
202   inlet_counter=0
203   outlet_counter=0
204   # for-loop: Running through all inlet pores.
205   # Calculating flowrate at current inlet-pore.
206   for i in range(len(in_nodes)):
207     inflow+=(g_inlet[i]*(p_in-xvec[int(in_nodes[i])-1]))
208     inlet_counter+=1
209     # for-loop: Running through all outlet pores.
210     # Calculating flowrate at current outlet-pore.
211   for i in range(len(out_nodes)):
212     outflow+=(g_outlet[i]*(p_out-xvec[int(out_nodes[i])-1]))
213     outlet_counter+=1
214   return inflow, outflow, inlet_counter, outlet_counter
215
216 def calc_permeability(p_in, p_out, inflow):
217   # Permeability calculation
218   length, width, depth = netRecon.get_dimensions()
219
220   dP=p_in-p_out           # Pressure difference over inlet.
221   area=width*depth          # Inlet area:     total_y_length *
          total_z_length
222   visc=1              # Viscosity:    Defined as 1 throughout.
223   perm=(inflow*length*visc)/(area*dP) # Permeability: By Darcys Law (in SI
          -unit: m^2)
224   perm_mdarcy=perm*1.01*10**15        # Permeability: In Darcy-unit: mD
          (1.01*12^-12 m^2 = 1 mD)
225
226   return perm_mdarcy
227
228 def calc_formation_factor(p_in, p_out, inflow):
229   length, width, depth = netRecon.get_dimensions()
230   area=width*depth
231   FF=((area*(p_in-p_out))/(inflow*length))
```

```
232   return FF
233
234 def calc_network_statistics(node1,node2,link1,link2):
235   length, width, depth = netRecon.get_dimensions()
236   area=width*depth
237   num_node=int(node1[-1][0])
238   # Porosity calculation and link/node statistics
239   pore_volume=0
240   pore_radius=[]
241   link_radius=0
242   total_connections=0
243   # for-loop: Running through all links.
244   for i in range(len(link2)):
245     pore_volume+=link2[i][6]+link2[i][7]
246     link_radius+=link1[i][3]
247   # for-loop: Running through all pores.
248   for i in range(len(node2)):
249     pore_volume+=node2[i][1]+node2[i][4]
250     pore_radius.append(node2[i][2])
251     total_connections+=node1[i][4]
252   avg_pore_radius=(sum(pore_radius)/num_node)*10**6
253   max_pore_radius=max(pore_radius)*10**6
254   min_pore_radius=min(pore_radius)*10**6
255   avg_link_radius=(link_radius/int(link1[-1][0]))*10**6
256   avg_con_num=total_connections/num_node
257   porosity=(pore_volume*100)/(length*area)
258   return avg_pore_radius, max_pore_radius, min_pore_radius,
        avg_link_radius, avg_con_num, porosity
259
260 def calc_flowrate(link1, gL_link, p_in, p_out, xvec):
261   # Set up q in each link
262   q_vec=[]
263   # for-loop: Running through all inlet pores.
264   for i in range(int(link1[-1][0])):
265     c1=int(link1[i][1])
266     c2=int(link1[i][2])
267     if c1==-1:
268       q_vec.append(gL_link[i]*(p_in-xvec[c2-1]))
269     elif c2==0:
270       q_vec.append(gL_link[i]*(xvec[c1-1]-p_out))
271     else:
272       q_vec.append(gL_link[i]*(xvec[c2-1]-xvec[c1-1]))
273   q_vec=np.asarray(q_vec)
274   q_vec=abs(q_vec)
275
276   return q_vec
```

B.8 netRecon.py

```
1  ######################################################################
2  #
3  # A script to import, export and modify network data files.
4  # This script contains several functions, such as: removal of non-spanning
       nodes and links, rearrangement
5  # of the network to the standard format, reading and writing data files,
      calculation of different pore
6  # structure measures.
7  #
8  # Anders Torland, anderstorland@gmail.com, 2018
9  #
10 import shutil
11 import os
12 import numpy as np
13 import linecache
14 import time
15 from tqdm import *
16 from operator import itemgetter
17
18 def get_network_file(file):
19   filename = file+'.dat'
20   if 'link' in file:
21     if 'original' in filename:
22       numberOfLines = int(linecache.getline('originalNetworks/link1.dat',
             1))
23     else:
24       numberOfLines = int(linecache.getline('link1.dat', 1))
25   elif 'node' in file:
26     if 'original' in filename:
27       numberOfLines = int(linecache.getline('originalNetworks/node1.dat',
             1).split()[0])
28     else:
29       numberOfLines = int(linecache.getline('node1.dat', 1).split()[0])
30   else:
31     print "File type not recognized!"
32
33   if '1' in file:
34     skip = 1
35   else:
36     skip = 0
37
38   outfile = []
39   for i in range(numberOfLines):
40     line=linecache.getline(filename, i+1+skip).split()
```

```
41      numLine = [float(i) for i in line]
42      outfile.append(numLine)
43    return outfile
44
45  def get_saturation_file(networkElements,floodingCycle):
46    filename = networkElements+'sat_'+floodingCycle+'.dat'
47    if 'link' in networkElements:
48      values = np.loadtxt(filename)
49      returnVector=values[:,3]
50    elif 'node' in networkElements:
51      values = np.loadtxt(filename, skiprows=1)
52      returnVector=values[:,3]
53    return returnVector
54
55  def get_surface_nodes(node1):
56    numberOfNodes = len(node1)
57    #print "length:", numberOfNodes
58    inletNodes = []
59    outletNodes = []
60    for i in range(numberOfNodes):
61      #print "line:", node1[i][4]
62      numberOfConnections = int(node1[i][4])
63      if node1[i][5+numberOfConnections] == 1:
64        inletNodes.append(int(node1[i][0]))
65      elif node1[i][6+numberOfConnections] == 1:
66        outletNodes.append(int(node1[i][0]))
67    return inletNodes, outletNodes
68
69  def get_dimensions():
70    line1=linecache.getline('node1.dat', 1).split()
71    length = float(line1[1])
72    width = float(line1[2])
73    depth = float(line1[3])
74    return length, width, depth
75
76  def get_zero_coord_nodes(node1):
77    zero_coord_nodes = 0
78    for line in node1:
79      if line[4] == 0:
80        zero_coord_nodes +=1
81    return zero_coord_nodes
82
83  def map_connections(fromNodes, node1):
84    def append_connected_nodes(ap_node):
85      connectionNumber = int(node1[ap_node-1][4])
86      connectedNodes = node1[ap_node-1][5:5+connectionNumber]
```

