Delft University of Technology

Faculty of Civil Engineering and Geosciences

Analysis of Two Problems in Network Transport

Flow through static foam in artificial fractures

and

Steady-state two-phase relative permeabilities in microfluidic devices

A thesis in partial fulfillment of the Master of Science in Geo-Energy Engineering

Author
E.J.M. Obbens

Supervisor
Prof. Dr. W.R. Rossen

Committee
Dr. D.V. Voskov
Dr. K.H.A.A. Wolf
Prof. Dr. S. Cox

Delft, The Netherlands
August 14, 2022

Summary

In this thesis two problems in network transport are analyzed.

The first problem concerns flow through static foam in artificial fractures. The objective is to determine whether the assumption of Li et al. (2021), that capillary pressure is uniform in the region of interest, with static foam in an artificial fracture, is justified. This would be the case if water can flow through Plateau borders at a rate that is quick enough for pressure differences to dissipate rapidly. Images of foam in the fractures are turned into networks of slits that are scaled down to flow through Plateau borders in foam. The results of this show that the capillary pressure can be assumed to be uniform.

The second problem concerns steady-state two-phase flow in microfluidic devices. The objective is to determine whether two phases can simultaneously flow at comparable fractional flows through a microfluidic device without alternating pore occupancy. This would be the case if the total mobility values of both phases are similar. To find the relative permeability values, a microfluidic device is simulated consisting of interface shapes based on the findings of Cox et al. (2022), arranged in a network according to bond percolation theory. The results show that it is unlikely that two phases with similar viscosity values can maintain steady-state flow at comparable fractional flows, and impossible if the viscosity ratio is that of gas and water. This implies that flow experiments done using microfluidic devices reflect the high-capillary-number flow regime where flow paths fluctuate in the pore network.

Contents

List of Figures

List of Tables

Nomenclature

Symbols

ΔP Pressure difference along the slit or Plateau border Pa

μ Viscosity $Pa \cdot s$

σ Surface tension N/m

B_{slit} Half the width of a slit m

H Hydraulic aperture m

L Length of a slit or Plateau border along the flow direction m

P_c Capillary pressure Pa

Q_{PB} Flow rate through a Plateau border $\frac{m^3}{s}$

Q_{slit} Flow rate through a slit $\frac{m^3}{s}$

$Q_{zero-stress}$ Flow rate through a Plateau border with a zero stress interfacial boundary condition $\frac{m^3}{s}$

$Q_{zero-velocity}$ Flow rate through a Plateau border with a zero velocity interfacial boundary condition $\frac{m^3}{s}$

R_{PB} Plateau border radius m

1 General Introduction

Liquid foams are cellular materials composed of gas enclosed by liquid films (lamellae). The rheological properties of foam give it a variety of applications in reservoir engineering. In environmental remediation, foams are used to displace non-aqueous liquids to clean aquifers and soil. For carbon storage it increases capillary trapping and improves sweep, resulting in less plume migration. For enhanced oil recovery (EOR), foam increases the sweep across heterogeneous layered reservoirs while simultaneously trapping gas such as carbon-dioxide (Rossen, 1996).

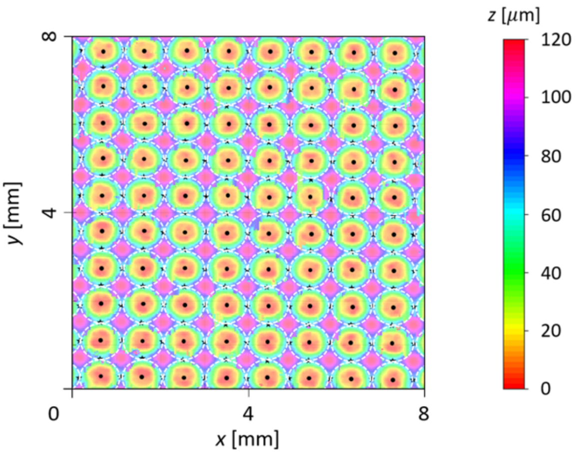
In order to enhance oil recovery, reservoirs are sometimes injected with gases. This results in higher production and recovery rates. Because the production of foam requires gas, it is best suited to use foam in processes in which gas is already being used. During this process, there is a possibility that gas will break through to the production well, which will prevent it from reaching large portions of the reservoir. This happens when preferential gas flow paths lead towards the production well, which results in a low sweep efficiency and a high produced gas/oil ratio. Consequently, this causes a decrease in the amount of oil produced (Rossen, 1996). The extent to which foam can sweep throughout the reservoir is a significant factor that determines how successful a foam application will be. Foam injection has not yet become a standard method for the recovery of oil in reservoir engineering, despite the fact that it possesses a variety of unique properties for a wide range of applications. One possible explanation is that the current foam flow models are unable to accurately predict how fluids will behave in a reservoir.

Two different topics concerning steady state flow calculations are discussed in this thesis. The first topic is about estimating the time required for capillary pressure to equalize across a small region by surfactant solution flowing through the Plateau borders of foam in model fractures. The purpose is to see whether the assumption that the capillary pressure is uniform over the region of interest is correct in (Li et al., 2021). The second topic is about finding the relative permeabilities of two-phase flow in a microfluidic device, to see whether these devices are able to represent three-dimensional rock correctly.

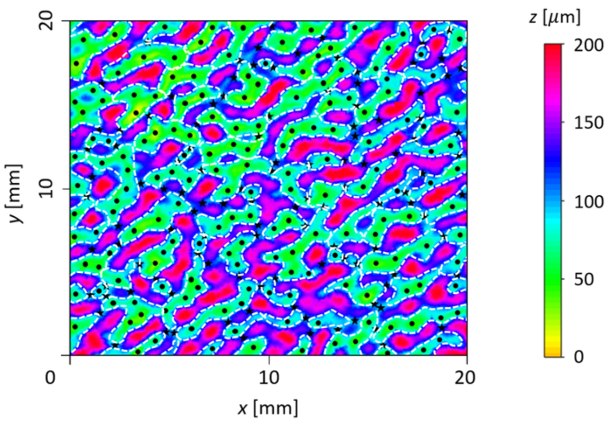
2 Artificial Fracture Topic

2.1 Introduction

(Li et al., 2021) introduced a method to determine the capillary pressure and water saturation in two model fractures based on 2D images of spatial water distribution in the model fractures. The top glass plate is smooth and the bottom glass plate is roughened on its top surface. The roughening and hydraulic aperture is different for the models. The measured hydraulic aperture of Model 1 and Model 2 obtained are 46 and 78 μm, respectively. Figure 1 shows the roughness patterns for Model 1 and 2. The pattern for Model 1 is shaped as a regular checkerboard grid of peaks and troughs and Model 2 is irregular.



(a) Model 1

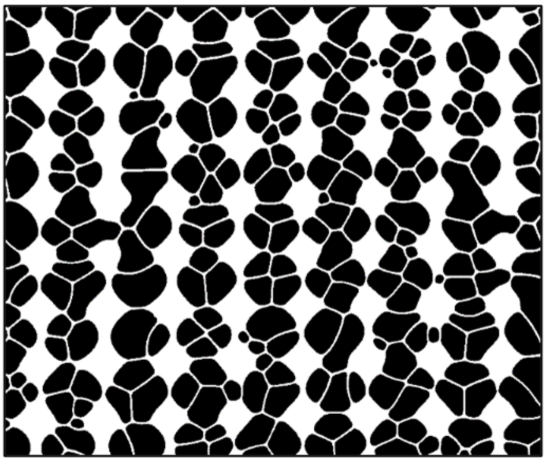


(b) Model 2

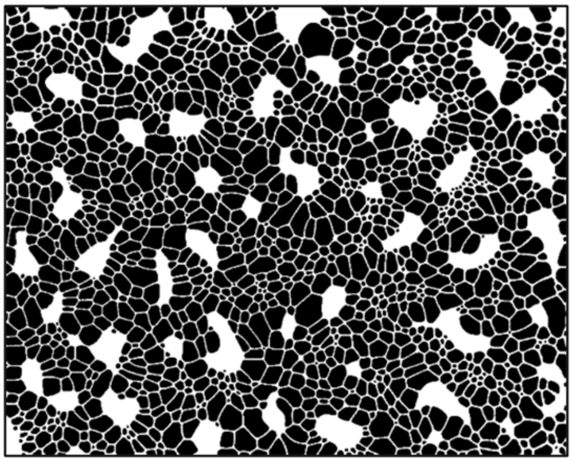
Figure 1: The roughness pattern that is applied to the glass plate for Model 1 and Model 2 (Li et al., 2021).

Foam is pre-generated and injected into the fractures. The inlet and outlet valves are closed when the foam reaches a steady state. A high-speed camera is used for a period of 24 hours to capture images of a small region (see Fig. 2). Most of the water of the foam resides in the regions where the aperture is narrowest. During this period the gas and the water in the foam redistributes. Water flows through water-occupied zones and Plateau borders along the top and bottom of lamellae in response to pressure gradients, while the gas mostly stays trapped. The purpose of Li et al.'s study was to quantify gas diffusion between trapped bubbles. A crucial assumption of this method is that the capillary pressure is roughly uniform within each image at any time. This is the case if the water flow through the network is of a magnitude that can equalize the pressure across a small region quickly enough. To find this necessary water flow rate a method has been developed that utilises the high-speed camera images of Li et al. (2021). The images of the phase distribution progression during the experiment are included in Appendix A. The first images of the phase distributions in both models are shown in Fig. 2. The water and gas are shown in white and black, respectively. The distribution of the roughness pattern determines the location where water fills locations of narrow aperture, which we call "water zones". This results in water zones that are distributed evenly for Model 1 and unevenly for Model 2. The flow rate cannot be calculated for some images: this includes the images of Model 1 after $t = 1.75\,h$ because these WP clusters in the narrow locations became too small to calculate a capillary pressure, and the image of Model 2 at $t = 17.10\,h$ because the water zones appear on the image to be disconnected from one side to another.

Although most lamellae appear to be nearly straight in the images, there is variation in the curvature of the lamellae. The smallest radius of curvature is approximately 1 mm for both models. The capillary pressure is roughly uniform within each image if the wetting phase is able to flow through the model at a rate quick enough to alleviate any capillary pressure difference that arises. It is unknown which rate would be enough, so a rough estimate is made by computing the time it takes for 10% of the total estimated volume of water in the image to flow through the model if the pressure difference is 10% of the capillary pressure in the model at that time. The total water volume in each image is found by Li et al. (2021).

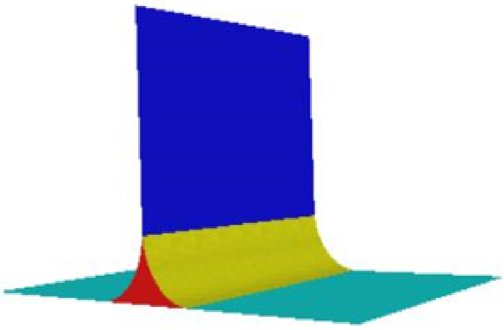


(a) Model 1 at $t = 0.09\,h$.

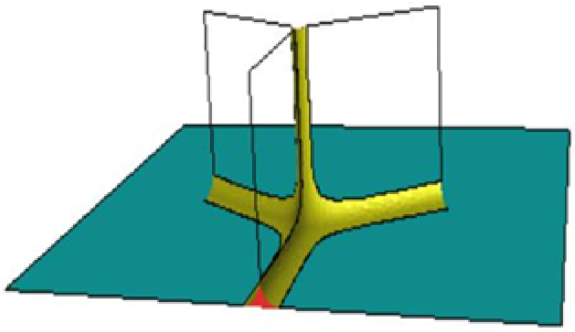


(b) Model 2 at $t = 0.10\,h$.

Figure 2: The first images of the phase distribution in both Model 1 and 2. The image on the left is of Model 1 and taken at $t = 0.09\,h$ the image on the right is of Model 2 and taken at $t = 0.10\,h$. The water is shown in white and the gas in black. The dimensions of the images are $7.8 \times 6.8\,mm$ for Model 1 and $12.3 \times 9.8\,mm$ for Model 2.



(a) Straight Plateau border.



(b) Plateau border vertex.

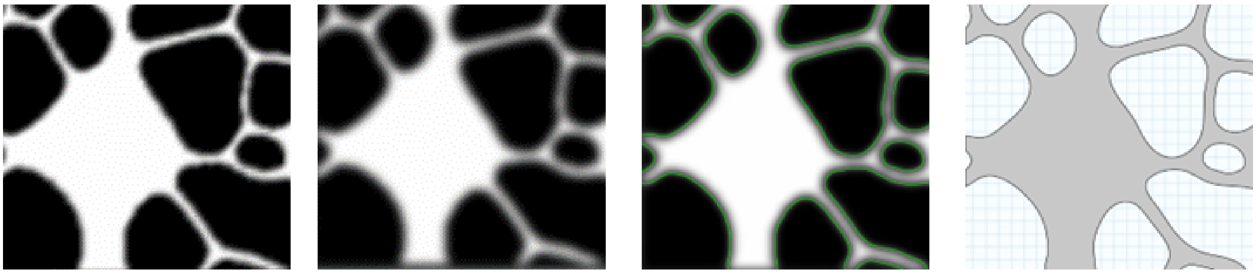
Figure 3: A straight Plateau border and a Plateau border vertex. The interfacial boundaries of the Plateau borders are marked in yellow and the cross-section of the Plateau border is marked in red. (Alonso et al., 2002). The vertical Plateau border in the image on the right does not contribute to flow.