```
87
88      for i in range(len(connectedNodes)):
89        connectedNodes[i] = int(connectedNodes[i])
90      connectedNodes = sorted(connectedNodes, reverse=True)
91      for i in connectedNodes:
92        if i > 0 and i not in checkNodes and i not in usedNodes:
93          checkNodes.append(i)
94          usedNodes.append(i)
95    conMap = [0]*len(node1)
96    checkNodes = []
97    usedNodes = []
98    for node in fromNodes:
99      checkNodes.append(node)
100     usedNodes.append(node)
101   while len(checkNodes)>0:
102     append_connected_nodes(checkNodes[0])
103     conMap[checkNodes[0]-1] = 1
104     checkNodes.remove(checkNodes[0])
105   return conMap
106
107 def delete_unreachable_nodes(nodesToRemove, node1, node2):
108   for node in nodesToRemove:
109     node1.pop(node-1)
110     node2.pop(node-1)
111   numNodesRemoved = len(nodesToRemove)
112   return numNodesRemoved, node1, node2
113
114 def delete_unreachable_links(nodesToRemove, link1, link2):
115   linksToRemove = []
116   for i in range(len(link1)):
117     conNode1 = link1[i][1]
118     conNode2 = link1[i][2]
119     if conNode1 in nodesToRemove or conNode2 in nodesToRemove:
120       linksToRemove.append(i+1)
121   linksToRemove = sorted(linksToRemove, reverse=True)
122   numLinksRemoved = len(linksToRemove)
123
124   for link in linksToRemove:
125     link1.pop(link-1)
126     link2.pop(link-1)
127   return numLinksRemoved,link1, link2
128
129 def get_unreachable_nodes(inMap, outMap, node1):
130   combMap = [0]*len(node1)
131   unreachableNodes = []
132   unreachInlet = 0
```

```
133   unreachOutlet = 0
134   unreachBoth = 0
135   for i in range(len(combMap)):
136     if inMap[i] == 1 and outMap[i] == 1:
137       combMap[i] = 1
138     else:
139       unreachableNodes.append(i+1)
140
141     if inMap[i] == 0:
142       unreachInlet += 1
143     elif outMap[i] == 0:
144       unreachOutlet += 1
145     if inMap[i] == 0 and outMap[i] == 0:
146       unreachBoth += 1
147   unreachableNodes = sorted(unreachableNodes, reverse=True)
148   return unreachableNodes, unreachInlet, unreachOutlet, unreachBoth
149
150 def remove_nonSpanning(node1,node2,link1,link2):
151   print "Removing non-spanning pores and throats..."
152   inletNodes, outletNodes = get_surface_nodes(node1)
153   zero_coord_nodes = get_zero_coord_nodes(node1)
154   inletConMap = map_connections(inletNodes, node1)
155   outletConMap = map_connections(outletNodes, node1)
156   nodesToRemove, unreachInlet, unreachOutlet, unreachBoth =
          get_unreachable_nodes(inletConMap,outletConMap, node1)
157   numNodesRemoved, node1, node2 = delete_unreachable_nodes(nodesToRemove,
          node1, node2)
158   numLinksRemoved, link1, link2 = delete_unreachable_links(nodesToRemove,
          link1, link2)
159
160   # Some network reconstruction statistics.
161   print "%s nodes are not reached from inlet." % unreachInlet
162   print "%s nodes are not reached from inlet." % unreachOutlet
163   print "%s nodes are not connected through network." % numNodesRemoved
164   print "%s links are not connected through network." % numLinksRemoved
165   print "%s nodes connected to either inlet or outlet." % unreachBoth
166   print "%s nodes have zero coordination number." % zero_coord_nodes
167
168   return node1, node2, link1, link2, numNodesRemoved, numLinksRemoved,
          unreachInlet, unreachOutlet, nodesToRemove, unreachBoth
169
170 def print_network_to_file(network_file, name):
171   if os.path.isfile(name):
172     move_network_files(name)
173   orgFilePath = 'originalNetworks/'+name
174   firstLine = linecache.getline(orgFilePath,1).split()
```

```
175   if int(firstLine[0]) != 1:
176     firstLine[0] = len(network_file)
177     firstLine = str(firstLine).strip('[]')
178     firstLine = firstLine.replace(",", "")
179     firstLine = firstLine.replace("'", "")
180     file = open(name, 'w')
181     file.write(firstLine+'\n')
182   else:
183     file = open(name, 'w')
184
185   for item in network_file:
186     string = str(item).strip('[]')
187     string = string.replace(",", "")
188       file.write(string+'\n' % item)
189     file.close()
190     print "%s created!" % name
191
192 def move_network_files(name):
193   source = os.getcwd()
194   dest = os.getcwd()+'/originalNetworks'
195   if not os.path.exists(dest):
196     os.makedirs(dest)
197   files = os.listdir(source)
198   for f in files:
199     if f == name:
200       if os.path.isfile(dest+'/'+name):
201         dest += '/duplicate'
202         if not os.path.exists(dest):
203           os.makedirs(dest)
204       shutil.move(f, dest)
205
206 def rearrange_network(node1, node2, link1, link2):
207   def change_connected_nodes(new, old, oldConnectedNodes,
        oldConnectedLinks):
208     for i in range(len(node1)):
209       if node1[i][0] in oldConnectedNodes:
210         connectionNumber = int(node1[i][4])
211         connectedNodes = node1[i][5:5+connectionNumber]
212         nodeRange = np.arange(5,5+connectionNumber)
213         if old in connectedNodes:
214           for j in nodeRange:
215             if node1[i][j] == old:
216               node1[i][j] = new
217
218     for i in range(len(link1)):
219       if link1[i][0] in oldConnectedLinks:
```

```
220         connections = [int(link1[i][1]), int(link1[i][2])]
221         nodeRange = np.arange(1,3)
222         if old in connections:
223           for j in nodeRange:
224             if link1[i][j] == old:
225               link1[i][j] = new
226               link2[i][j] = new
227
228   def change_connected_links(new, old, connections):
229     for i in range(len(node1)):
230       if node1[i][0] in connections:
231         connectionNumber = int(node1[i][4])
232         connectedLinks = node1[i][7+connectionNumber:7+2*connectionNumber]
233         linkRange = np.arange(7+connectionNumber,7+2*connectionNumber)
234         if old in connectedLinks:
235           for j in linkRange:
236             if node1[i][j] == old:
237               node1[i][j] = new
238
239   print "Rearranging␣the␣network..."
240   # Changeing node numbers in both node and link file
241   print "Rearranging␣nodes..."
242   for i in tqdm(range(len(node1))):
243     if node1[i][0] != (i+1):
244       newNumber = i+1
245       oldNumber = node1[i][0]
246       connectionNumber = int(node1[i][4])
247       oldConnectedNodes = node1[i][5:5+connectionNumber]
248       oldConnectedLinks = node1[i][7+connectionNumber:7+2*connectionNumber
              ]
249       node1[i][0] = newNumber
250       node2[i][0] = newNumber
251       change_connected_nodes(newNumber,oldNumber, oldConnectedNodes,
              oldConnectedLinks)
252
253   print "Rearranging␣links..."
254   for i in tqdm(range(len(link1))):
255     if link1[i][0] != (i+1):
256       newNumber = i+1
257       oldNumber = link1[i][0]
258       connections = [link1[i][1],link1[i][2]]
259       link1[i][0] = newNumber
260       link2[i][0] = newNumber
261       change_connected_links(newNumber,oldNumber, connections)
262   return node1, node2, link1, link2
263
```

```
264 def get_coordination_number(node1):
265   total_connections = 0
266   for i in range(len(node1)):
267     total_connections += node1[i][4]
268   avg_con_num = 1.0*total_connections/len(node1)
269   return avg_con_num
270 
271 def get_redundant_links(link1):
272   paths = []
273   numRedundantPaths = 0
274   for line in link1:
275     path = [line[1],line[2]]
276     path = sorted(path)
277     if path in paths:
278       numRedundantPaths += 1
279     paths.append(path)
280   return numRedundantPaths
281 
282 def get_local_aspect_ratio(): #needs original networkfiles
283   def print_sor_aspect_to_file(sor,aspect):
284     file = open('output/sor_aspect_min.txt', 'w')
285     for i in range(len(sor)):
286       string = str(sor[i])+'\t'+str(aspect[i])
287         file.write(string+'\n')
288       file.close()
289   nodesat_wf = np.loadtxt('nodesat_wf.dat', skiprows=2)
290 
291   dest = os.getcwd()+'/originalNetworks'
292   if os.path.exists(dest):
293     path = 'originalNetworks/'
294     link1 = get_network_file(path+'link1')
295     link2 = get_network_file(path+'link2')
296     node2 = get_network_file(path+'node2')
297     node1 = get_network_file(path+'node1')
298     print "Found original network data files."
299   else:
300     print "Plese locate original network data files."
301     sys.exit()
302 
303   if len(nodesat_wf) != len(node1):
304     print "ERROR: The length of the node1.dat and nodesat_wf.dat is not
          equal!"
305     print "Maybe updated network files are used."
306 
307   numberOfNodes = len(nodesat_wf)
308   temp_min_aspect_sor = []
```

```
309   min_loc_aspect_ws = [] # The minimum local aspect ratio in the non water
          saturated nodes.
310   loc_sor_ws = []
311   loc_aspect_avg = []
312   loc_sor_all = []
313   loc_avg_aspect = []
314   loc_aspect_min = []
315
316   # for-loop: Running through the total number of nodes.
317   for i in range(numberOfNodes):
318     num_con=int(node1[i][4])
319     loc_aspect_current_node= []
320     for j in range(num_con):
321       # If the node is not in/outlet and coordination number > 0
322       if int(node1[i][5+j])>0 and int(node1[i][4])>0:
323         loc_aspect_current_node.append(node2[i][2]/link1[int(node1[i][7+
              num_con+j])-1][3])
324
325     if len(loc_aspect_current_node)>0:
326       # Store average and minimum aspect for every node
327       loc_aspect_avg.append(np.average(loc_aspect_current_node))
328       loc_aspect_min.append(np.min(loc_aspect_current_node))
329       loc_sor_all.append(1-nodesat_wf[i][6])
330
331       if nodesat_wf[i][6] < 1 :
332         temp_min_aspect_sor.append([loc_aspect_min[-1], 1-nodesat_wf[i
              ][6]])
333
334       else: # fully watersaturated nodes
335         min_loc_aspect_ws.append(loc_aspect_min[-1])
336         loc_sor_ws.append(1-nodesat_wf[i][6])
337
338   min_loc_aspect = [] # The average local aspect ratio in the non water
          saturated nodes.
339   loc_sor = [] # The Sor in the non water saturated nodes.
340
341   temp_min_aspect_sor = sorted(temp_min_aspect_sor, key=itemgetter(1))
342   for i in range(len(temp_min_aspect_sor)):
343     min_loc_aspect.append(temp_min_aspect_sor[i][0])
344     loc_sor.append(temp_min_aspect_sor[i][1])
345
346   # The average of the local minimum aspect ratio
347   avg_all_min_loc_aspect = np.average(loc_aspect_min)
348   avg_all_loc_aspect = np.average(loc_aspect_avg)
349   print_sor_aspect_to_file(loc_sor,min_loc_aspect)
350   return avg_all_min_loc_aspect, avg_all_loc_aspect
```

```
351
352 def output_network_mod_file(numNodesRemoved, numLinksRemoved, unreachInlet
        , unreachOutlet, unreachableNodes, unreachBoth, zero_coord_nodes):
353   foldername = os.path.basename(os.getcwd())
354   file = open(foldername + '_netRecon.txt', 'w')
355   file.write('Network␣reconstruction␣data␣for␣network:␣'+foldername+'\n\n'
          )
356   file.write('%s␣nodes␣not␣reached␣from␣inlet.␣␣\n' % unreachInlet)
357   file.write('%s␣nodes␣not␣reached␣from␣outlet.␣\n' % unreachOutlet)
358   file.write('%s␣nodes␣not␣reached␣from␣both␣inlet␣and␣outlet.␣\n' %
          unreachBoth)
359   file.write('%s␣nodes␣with␣zero␣coordination␣number.␣\n' %
          zero_coord_nodes)
360   file.write('%s␣nodes␣removed.␣\n' % numNodesRemoved)
361   file.write('%s␣links␣removed.␣\n␣\n' % numLinksRemoved)
362   file.write('Index␣of␣nodes␣removed:␣\n')
363   for node in unreachableNodes:
364       file.write(str(node)+'\n')
365     file.close()
366
367 def print_network_statistics(node1, node2, link1, link2, network):
368   inletNodes, outletNodes = get_surface_nodes(node1)
369   numInletNodes = len(inletNodes)
370   numOutletNodes = len(outletNodes)
371   length, width, depth = get_dimensions()
372   area=width*depth
373   num_node=int(node1[-1][0])
374   # Porosity calculation and link/node statistics
375   pore_volume = 0.0
376   link_volume = 0.0
377   pore_radius = []
378   link_radius = []
379   pore_radius_vol = []
380   link_radius_vol = []
381   total_connections = 0
382   # for-loop: Running through all links.
383   for i in range(len(link2)):
384     link_volume += link2[i][6]+link2[i][7]
385     link_radius.append(link1[i][3])
386     if link2[i][6] > 0:
387       link_radius_vol.append(link1[i][3]*link2[i][6])
388
389   # for-loop: Running through all pores.
390   for i in range(len(node2)):
391     pore_volume += node2[i][1]+node2[i][4]
392     pore_radius.append(node2[i][2])
```