In the experiment the foam bubbles redistribute with time but not significantly over the time scale that is expected for the capillary pressure redistribution. Thus the distribution of the phases is constant for each calculation of the flow rate. The water in the model consists of water-occupied zones (in locations of narrow aperture) and lamellae. The water can flow across the model through Plateau borders as shown in Fig. 3. These are formed where lamellae connect at the top and bottom of the model with the glass plates. In a foam with uniform capillary pressure these Plateau borders have the same cross-sectional area everywhere. Because water zones are relatively large compared to the lamellae, the water can flow through those zones relatively unobstructed. The flow through a network is largely determined by the largest resistances to flow, which in this case is the flow through the Plateau borders. We assume this because the width (seen from above) of the Plateau borders is so much narrower than the width of the water-filled zones in locations of narrow aperture.

2.2 Method

The goal is to see whether the assumption that the capillary pressure in these models is roughly uniform is correct. The capillary pressure is uniform if water moves through the model quickly enough for pressure differences to equalize. This flow rate is found in two dimensions with COMSOL and transformed into three dimensions by scaling the flow rate through a slit with the hydraulic aperture of the network as the height of the slit down to flow through Plateau borders, with the width set by capillary pressure.

The first step is to turn the images of the phase distributions obtained by Li et al. (2021) into two-dimensional geometries that enable flow calculations in COMSOL Multiphysics®. Visualizations of the steps are shown in Fig. 4. If edge detection is utilised with the original images, the edges become too jagged. This causes artifacts where there are width fluctuations within some lamellae. So the image is smoothed out using a Gaussian blur. This image is then used with the "Image to Curve" add-in of COMSOL Multiphysics® to detect the contours of the regions occupied by the water. The contour threshold is modified so that none of the lamellae are disconnected. These contours are then converted into a solid curve geometry, which is used as the boundary within which the water flows through the model.



(a) (b) (c) (d)

Figure 4: Detail of the bottom left corner of Model 1 at t=0.09 hours. (a) The original image. The resolution of the image causes the interfacial boundaries to appear jagged. (b) Gaussian filter is applied to smooth-out the image. (c) Contours (green lines) are detected at the interfacial boundaries. (d) The contours are converted to a geometry that contains the water and is used for the flow calculation.

The fluid is COMSOL Multiphysics® Material "H2O (water)" at a temperature of 293.15 K. It has a density of approximately 10^3 kg/m^3 and viscosity of 10^{-4} Pa·s. The "Creeping flow" module is used, which neglects the inertial term in the Navier-Stokes equation. All boundaries have the zero-velocity boundary condition. The flow rate is calculated for both principal directions across the images. The inlet and outlet pairs are set on opposite sides of each image. Figure 5 shows the velocity contours and pressure fields for the first image of both models. What stands out here is that there is almost no pressure drop in the water-filled regions of the image. Because the flow is calculated through a two dimensional image, it is as if the water flows through a network of slits of varying widths with infinite height (i.e., ignoring the effects of top and bottom plates) at the given pressure difference. If the channels are instead shaped like Plateau borders, then the resistance to flow is greater for the Plateau borders. The height of a Plateau border is smaller than that of a water-filled region that reaches from the bottom plate to the top glass plate. So it is assumed that most of the resistance to flow is within the Plateau borders.

This results in a velocity through the slit model that is greater than possible in Plateau borders because the fluid is more constricted in the Plateau borders (see Fig. 6). Moreover, one must assume a height to compute a flow rate. The flow rate is calculated with the velocity through a slit using the hydraulic aperture as the height, and then related to the flow through a Plateau border. This is possible under the assumption that the width of the slits and Plateau borders are each roughly uniform throughout the model at any given time. The scaling factor is given in Eq. 1, where an equal pressure gradient is assumed for both flow geometries:

$$Scaling\ Factor = \frac{Q_{PB}}{Q_{slit}} \tag{1}$$

A schematic showing why the flow through Plateau borders is smaller than through a slit with equal width is shown in Fig. 6, where, for illustration, the slit is assumed to have the same width as the Plateau border.

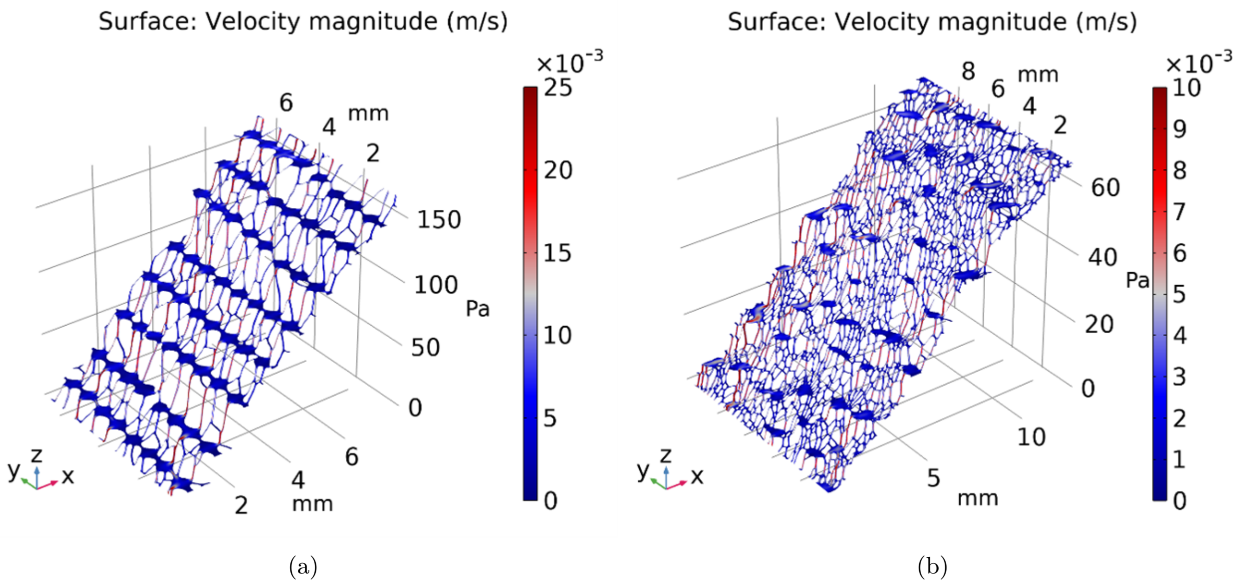


Figure 5: The resulting velocity contours (color scale) and pressure field (height) (a) for Model 1 at t = 0.09 hr and (b) Model 2 at t = 0.10 hr. In this example the fluid flows in the negative x-direction. A pressure difference of 10% of the capillary pressure is assumed here.

Note that the flow rate through the Plateau borders is independent of the height of the lamella, as long as it is as least twice as tall as one Plateau border.