```
393       pore_radius_vol.append(node2[i][2]*node2[i][1])
394       total_connections += node1[i][4]
395
396     avg_pore_radius = np.average(pore_radius)*10**6
397     avg_link_radius = np.average(link_radius)*10**6
398     glob_aspect_vol = ((np.sum(pore_radius_vol)/pore_volume)/(np.sum(
            link_radius_vol)/link_volume))
399     avg_all_min_loc_aspect, avg_all_loc_aspect = get_local_aspect_ratio()
400     max_pore_radius = max(pore_radius)*10**6
401     min_pore_radius = min(pore_radius)*10**6
402     max_link_radius = max(link_radius)*10**6
403     min_link_radius = min(link_radius)*10**6
404     avg_con_num = total_connections/num_node
405     porosity = ((pore_volume+link_volume)/(length*area))*100.0
406
407     if network == 'updated':
408       porosityString = "Total␣porosity␣(wo/non-span.):␣␣%.2f" % porosity + "
              ␣%"
409       networkString = "\tThe␣network␣data␣files␣are␣updated:␣Non-spanning␣(
              non-span)␣nodes␣\n␣\tand␣links␣are␣removed."
410       conNumString = "(NB!␣Non-span.␣removed.)"
411     else:
412       porosityString = "Total␣porosity:␣␣␣␣%.2f" % porosity
413       networkString = "The␣network␣data␣files␣are␣original."
414       conNumString = ""
415
416     print "-+-+-␣Network␣statistics", "-+"*27
417     print "INFO:", networkString
418     print "␣"
419     print "Number␣of␣nodes:␣␣␣%s␣␣|␣Inlet␣nodes:␣%s,␣Outlet␣nodes:␣%s" % (
            len(node1), numInletNodes, numOutletNodes)
420
421     print "Number␣of␣links:␣␣␣%s" % (len(link1))
422     print "␣"
423     print "Global␣aspect␣ratio:␣␣␣␣%.4f␣␣␣␣(average(p_r)/average(l_r))" % (
            avg_pore_radius/avg_link_radius)
424     print "Global␣aspect␣ratio␣(vol):␣␣␣%.4f␣" % (glob_aspect_vol)
425     print "Avg.␣local␣aspect␣ratio:␣␣␣%.4f␣␣␣␣(p_r/average(bounding(l_r))" %
             avg_all_loc_aspect
426     print "Local␣(minimum)␣aspect␣ratio:␣␣%.4f␣␣␣␣(p_r/maximum(bounding(l_r)
            )" % avg_all_min_loc_aspect
427     print "Average␣connection␣number:␣␣␣%.4f␣␣␣␣" % avg_con_num,
            conNumString
428     print ""
429     print porosityString
430     print "␣"
```

```
431   print "* Node metrics"
432   print "- Average node radius:     %.2f  micro-meter" % avg_pore_radius
433   print "- Maximum node radius:     %.2f  micro-meter" % max_pore_radius
434   print "- Minimum node radius:     %.2f  micro-meter" % min_pore_radius
435   print "* Link metrics"
436   print "- Average link radius:     %.2f  micro-meter" % avg_link_radius
437   print "- Maximum link radius:     %.2f  micro-meter" % max_link_radius
438   print "- Minimum link radius:     %.2f  micro-meter" % min_link_radius
439
440 def check_location_of_surface_nodes(node1):
441   inletNodes, outletNodes = get_surface_nodes(node1)
442   in_tot_x = 0
443   in_tot_y = 0
444   in_tot_z = 0
445   out_tot_x = 0
446   out_tot_y = 0
447   out_tot_z = 0
448
449   for node in inletNodes:
450     in_tot_x += node1[node-1][1]
451     in_tot_y += node1[node-1][2]
452     in_tot_z += node1[node-1][3]
453
454   for node in outletNodes:
455     out_tot_x += node1[node-1][1]
456     out_tot_y += node1[node-1][2]
457     out_tot_z += node1[node-1][3]
458   print "Inlet nodes:"
459   print "Total x-dist:   %.2f" % in_tot_x
460   print "Total y-dist:   %.2f" % in_tot_y
461   print "Total z-dist:   %.2f" % in_tot_z
462   print ""
463   print "Outlet nodes:\n"
464   print "Total x-dist:   %.2f" % out_tot_x
465   print "Total y-dist:   %.2f" % out_tot_y
466   print "Total z-dist:   %.2f" % out_tot_z
467
468 def get_coord(node1):
469   num_node=int(node1[-1][0])
470   coord_all=np.zeros((num_node, 3))
471   coord_in=[]
472   coord_out=[]
473   coord_nonsurf=[]
474   for i in range(num_node):
475     temp=node1[i]
476     num_con=int(temp[4])
```

```
477       coord_all[i,:]=([temp[1], temp[2], temp[3]])
478       if int(temp[5+num_con]) == 1:
479         coord_in.append(([temp[1], temp[2], temp[3]]))
480       elif int(temp[6+num_con]) == 1:
481         coord_out.append(([temp[1], temp[2], temp[3]]))
482       else:
483         coord_nonsurf.append([temp[1], temp[2], temp[3]])
484     coord_in=np.asarray(coord_in)
485     coord_out=np.asarray(coord_out)
486     coord_nonsurf=np.asarray(coord_nonsurf)
487     return coord_all, coord_in, coord_out, coord_nonsurf
488
489 def bin_data(x_data, y_data, binType, bin_size):
490     # Binning parameters:
491     #bin_interval = 0.05    # For EQUAL INTERVAL binning
492     #bin_count = 100    # For EQUAL COUNT binning
493     if binType == 'EI':
494       # Equal interval bin
495       bin_ticker_EI = bin_size
496       temp_bin_x = []
497       temp_bin_y = []
498       binned_x = []
499       binned_y = []
500       data_points_in_bin = []
501
502       for i in range(len(y_data)):
503         if x_data[i] <= bin_ticker_EI:
504           temp_bin_x.append(x_data[i])
505           temp_bin_y.append(y_data[i])
506         else:
507           if len(temp_bin_x) > 0:
508             binned_x.append(np.average(temp_bin_x))
509             binned_y.append(np.average(temp_bin_y))
510             data_points_in_bin.append(len(temp_bin_x))
511             temp_bin_x = []
512             temp_bin_y = []
513             temp_bin_x.append(x_data[i])
514             temp_bin_y.append(y_data[i])
515           bin_ticker_EI += bin_size
516
517       binned_x.append(np.average(temp_bin_x))
518       binned_y.append(np.average(temp_bin_y))
519       data_points_in_bin.append(len(temp_bin_x))
520
521     if binType == 'EC':
522       # Equal count bin
```

```
    bin_count = bin_size
    bin_ticker_EC = 0
    temp_bin_x = []
    temp_bin_y = []
    binned_x = []
    binned_y = []
    data_points_in_bin= []

    for i in range(len(aspect)):
      if bin_ticker_EC < bin_count:
        temp_bin_x.append(sor[i])
        temp_bin_y.append(aspect[i])
        bin_ticker_EC += 1
      else:
        binned_x.append(np.average(temp_bin_x))
        binned_y.append(np.average(temp_bin_y))
        data_points_in_bin.append(len(temp_bin_x))
        temp_bin_x = []
        temp_bin_y = []
        temp_bin_x.append(sor[i])
        temp_bin_y.append(aspect[i])
        bin_ticker_EC = 1

    binned_x.append(np.average(temp_bin_x))
    binned_y.append(np.average(temp_bin_y))
    data_points_in_bin.append(len(temp_bin_x))
  return binned_x, binned_y, data_points_in_bin
```

B.9 percolation.py