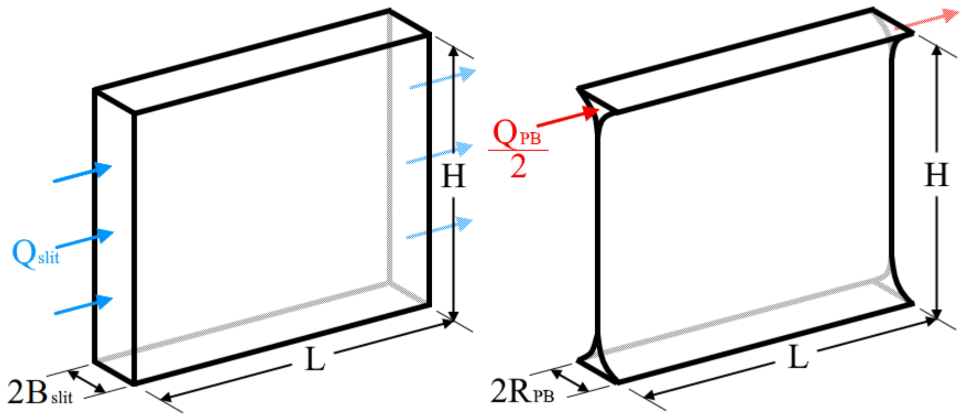


Figure 6: Schematic of the flow through a slit and flow through the Plateau borders. If width of the slit and Plateau borders is similar, then the Plateau borders constrict the flow of the liquid more.

The local velocities computed by COMSOL Multiphysics® are sensitive to local slit width, which, in the images, is not uniform. We use ImageJ to measure the widths of the slits in the images as shown in Fig. 7.

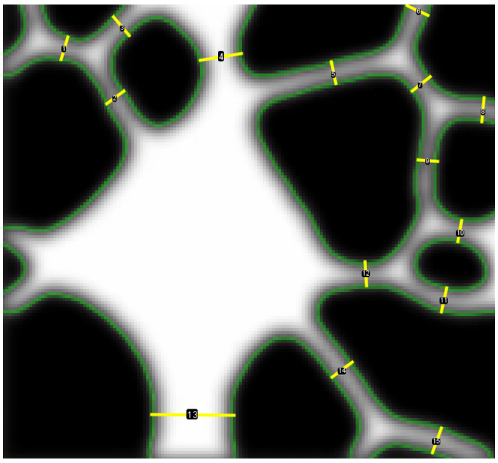


Figure 7: An example of a slit width measurements using ImageJ. The yellow lines indicate the extent of each measurement. These lines are manually drawn in the middle of the slits starting and ending on the contours recognized in COMSOL Multiphysics®. The measurements result in a width and an angle towards the horizontal.

Figure 1 shows that the model is asymmetric: the spacing of the peaks on the bottom plate is different in the two directions. Because the average slit width may differ depending on orientation, a weighted average is used with the orientation as the weights. These averages are found in Appendix A for both directions in each image. In the images of Model 1 the distance between water zones in locations of narrow aperture is different in the x and y directions. The liquid clusters are closer together in the y-direction, and the slit width wider for the flow in the y-direction. Are they so wide that these connections are liquid lenses instead of lamellae with Plateau borders? If so, then flow in the flow across Model 1 in the y-direction is not through Plateau borders as we assume. However, the capillary pressures reported by Li et al. (2021) are too great to allow for liquid lenses across pore bodies. Therefore we assume that all these paths comprise lamellae with Plateau borders. For Model 2 the distribution is irregular, resulting in width values that are more similar in the two directions.

The flow of the water flowing through an infinite slit is calculated with Eq. 2 (Bird et al., 2014).

$$Q_{slit} = \frac{2}{3}\frac{\Delta P B_{slit}^3 H}{\mu L} \tag{2}$$

The next step is to find an equation for the flow rate through Plateau borders as a function of the Plateau border width. The width of the Plateau borders is not the same as that of the slits, because the edges shown in the images could be distorted. The width of the Plateau borders is related to the capillary pressure in the model. Therefore the width is obtained as a function of the capillary pressure using Equation 3, which assumes that the Plateau borders are nearly cylindrical in shape.

$$P_c = \frac{\sigma}{R_{PB}} \tag{3}$$

Where R_{PB} is the radius of the Plateau border, i.e. $\frac{1}{2}$ the width, as determined for each image by Li et al. (2021).

Equation 3 applies only to cylindrical Plateau borders. More generally, the capillary pressure depends on two radii of curvature. In a Plateau border these are the radius of the interfacial curvature and the curvature of the lamella. Because the capillary pressure is uniform, the interfacial curvature changes along a Plateau border if the lamella is tightly curved. In the model, the lamellae can curve in three dimensions. Two dimensions of curvature are observed by looking down on the images, and it shows that the distance over which the lamellae curves are in the order of mm or cm. That is multiple orders of magnitude greater than the radius of the Plateau border determined by capillary pressure. The curvature in the third dimension is determined by the curvature of the height on the bottom glass plate. Fig. 1 makes clear that this curvature is likewise negligible compared to the radius of the Plateau borders. Because the curvature of the lamellae is insignificant compared to that of the interfacial curvature, we can assume that the surfaces of the Plateau borders are well approximated as cylinders.

The equations for flow through a Plateau border are obtained by calculating the flow rate through Plateau borders of differing widths in COMSOL Multiphysics®. Due to symmetry, just one half of the Plateau border is simulated. The symmetry plane is given a symmetry BC, and the boundary of the Plateau border with the glass is given a zero velocity BC. We do calculations assuming both no-slip on the gas-water interface and zero shear stress at that interface, to allow for surfactants with large or negligible surface viscosity.

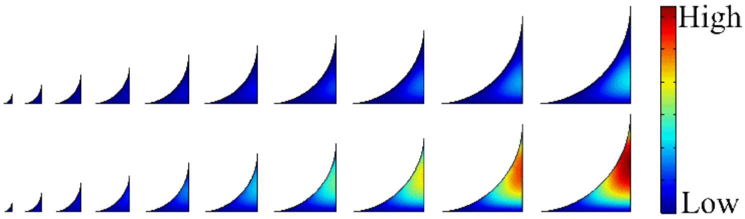


Figure 8: Velocity magnitude (m/s) through a Plateau border with an interfacial zero velocity BC (upper row) and an interfacial zero stress BC (lower row). Velocity is zero on the bottom (glass plate) and there is zero shear stress along the right, due to symmetry. The fluid flows in or out of the page.

2.3 Results

The flow rate is calculated through Plateau borders of ten different sizes. A curve with a power of 4 is fitted to these values as shown in 9.

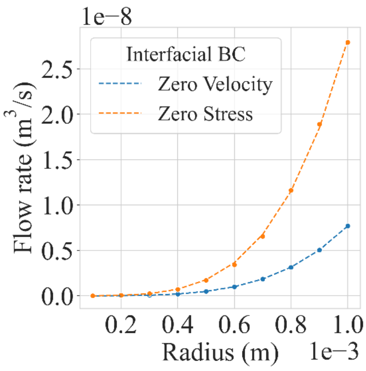


Figure 9: The flow rate values calculated for plateau borders (consisting of 4 times the shapes in Fig. 9) with a radius ranging from 0.1 to 1 mm and a pressure of 1 Pa.

Equations 4 & 5 describe the curves that are fitted to the data in Fig. 9.

$$Q_{zero-velocity} \approx 7.689 \cdot 10^{-3} \frac{\Delta P R_{PB}^4}{\mu L} \tag{4}$$

$$Q_{zero-stress} \approx 2.816 \cdot 10^{-2} \frac{\Delta P R_{PB}^4}{\mu L} \tag{5}$$

At a late stage during this research we were made aware of equations for flow through Plateau borders derived by Drenckhan et al. (2007). The flow rate is slightly smaller for a Plateau border with a zero-velocity BC by a factor of 0.949, and greater for a Plateau border with a zero-stress BC by a factor of 1.939. Our results for the pressure equalization duration are shown in Fig. 10 & 11. Using the equations of Drenckhan et al. (2007) the pressure equalization duration of the networks with the zero-velocity BC would take slightly more time, but still satisfy the assumption of Li et al. (2021) that the equalibration is rapid.

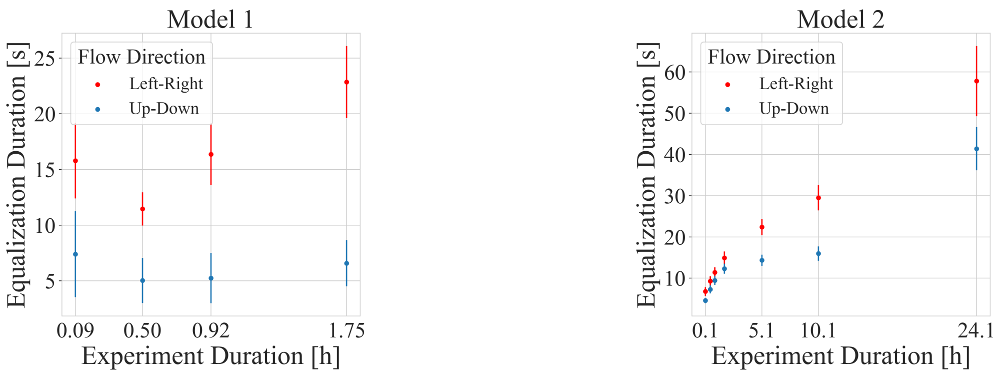


Figure 10: The pressure equalization duration for both models with a zero-velocity interfacial BC.

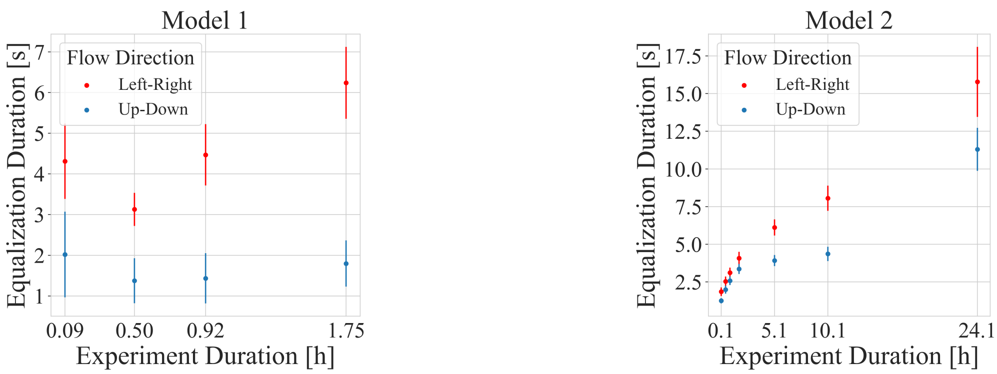


Figure 11: The pressure equalization duration for both models with a zero-stress interfacial BC.

2.4 Discussion

In this topic we have looked at the water flow conductivity through the static foam in the artificial fracture models of Li et al. (2021). The experiments took approximately 24 hours, and 8 measurements were taken for each experiment in total. The flow rate is calculated using COMSOL Multiphysics® through the images taken from above of the foam in the fractures at different durations of the experiment. Li et al. (2021) assumed that the water zones have essentially no resistance to flow compared to other parts of the network.

Because the images are two-dimensional, the flow rate through them is calculated as if it is through a network of slits. A method was found to scale down this flow rate to that of a network of Plateau borders. The dimensions of the Plateau borders are found using the capillary pressure values obtained by Li et al. (2021). But it is not possible to determine a capillary pressure value for every image in Model 1 as the water zones become too small as the experiment progressed. The flow rate through a Plateau border of a fluid with a 0° contact angle as a function of the radius is found. The assumption of zero contact angle may introduce an error into the result, but this assumption is made because the contact angle is unknown, and water is strongly wetting in a foam.

After scaling down the flow rate to that of a network consisting of Plateau borders, the time required for 10% of the total liquid volume to flow across the image under a pressure difference between two sides of the model equal to 10% of the capillary pressure is found. Even a smaller pressure difference such as 1% of the capillary pressure would result in a equalization duration in the order of minutes. This value is sufficiently quick compared to the total duration of the experiment to satisfy the assumption of Li et al. (2021). If there is a zero-stress interfacial boundary condition instead, as there is not according to Hirasaki (2022) for the C14-16 AOS surfactant that is used in the experiments of Li et al. (2021), then the pressure equalizes approximately 3.7 times faster (Fig. 9). The flow rate is higher in the up-down direction than in the left-right in both models, because of the asymmetry of the model.

2.5 Conclusion

The results show that water is able to redistribute itself through the plateau borders to equalize capillary pressure across the image area in the order of tens of seconds. This is multiple orders of magnitude quicker than the duration of the experiments. So the capillary pressure can be assumed to be constant across the images, as assumed by Li et al. (2021).

3 Microfluidic Device Topic

3.1 Introduction

To learn about the flow through porous media, two-dimensional microfluidic networks are used. However, because these networks are two-dimensional and simplified, they may be incapable of fully representing the processes at work in three-dimensional geological porous media. However, conclusions drawn from two-dimensional networks could serve as the foundation for a better understanding of behaviour in three-dimensional networks.

The goal of this work is to determine the conditions that allow for stable simultaneous two-phase flow in a microfluidic network (Cox et al., 2022). Stability in this context means that the flow paths do not change over time. This is known to be possible in the 3D pore networks of geological porous media (Sahimi, 2014). Wetting (WP) and non-wetting (NWP) phases coexist in microfluidic networks. A phase can flow within the network space it occupies without changing the phase distribution. In a two-dimensional network, the possibility of stable flow is not guaranteed, because percolation theory shows that only one phase can flow through a 2D isotropic network at a time (Fisher, 1961). It is possible, however, with the presence of WP bridges across the top and bottom of a constriction in the network, where WP and NWP cross in the same pore throat; see Fig. 12c.