```
import numpy as np
import math as m
import os
import sys
import time
from operator import itemgetter
from tqdm import *

import netRecon

dest = os.getcwd()+'/originalNetworks'
if os.path.exists(dest):
  path = 'originalNetworks/'
  link1 = netRecon.get_network_file(path+'link1')
  link2 = netRecon.get_network_file(path+'link2')
```

```
16    node2 = netRecon.get_network_file(path+'node2')
17    node1 = netRecon.get_network_file(path+'node1')
18    print "Found original network data files."
19  else:
20    print "Plese locate original network data files."
21    sys.exit()
22
23  avg_con_num = netRecon.get_coordination_number(node1)
24
25  def sort_links_by_radius(link1):
26    linkRadi = []
27    for i in range(len(link1)):
28      linkRadi.append([int(link1[i][0]), link1[i][3]])
29    linkRadi = sorted(linkRadi, key=itemgetter(1))
30    return linkRadi
31
32  def remove_link(removeLink, node1, inletNodes, outletNodes):
33    conNodes = [int(link1[removeLink-1][1]), int(link1[removeLink-1][2])]
34    if -1 in conNodes or 0 in conNodes:
35      if max(conNodes) in inletNodes:
36        inletNodes.remove(max(conNodes))
37      if max(conNodes) in outletNodes:
38        outletNodes.remove(max(conNodes))
39
40    for node in conNodes:
41      if node != -1 and node != 0:
42        connectionNumber = int(node1[node-1][4])
43        connectedLinks = node1[node-1][7+connectionNumber:7+2*
              connectionNumber]
44        removeNodeIndex = 5+connectedLinks.index(removeLink)
45        removeLinkIndex = 7+connectionNumber+connectedLinks.index(removeLink
              )
46        node1[node-1].pop(removeLinkIndex)
47        node1[node-1].pop(removeNodeIndex)
48        node1[node-1][4] = node1[node-1][4]-1
49    return node1
50
51  def check_connectivity(inletConMap, outletConMap):
52    connected = False
53    for i in range(len(inletConMap)):
54      if inletConMap[i] == 1 and outletConMap[i] == 1:
55        connected = True
56        break
57    return connected
58
59  def check_link_for_link(node1, inletNodes):
```

```
60    for i in tqdm(range(len(linkRadi))):
61      node1 = remove_link(linkRadi[i][0], node1)
62      inletConMap = netRecon.map_connections(inletNodes, node1)
63      outletConMap = netRecon.map_connections(outletNodes, node1)
64      print "Connected?", check_connectivity(inletConMap, outletConMap)
65      if check_connectivity(inletConMap, outletConMap) == False:
66        print "Link␣to␣break␣connectivity:", linkRadi[i], "which␣is␣the", i
              +1, "of", len(linkRadi), "link␣to␣be␣removed."
67        break
68    return linkRadi[i]
69
70  def find_percolation_radius(node1):
71    inletNodes, outletNodes = netRecon.get_surface_nodes(node1)
72    original_inletConMap = netRecon.map_connections(inletNodes, node1)
73    original_outletConMap = netRecon.map_connections(outletNodes, node1)
74    linkRadi = sort_links_by_radius(link1)
75    node1 = netRecon.get_network_file(path+'node1')
76    radiFound = False
77    upLim = len(linkRadi)
78    lowestUpLim = 0
79
80    print "Finding␣percolation␣radius..."
81    t1 = time.clock()
82    while radiFound == False:
83      for i in range(upLim):
84        node1 = remove_link(linkRadi[i][0], node1, inletNodes, outletNodes)
85      inletConMap = netRecon.map_connections(inletNodes, node1)
86      outletConMap = netRecon.map_connections(outletNodes, node1)
87
88      if check_connectivity(inletConMap, outletConMap) == False:
89        highestUpLim = upLim
90        upLim = lowestUpLim + int((highestUpLim - lowestUpLim)/2)
91      elif check_connectivity(inletConMap, outletConMap) == True:
92        lowestUpLim = upLim
93        upLim = lowestUpLim + int((highestUpLim - lowestUpLim)/2)
94
95      prev_node1 = node1
96      node1 = netRecon.get_network_file(path+'node1')
97      inletNodes, outletNodes = netRecon.get_surface_nodes(node1)
98
99      print "Difference␣between␣limits:␣%6s␣␣␣␣␣(%5.2f/100␣%%)" % ((
            highestUpLim-lowestUpLim), 100-1.0*((highestUpLim-lowestUpLim)
            *100)/len(linkRadi))
100     if (highestUpLim - lowestUpLim) <= 1:
101       radiFound = True
102
```

```
  t2 = time.clock()
  print "\nTime␣to␣find␣percolation␣radius:␣%.2f␣sec.\n" % (t2-t1)
  return linkRadi[lowestUpLim]

percolationRadius = find_percolation_radius(node1)
linkRadi = sort_links_by_radius(link1)

print "*␣PERCOLATION␣RESULTS"
print "Radius␣of␣link␣to␣break␣connectivity:␣␣%.4E␣micro-meter␣(Link␣#␣%s)
    " % (percolationRadius[1], percolationRadius[0])
p_cb = 1-(1.0*linkRadi.index(percolationRadius)/len(linkRadi))
print "Bond␣percolation␣threshold:␣␣␣␣%.4f␣(%s␣of␣%s␣links␣removed)" % (
    p_cb, linkRadi.index(percolationRadius), len(linkRadi))
print "Average␣connection␣number:␣␣␣␣␣%.2f" % avg_con_num
print "B_c␣␣␣␣␣␣␣␣␣␣%.2f" % (avg_con_num*p_cb)

interfacialTension = 30.0/1000
advContAng = m.radians(30)
print "\n*␣FLUID␣PROPERTIES"
print "Interfacial␣tension␣␣␣␣%.1E␣␣[N/m]" % interfacialTension
print "Contact␣angle:␣␣␣␣␣␣␣%.2f␣␣␣␣[rad.]" % advContAng

capPres = ((interfacialTension*m.cos(advContAng))/percolationRadius[1])
    *(1-m.tan(advContAng)/m.sqrt(3))
print "\n*␣ASSUMING␣EQUILATERAL␣TRIANGLE:"
print "Threshold␣capillary␣pressure:␣␣%.3f␣␣[Pa]" % capPres

def find_corner_half_angles(link1):
  shapeFac_link = []
  notTri = 0
  for i in range(len(link1)):
    shapeFac_link.append(link1[i][4])
    if link1[i][4] > m.sqrt(3)/36:
      notTri += 1
  print "\n*␣FOLLOWING␣METHOD␣BY␣PATZEK␣AND␣SILIN␣(2001):"

  avg_shapeFac = np.average(shapeFac_link)
  b2_min = m.atan( (2/np.sqrt(3)) * m.cos( ( (m.acos(-12*m.sqrt(3)*
      avg_shapeFac) ) / (3) ) +(4*m.pi)/3 ) )
  b2_max = m.atan( (2/np.sqrt(3)) * m.cos( ( (m.acos(-12*m.sqrt(3)*
      avg_shapeFac) ) / (3) ) ) )
  print "The␣shape␣factor␣limits␣for␣triangles␣are␣[%s,%.3f]" % (0, m.sqrt
      (3)/36)
  print "The␣average␣shape␣factor␣is:␣␣␣␣␣%.3f" % avg_shapeFac
  print "Number␣of␣non-triangular␣elements:␣%s␣-␣%.1f␣%%␣of␣total." % (
      notTri, (1.0*notTri/len(link1))*100)
```