These WP bridges are able to form only under certain conditions (Cox et al., 2022). Earlier studies of two-phase flow in these networks (Hadjisotiriou, 2020, Holstvoogd, 2020, Obbens, 2020) found that WP could flow at only a small fraction of the rate of NWP. Therefore, where possible, we make assumptions that favour the flow of WP relative to NWP, in order to be sure of this conclusion. The simulated microfluidic device must be created in a way that allows WP bridges to form without WP re-invading the throat to block the flow of NWP (Cox et al., 2022). We assume this is the case. This is one of the conditions that we set to be favorable to the flow of WP. The flow rates through those elements are calculated using the COMSOL Multiphysics® Microfluidics Module (COMSOL, 2020). A relative permeability for each phase can be calculated after determining the distribution of phases within the microfluidic device.

The shapes were created in Surface Evolver, which is a program that minimizes the surface energy of an interface (Brakke, 1992). These shapes, which are the basis for the shapes we assume in the network, are shown in Fig. 12. These shapes show what space could be occupied by the WP to form a path around a pore body or throat occupied by NWP; the third image shows WP forming a bridge across an NWP-occupied throat. Because the radius and distance of the pillars is constant, these shapes are determined at differing capillary pressures. We assume for simplicity that there is one shape for each kind of connection in the WP flow path. This simplification is done because otherwise a great number of different interfaces have to be determined for the different combinations of pillar sizes. More WP could be added around the pillar to benefit its flow more, but these shapes have been chosen because the width of the corner flow around the pillar is half the distance between two pillars. This way the WP occupies as much volume as possible without connecting to the water around adjacent pillars where there is no bridge, e.g. pillars to the upper left and bottom right of Fig. 12a. To create a network out of these individual interfaces they have to align, and be able to fit together in every possible arrangement in the network. This would be impossible with the interface shapes solved for isolated pillars using the Surface Evolver (Cox et al., 2022). So equivalent modular shapes are created that are similar in shape and have a similar flow rate for the WP.

If pillars in a square lattice are represented by vertices and the pore throats by edges, then the individual interface shapes can be distributed in a network according to invasion percolation using bond-percolation theory (Sahimi, 2014). We assume that each pore throat is independently open (NWP can flow) with probability $p \in [0, 1]$ and closed (not invaded by NWP) with probability $q = 1 - p$. Thus the wider throats have p values closer to 1. This also implies that the throats can be ordered according to their capillary entry pressure, with larger values of p representing smaller capillary entry pressure. This probability value is used instead of explicitly representing the pore-throat widths because, for simplicity, the WP interface shapes have been determined for one pillar radius and gap width. This implies (for the purpose of flow calculations, after the phase distribution is set) that all the pillars have the same distance between them in the microfluidic device. If this assumption were used during invasion percolation, it would cause the NWP to invade all pore throats at the same capillary-pressure value. That would result in a uniform distribution of the WP throughout the microfluidic device.

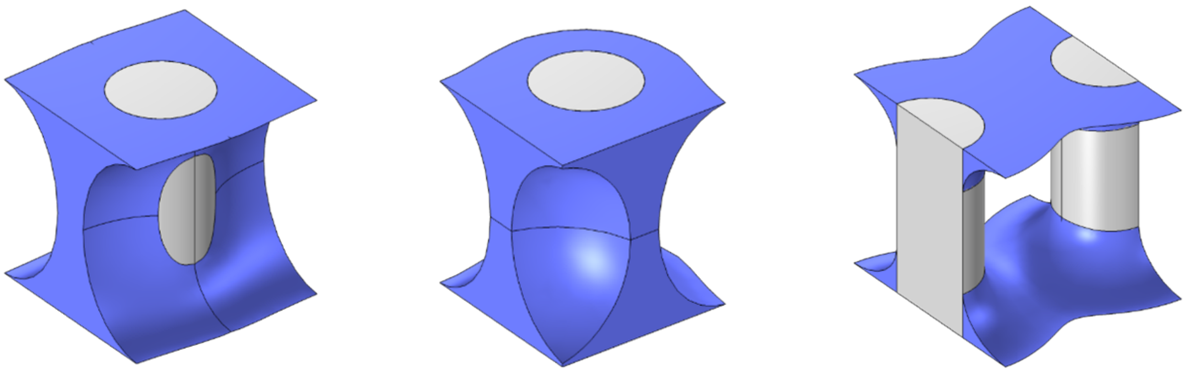


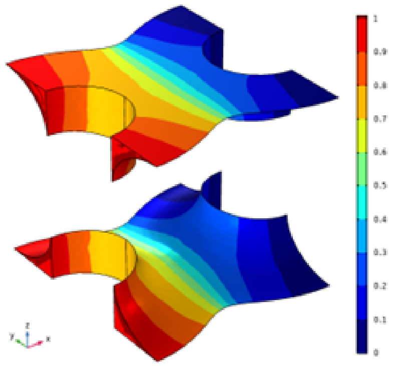
Figure 12: Interface shapes created with Surface Evolver for WP to flow around pore throats or pore bodies occupied by NWP. The grey volume represents cylindrical pillars (grains) in the microfluidic device. The blue volume is occupied by the WP, and the NWP occupies the remaining space.

Networks differ according to the lattice size and random distribution of these probability values. In percolation theory a percolation threshold $p_{threshold}$ is found: infinite networks for which $p > p_{threshold}$ will have infinite connected sub-networks occupied by WP and those with $p < p_{threshold}$ do not. The opposite is true for the NWP. So when two phases are injected into an infinitely large two-dimensional network, both phases would not be able to flow without alternating pore occupancy: opening a pore throat for one phase closes it off for the other phase. The discontinuous phase will build up pressure as it is injected until it is able to enter pores occupied by the continuous phase to connect. So unless it is possible for both phases to flow through a pore throat simultaneously, steady-state two-phase flow is impossible. For simultaneous two-phase flow to be possible without fluctuating pore occupancy, a stable bridge across the throat must exist without WP flooding back into the throat and blocking access to the NWP. Both phases are able to flow through a pore throat if the pore throats are concave (Cox et al., 2022). By making the grains cylindrical in the microfluidic device, bridges are able to form in every pore throat for the largest range of capillary pressure. This benefits the flow of the WP relative to the NWP, because bridges have the possibility to open-up new WP flow paths while reducing the gap between grains for the NWP to flow through.

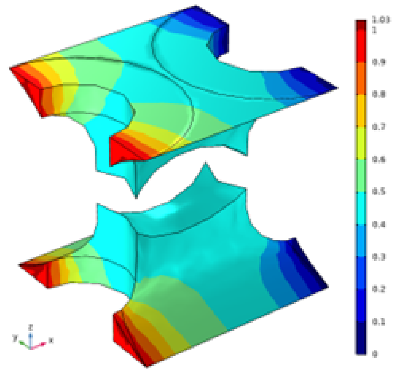
Because in our method the flow through the networks is calculated numerically, modelling infinitely large networks is impossible. To better approximate an infinite bond model, a periodic, or wrap-around, boundary condition is used where the flow paths must connect-up at opposite sides of the model. This results still in a different value for the percolation threshold than for an infinitely large network. But as the size of the network increases it approaches the value for an infinite network. The network also differs from the standard bond-percolation theory because of the addition of bridges that allow both phases to flow simultaneously without alternating pore throat occupancy.