```
142
143   b2 = np.average([b2_min,b2_max])
144   if avg_shapeFac > m.sqrt(3)/36:
145     print "*"*80
146     print "WARNING: Average shape factor is larger than for triangular
          elements!"
147     print "*"*80
148
149   b1 = 0.5*m.asin(((m.tan(b2)+4*avg_shapeFac)/(m.tan(b2)-4*avg_shapeFac))*
        m.sin(b2))-0.5*b2
150   b3 = m.pi/2-(b1+b2)
151
152   print "This corresponds to a triangle with corner half angles of:"
153   print "b1 =   %.2f  [deg.]" % m.degrees(b1)
154   print "b2 =   %.2f  [deg.]" % m.degrees(b2)
155   print "b3 =   %.2f  [deg.]" % m.degrees(b3)
156   print " -----"
157   print "Sum =  %.2f  [deg.]" % (m.degrees(b1)+m.degrees(b2)+m.degrees(b3)
        )
158   return b1, b2, b3
159
160 def calc_cap_pres(b1,b2,b3,interfacialTension,advContAng,radius):
161   PC_SO_1 = (interfacialTension/radius)*(m.cos(advContAng)- (2*m.sin(
        advContAng))/(1/m.tan(b1)+1/m.tan(b2)))
162   print "Threshold capillary pressure (with G):   %.3f  [Pa]" % PC_SO_1
163
164 b1,b2,b3=find_corner_half_angles(link1)
165 calc_cap_pres(b1,b2,b3,interfacialTension,advContAng,percolationRadius[1])
```

B.10 clusterTrack.py

```
1  import numpy as np
2  import matplotlib.pylab as plt
3  import time
4  import os
5  import sys
6  import argparse
7  from tqdm import *
8  import netRecon
9
10 # The cluster track script use the original network files.
11 dest = os.getcwd()+'/originalNetworks'
12 if os.path.exists(dest):
13   path = 'originalNetworks/'
14   link1 = netRecon.get_network_file(path+'link1')
```

```
15    link2 = netRecon.get_network_file(path+'link2')
16    node2 = netRecon.get_network_file(path+'node2')
17    node1 = netRecon.get_network_file(path+'node1')
18    print "Found original network data files."
19  else:
20    print "Plese locate original network data files."
21    sys.exit()
22
23  nodesat_wf = np.loadtxt('nodesat_wf.dat', skiprows=2)
24  def locate_clusters():
25    def map_connections(fromNodes, marker):
26        def append_connected_nodes(ap_node):
27          connectionNumber = int(node1[ap_node-1][4])
28          connectedNodes = node1[ap_node-1][5:5+connectionNumber]
29          for i in range(len(connectedNodes)):
30            connectedNodes[i] = int(connectedNodes[i])
31          connectedNodes = sorted(connectedNodes, reverse=True)
32          for i in connectedNodes:
33            if i > 0 and i not in checkNodes and i not in usedNodes and
                  oil_sat[i-1] != 0:
34              checkNodes.append(i)
35              usedNodes.append(i)
36        conMap = [0]*len(node1)
37        checkNodes = []
38        usedNodes = []
39        clusters = []
40        for node in tqdm(fromNodes):
41          # Exit condition to avoid unneccasary loops.
42          exit = False
43          for i in range(len(clusters)):
44            if node in clusters[i]:
45              exit = True
46              break
47          if exit == True:
48            continue
49          temp_cluster = []
50          checkNodes = []
51          usedNodes = []
52          if node not in usedNodes and node not in checkNodes and oil_sat[
                node-1] != 0:
53            checkNodes.append(node)
54            usedNodes.append(node)
55
56          while len(checkNodes)>0:
57            append_connected_nodes(checkNodes[0])
58            conMap[checkNodes[0]-1] = 1
```

```
            temp_cluster.append(checkNodes[0])
            checkNodes.remove(checkNodes[0])
          temp_cluster = sorted(temp_cluster)
          if temp_cluster not in clusters:
            clusters.append(temp_cluster)
        return conMap, clusters

    oil_sat = []
    oil_sat_nodes = []
    for i in range(len(nodesat_wf)):
      oil_sat.append(1-nodesat_wf[i][6])
      if nodesat_wf[i][6] != 1:
        oil_sat_nodes.append(i+1)

    conMap, clusters = map_connections(oil_sat_nodes,1)
    return clusters, oil_sat_nodes

def get_cluster_size():
    clusters, oil_sat_nodes = locate_clusters()
    oil_nodes = 0
    max_cluster = 0
    max_cluster_idx = []
    cluster_size =  []
    for i in range(len(clusters)):
      oil_nodes += len(clusters[i])
      cluster_size.append(len(clusters[i]))
      if len(clusters[i]) > max_cluster:
        max_cluster = len(clusters[i])
        max_cluster_idx = clusters[i]

    cluster_size = sorted(cluster_size, reverse=True)
    cluster_size = np.array(cluster_size)
    unique_clusters = np.unique(cluster_size)
    singlet_clusters = len(np.where(cluster_size == 1)[0])
    return cluster_size, max_cluster_idx, clusters, oil_nodes

def print_cumulative_dist(unique_clusters, P_dist, name):
    file = open('output/cumulative_cluster_dist'+name+'.txt', 'w')
    for i in range(len(unique_clusters)):
      string = str(unique_clusters[i])+'\t'+str(P_dist[i])
        file.write(string+'\n')
      file.close()

def get_cumulative_clust_dist():
    cluster_size, max_cluster_idx, clusters, oil_nodes = get_cluster_size()
    # Volume weighted
```

```
105   cluster_vol = []
106   for cluster in clusters:
107     cluster_vol_temp = 0.0
108     for node in cluster:
109       cluster_vol_temp += node2[node-1][1]
110     cluster_vol.append(cluster_vol_temp)
111
112   print "Number of unique clusters:   %s" % len(cluster_vol)
113   print "Number of clusters of unique volume: %s" % len(np.unique(
          cluster_vol))
114   tot_cluster_vol = np.sum(cluster_vol)
115   print "Total volume of clusters:  %.3f" % tot_cluster_vol
116
117   cluster_vol = np.array(cluster_vol)
118   unique_clusters_vol = np.unique(cluster_vol)
119   cluster_vol_dist = []
120   cluster_vol_unique = []
121   num_eq_cluster_vol = 0
122   for cluster in unique_clusters_vol:
123     num_eq_cluster_vol = len(np.where(cluster_vol == cluster)[0])
124     cluster_vol_dist.append([cluster, num_eq_cluster_vol])
125     cluster_vol_unique.append(cluster*num_eq_cluster_vol)
126   cluster_vol_unique.sort()
127
128   print "Vol. frac. of largest cluster: %.2f" % (cluster_vol_unique[-1]/
          tot_cluster_vol)
129
130   P=0
131   f_cluster_vol = []
132   P_dist_vol = []
133   x_val = []
134   for cluster in cluster_vol_unique:
135     f_cluster_vol.append((cluster)/tot_cluster_vol)
136     P += f_cluster_vol[-1]
137     P_dist_vol.append(P)
138
139   print_cumulative_dist(f_cluster_vol,P_dist_vol, 'vol')
140
141   # CLUSTER DIST. BASED ON NUMBER OF NODES IN EACH CLUSTER
142   #unique_clusters = np.unique(cluster_size)
143   #cluster_dist = []
144   #f_cluster = []
145   #P_dist = []
146   #P=0
147   #
148   #for cluster in unique_clusters:
```