3.2 Method

The geometries created with Surface Evolver cannot be seamlessly integrated into a network. To address this issue, modular geometries with dimensions similar to those of the Surface Evolver geometries have been developed (shown in Appendix A). These are similar to the original except for the geometry created for the bridge shown in Fig. 12c. So comparison between two geometries of the NWP as it flows through the gap in a bridge occupied pore throat is made in Fig. 14 & 15. Instead of comparing our NWP geometry with that of the one obtained with Surface Evolver, it is compared to a gap with the dimensions that would physically be the most constricting to NWP flow possible without snap-off occurring in the pore throat. This seems to happen when the liquid films around the bottom and top of the pillar reach to the middle of the height of the pillar (Cox et al., 2022). The result of using this different geometry for the bridge is shown in Appendix A in the flow rates calculated for the constrictions the WP and NWP experience as they flow through a pore throat with a bridge with the adjustment we make to the Surface Evolver shapes. These values show that the flow of WP is greatly increased and NWP is decreased. If the original bridge geometry is used, it would be the main factor determining the flow rate through the WP flow paths in the network: it would be the largest resistance to flow, and every WP flow path includes at least one bridge. But because the geometry is modified in our calculations it results in the pore throat with the bridge having a similar resistance to flow to the WP as a WP-occupied pore throat thus no longer being the largest resistance to flow in a WP flow path.

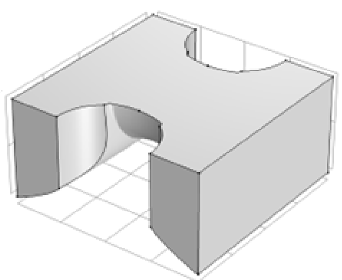


(a) Surface Evolver

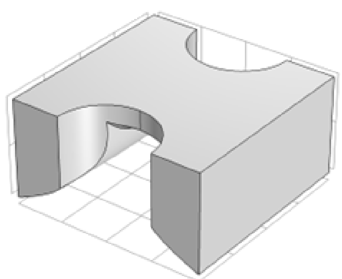


(b) Modified

Figure 13: Pressure profiles for (a) WP flow through the bridge occupied pore throat for the geometry determined with Surface Evolver and (b) the geometry used in our network calculations. The largest pressure drop in the geometry determined with Surface Evolver is located in the middle of the pore throat. In the modified geometry, the largest pressure drop is located around the pillars; the resistance to flow of the WP through this shape is similar to that of a WP filled pore throat.

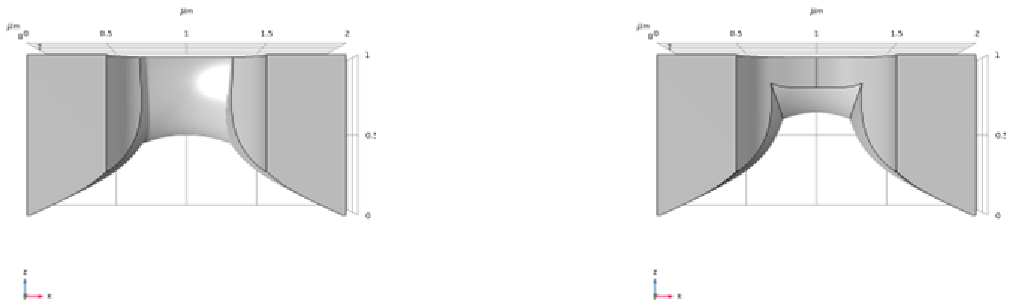


(a) NWP occupied space if the gap is circular.



(b) NWP occupied space of Fig. 13b.

Figure 14: Lower half of the space occupied by the NWP as it flows through the gap in a pore throat with a WP bridge. The lower half is shown because it is symmetrical and details would be obscured otherwise, i.e, where the NWP channel is cylindrical. (a) Geometry with the smallest possible gap for the NWP to flow through without snapping off. (b) The geometry assumed in the flow calculations.



(a) NWP occupied space if the gap is circular. (b) NWP occupied space of Fig. 13b.

Figure 15: Side view of the geometries shown in Fig. 14.

The arrangement of the network is achieved with the following steps. First a matrix is initialized, illustrated in Fig. 16. To obtain the arrangement of pillars in a square grid; pillars are assigned to the intersections of the even rows and columns. Then for each network realization, on the positions of the pore throats which are located between pillars, each pore throat is randomly assigned a value $p \in [0, 1]$. This value is used to determine the ranking, which determines in which order the pore throats are invaded. Another value, which we call $p_{network}$, is used to find distributions of the phases in a network. $p_{network}$ starts with a value of 1 and is then decreased with intervals of 0.01, reflecting invasion of narrower throats as capillary pressure rises. A pore throat and its two adjacent pore bodies are filled with NWP if $p_{network} < p$. So as $p_{network}$ is reduced, the proportion of NWP to WP increases in the network. The value of $p_{network}$ is reduced until a threshold is found where enough pore throats and bodies are filled with NWP for continuous NWP flow paths in both directions across the network. If a path does not exist from a given NWP-filled pore throat or pore body to the side of the network, then it could not be reached during drainage (called an "isolated cluster" in percolation theory); we remove NWP from those throats and bodies.

0.01		0.60		0.87		0.94	
	0.98		0.16		0.79		0.53
0.22		0.98		0.95		0.41	
	0.70		0.86		0.44		0.73
0.71		0.20		0.96		0.17	
	0.97		0.78		0.04		0.60
0.55		0.43		0.39		0.07	
	0.97		0.25		0.01		0.17

Figure 16: Example of an arrangement of p values in a square lattice of coordination number 4. The grey squares are grains and blue is the pore space initially occupied by WP. The pore throats are adjacent to the grains and are assigned a p value. The pore bodies are the blue squares without a p value adjacent to the pore throats.

At the threshold for flow of the NWP, there are no continuous flow paths for the WP. Continuous flow paths for both phases are established by introducing bridges in the network where both phases can flow across a pore throat. The locations of these bridges in the network are decided by increasing the value of $p_{network}$ with intervals of 0.01 starting at $p_{threshold}$. A pore throat is assigned a bridge if $p_{threshold} < p \leq p_{network}$. The value of $p_{network}$ is increased until there are continuous flow paths for both phases. Finally, the bridges that are placed in pores adjacent to WP-filled pore bodies in this process are replaced by WP filled pore throats. Figure 17 shows the sequence of steps in setting up the network. This sequence stops when both phases can flow across the network. Below we do additional calculations with the value of $p_{network}$ raised further.