```
149   # num_eq_cluster = len(np.where(cluster_size == cluster)[0])
150   # cluster_dist.append([cluster, num_eq_cluster])
151   # f_cluster.append((cluster_dist[-1][0]*cluster_dist[-1][1])/(oil_nodes
          *1.0))
152   # P += f_cluster[-1]
153   # P_dist.append(P)
154   #f_cluster.reverse()
155
156 def plot_cumulative_clust_dist(f_cluster, P_dist):
157   fig = plt.figure(figsize=(9,4))
158   ax = plt.subplot(111)
159   ax.semilogx(f_cluster,P_dist)
160   plt.ylim((0,1))
161   plt.xlim((0,np.max(f_cluster)))
162   plt.show()
163
164 def print_cluster_stats(cluster_size, max_cluster_idx, clusters, oil_nodes
      ):
165   print "Oil␣saturated␣nodes:␣␣␣%s␣(%.1f␣%%␣of␣all␣nodes)" % (oil_nodes,
          oil_nodes*100.0/len(node1))
166   print "Number␣of␣clusters:␣␣%s" % len(clusters)
167   print "Average␣cluster␣size:␣␣%.4f" % ((oil_nodes*1.0)/len(clusters))
168   print "Size␣of␣maximum␣cluster:␣␣␣%s␣(%.1f␣%%␣of␣cluster␣nodes)" % (len(
          max_cluster_idx),len(max_cluster_idx)*100.0/oil_nodes)
169   print "Number␣of␣singlet␣'clusters':␣␣%s" % len(np.where(cluster_size ==
           1)[0])
170
171 cluster_size, max_cluster_idx, clusters, oil_nodes = get_cluster_size()
172 print_cluster_stats(cluster_size, max_cluster_idx, clusters, oil_nodes)
173 #get_cumulative_blob_dist()
174 #plot_cumulative_clust_dist(f_cluster_vol,P_dist_vol)
```

B.11 createNetworkXML.py

```
1 import numpy as np
2 import os
3 import sys
4 import time
5 import argparse
6
7 import netRecon
8
9 parser = argparse.ArgumentParser(prog='createNetworkXML', description="A␣
      program␣to␣create␣XML-files␣for␣plotting␣ball␣&␣stick␣representations␣
      of␣the␣networks.␣The␣files␣can␣be␣imported␣into␣the␣'Ball␣&␣Stick'␣app
```

```
    for Windows 10 by fourelem.")
10  parser.add_argument("-colorResidual", action='store_true', help="Color
        nodes with residual oil saturation.", default=False)
11  parser.add_argument("-onlyResidual", action='store_true', help="Only show
        nodes with residual oil saturation.", default=False)
12  parser.add_argument("-onlyWater", action='store_true', help="Only show
        nodes with water saturation = 1.", default=False)
13  args = parser.parse_args()
14
15  node1 = netRecon.get_network_file('originalNetworks/node1')
16  node2 = netRecon.get_network_file('originalNetworks/node2')
17  link1 = netRecon.get_network_file('originalNetworks/link1')
18  link2 = netRecon.get_network_file('originalNetworks/link2')
19
20  foldername = os.path.basename(os.getcwd())
21  if args.colorResidual == True:
22    xmlFileName = foldername + '_ballAndStick_residual.xml' # e.g. F8_plot.
        xml or node22_plot.xml
23  elif args.onlyResidual == True:
24    xmlFileName = foldername + '_ballAndStick_OnlyRes.xml'
25  elif args.onlyWater== True:
26    xmlFileName = foldername + '_ballAndStick_OnlyWat.xml'
27  else:
28    xmlFileName = foldername + '_ballAndStick.xml' # e.g. F8_plot.xml or
        node22_plot.xml
29
30  file=open('output/'+xmlFileName, 'w+')
31  file.write('<?xml version="1.0" encoding="utf-8"?>\n<Model xmlns="http://
        atom.fourelem.com/ns/2016">\n')
32  file.write('\t<Group>\n')
33
34  if args.onlyResidual == True:
35    t1 = time.clock()
36    print "Locating maximum cluster ..."
37    import clusterTrack
38    cluster_size, max_cluster_idx, clusters, oil_nodes = clusterTrack.
        get_cluster_size()
39    print "         [DONE] - %.2f sec." % (time.clock()-t1)
40
41  tic = time.clock()
42  print "Creating XML-file...",
43  sys.stdout.flush()
44
45  # Ball properties
46  ballSizeFactor = 10e3
47  ballColor = "Red"
```

```
48  ballNames = "N"
49  locationMult = 10000
50
51  ballColorOil = "Lime" #"LawnGreen"
52  ballColorOilMax = "SaddleBrown"
53  ballColorWat = "RoyalBlue"
54
55  if args.colorResidual == True or args.onlyResidual == True or args.
        onlyWater == True:
56    nodesat_wf = np.loadtxt('nodesat_wf.dat', skiprows=2)
57
58    oil_node_idx = []
59    wat_node_idx = []
60    for i in range(len(nodesat_wf)):
61      if nodesat_wf[i][6] < 1:
62        oil_node_idx.append(i+1)
63      else:
64        wat_node_idx.append(i+1)
65
66  # Stick properties
67  stickSizeFactor = 0.5*10e3
68  stickColor = "GhostWhite"
69  stickColorOil = "OliveDrab"
70  stickColorWat = "DodgerBlue"
71  stickNames = "S"
72
73  # Node/link plot stats
74  numOnlyResNodes = 0
75  numOnlyWatNodes = 0
76  numOnlyResLinks = 0
77  numOnlyWatLinks = 0
78
79
80  for i in range(len(node1)):
81    if args.colorResidual == True:
82      if nodesat_wf[i][6] < 1:
83        file.write('\t\t<Ball␣Id␣="%s%s"␣Position="%s,%s,%s"␣Radius="%s"␣
              Color="%s"/>\n' % ( ballNames, (i+1), node1[i][1]*locationMult,
              node1[i][2]*locationMult, node1[i][3]*locationMult, node2[i][2]*
              ballSizeFactor, ballColorOil))
84      else:
85        file.write('\t\t<Ball␣Id␣="%s%s"␣Position="%s,%s,%s"␣Radius="%s"␣
              Color="%s"/>\n' % ( ballNames, (i+1), node1[i][1]*locationMult,
              node1[i][2]*locationMult, node1[i][3]*locationMult, node2[i][2]*
              ballSizeFactor, ballColorWat))
86    elif args.onlyResidual == True:
```

```
87      if nodesat_wf[i][6] < 1:
88        if ((i+1) in max_cluster_idx):
89          file.write('\t\t<Ball␣Id␣="%s%s"␣Position="%s,%s,%s"␣Radius="%s"␣
                Color="%s"/>\n' % ( ballNames, (i+1), node1[i][1]*locationMult
                , node1[i][2]*locationMult, node1[i][3]*locationMult, node2[i
                ][2]*ballSizeFactor, ballColorOilMax))
90        else:
91          file.write('\t\t<Ball␣Id␣="%s%s"␣Position="%s,%s,%s"␣Radius="%s"␣
                Color="%s"/>\n' % ( ballNames, (i+1), node1[i][1]*locationMult
                , node1[i][2]*locationMult, node1[i][3]*locationMult, node2[i
                ][2]*ballSizeFactor, ballColorOil))
92        numOnlyResNodes += 1
93      else:
94        file.write('\t\t<Ball␣Id␣="%s%s"␣Position="%s,%s,%s"␣Radius="%s"␣
              Color="%s"/>\n' % ( ballNames, (i+1), node1[i][1]*locationMult,
              node1[i][2]*locationMult, node1[i][3]*locationMult, 0, "White"))
95    elif args.onlyWater == True:
96      if nodesat_wf[i][6] == 1:
97        file.write('\t\t<Ball␣Id␣="%s%s"␣Position="%s,%s,%s"␣Radius="%s"␣
              Color="%s"/>\n' % ( ballNames, (i+1), node1[i][1]*locationMult,
              node1[i][2]*locationMult, node1[i][3]*locationMult, node2[i][2]*
              ballSizeFactor, ballColorWat))
98        numOnlyWatNodes += 1
99      else:
100       file.write('\t\t<Ball␣Id␣="%s%s"␣Position="%s,%s,%s"␣Radius="%s"␣
              Color="%s"/>\n' % ( ballNames, (i+1), node1[i][1]*locationMult,
              node1[i][2]*locationMult, node1[i][3]*locationMult, 0, "White"))
101   else:
102     file.write('\t\t<Ball␣Id␣="%s%s"␣Position="%s,%s,%s"␣Radius="%s"␣Color
            ="%s"/>\n' % ( ballNames, (i+1), node1[i][1]*locationMult, node1[i
            ][2]*locationMult, node1[i][3]*locationMult, node2[i][2]*
            ballSizeFactor, ballColor))
103 file.write('\t</Group>\n')
104 file.write('\t<Group>\n')
105 for i in range(len(link1)):
106   if args.onlyResidual == True:
107     if int(link1[i][1]) != -1 and int(link1[i][2])!= 0 and (int(link1[i
          ][1]) in oil_node_idx) and (int(link1[i][2]) in oil_node_idx):
108       file.write('\t\t<Stick␣Id␣="%s%s"␣BallIds="N%s,N%s"␣Radius="%s"␣
              Color="%s"/>\n' % ( stickNames, (i+1), int(link1[i][1]), int(
              link1[i][2]),  link1[i][3]*stickSizeFactor, stickColorOil ))
109       numOnlyResLinks += 1
110   elif args.onlyWater == True:
111     if int(link1[i][1]) != -1 and int(link1[i][2])!= 0 and (int(link1[i
          ][1]) in wat_node_idx) and (int(link1[i][2]) in wat_node_idx):
112       file.write('\t\t<Stick␣Id␣="%s%s"␣BallIds="N%s,N%s"␣Radius="%s"␣
```

```
            Color="%s"/>\n' % ( stickNames, (i+1), int(link1[i][1]), int(
            link1[i][2]),  link1[i][3]*stickSizeFactor, stickColorWat ))
113       numOnlyWatLinks += 1
114   else:
115     if int(link1[i][1]) != -1 and int(link1[i][2])!= 0:
116       file.write('\t\t<Stick␣Id␣="%s%s"␣BallIds="N%s,N%s"␣Radius="%s"␣
            Color="%s"/>\n' % ( stickNames, (i+1), int(link1[i][1]), int(
            link1[i][2]),  link1[i][3]*stickSizeFactor, stickColor ))
117 file.write('\t</Group>\n')
118 file.write('</Model>')
119 file.close()
120 toc = time.clock()
121
122 print "␣␣␣␣␣[DONE]␣-␣%.2f␣sec.␣\n" % (toc-tic)
123 print "␣*␣Nodes␣*"
124 if args.colorResidual == True:
125   print "Nodes␣plotted:␣␣␣␣␣%s" % len(node1)
126   print "Color␣of␣oil␣sat.␣nodes:␣%s␣␣␣␣(So>0)" % ballColorOil
127   print "Color␣of␣water␣sat.␣nodes:␣%s␣\n" % ballColorWat
128 elif args.onlyResidual == True:
129   print "Only␣oil␣saturated␣nodes␣(So>0)␣are␣plotted."
130   print "Nodes␣plotted:␣␣␣␣␣%s" % numOnlyResNodes
131   print "Color␣of␣oil␣sat.␣nodes:␣%s␣␣␣␣(So>0)␣\n" % ballColorOil
132 elif args.onlyWater == True:
133   print "Only␣fully␣water␣saturated␣nodes␣are␣plotted."
134   print "Nodes␣plotted:␣␣␣␣␣%s" % numOnlyWatNodes
135   print "Color␣of␣water␣sat.␣nodes:␣%s␣\n" % ballColorWat
136 else:
137   print "Nodes␣plotted:␣␣␣%s" % len(node1)
138   print "Color␣of␣nodes:␣␣␣␣%s␣\n" % ballColor
139 print "␣*␣Links␣*"
140 if args.colorResidual == True:
141   print "Links␣plotted:␣␣␣␣␣%s" % len(link1)
142   print "Color␣of␣links:␣␣␣␣␣␣%s\n" % stickColor
143 elif args.onlyResidual == True:
144   print "Only␣oil␣saturated␣links␣are␣plotted."
145   print "Links␣plotted:␣␣␣␣␣%s" % numOnlyResLinks
146   print "Only␣oil␣saturated␣links:␣␣%s␣\n" %stickColorOil
147 elif args.onlyWater == True:
148   print "Only␣water␣saturated␣links␣are␣plotted."
149   print "Links␣plotted:␣␣␣␣␣%s" % numOnlyWatLinks
150   print "Color␣of␣water␣sat.␣links:␣%s␣\n" % stickColorWat
151 else:
152   print "Links␣plotted:␣␣␣%s" % len(link1)
153   print "Color␣of␣sticks:␣%s␣\n" % stickColor
154
```

```
155
156 print "File␣('%s')␣saved␣to␣output␣folder." % xmlFileName
```