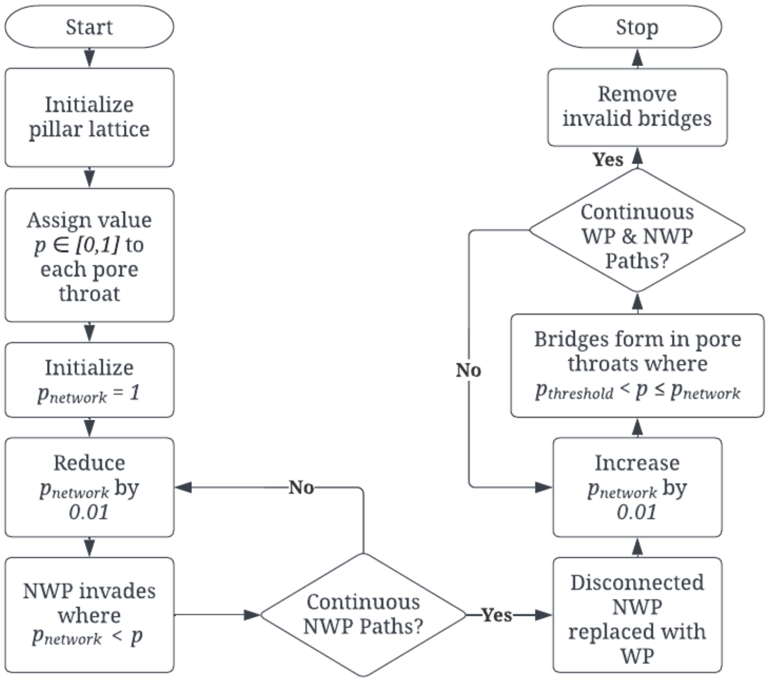


Figure 17: Flow chart of the network arrangement procedure.

A schematic showing the process of draining and subsequent forming of bridges in a network is shown in Fig. 18. At first $p_{network} = 1.00$ and the pore space is fully saturated with WP. Then as $p_{network}$ decreases, pore throats open up where $p_{network} < p$ and NWP is able to move into pore bodies. The next image shows the network at $p_{network} = 0.75$. The NWP has invaded part of the microfluidic device here but has not been able to create a path across the network. At $p_{network} = 0.53$ the first path for the NWP is formed in the top-bottom direction, and the first left-right path is formed at $p_{network} = 0.52$. This is the first value of $p_{network}$ where NWP flow is possible across the network in both directions and satisfying the wrap-around boundary conditions. The value could be lowered further to find more NWP flow paths, but this is not done, because that would benefit the relative permeability of the NWP and be detrimental to the relative permeability of the WP, as fewer pore throats will be occupied by WP. So this is the lowest value that $p_{network}$ becomes.

At this point WP cannot flow; it accumulates, P_c falls, and WP may form bridges in the narrowest throats previously invaded by NWP. We model this by raising the value of $p_{network}$ and assigning bridges to those throats occupied by NWP with $p \leq p_{network}$. As $p_{network}$ increases, the pore throats that were the last to be invaded by NWP will be the first to form bridges (Cox et al., 2022). At $p_{network} = 0.53$ the first vertical flow path for the WP is formed, and at $p_{network} = 0.55$ the first horizontal path. At this value of $p_{network}$ both phases are able to flow across the network in both directions and satisfy the wrap-around boundary condition. (We have assumed in these calculations that no throats are blocked by WP again for the given value of $p_{network}$.) Thus the relative permeabilities can be calculated. This is the minimum number of bridges that are necessary. More bridges can be added by increasing $p_{network}$ more, but it is unclear how much it could increase before the bridges with the lowest $p_{network}$ value snap-off instead (Cox et al., 2022). So besides calculating the relative permeabilities of networks with the minimum amount of bridges necessary, simulations are also done for networks at differing values of $\Delta p_{network}$ above the first value of $p_{network}$ where all conditions for NWP flow are satisfied.

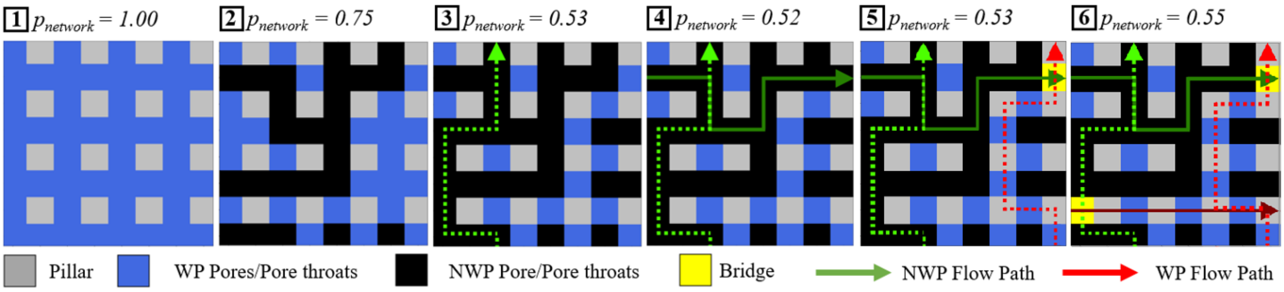


Figure 18: Schematic showing how the distribution of the interfaces is determined for a network with the p-value distribution of Fig. 16. Each pore throat is assigned a random value: $0 < p < 1$. (1) The device is saturated by WP. (2 & 3) NWP invades pores when $p_{network} < p$. (4) NWP flows from left→right & bottom→top, $p_{network}$ stops decreasing. (5) Bridges form as $p_{network}$ increases. (6) If WP flows from left→right & top→bottom; arrangement of pillars and liquid interfaces can be used in the next step.

Now that the interface shapes and arrangement in a network are decided. the network can be assembled in COMSOL Multiphysics® for the flow calculations. Manual arrangement of all the interface shapes would take a sizeable amount of time for each network. So instead a program was written that automates it. The code is attached in Appendix B. The program arranges the interface shapes, assigns the boundary conditions and computes the steady-state flow through the space each phase occupies within a network. An example of a network with a lattice size of 8x8 is shown in Fig. 19.

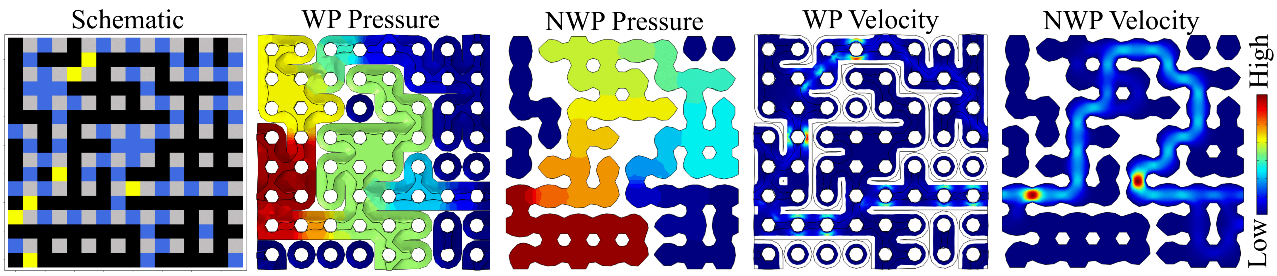


Figure 19: Pressure and velocity plots for left-right flow in a lattice of 8x8 pillars.

A phase's relative permeability is a dimensionless measure of its effective permeability. It is the ratio of that phase's effective permeability to the network's absolute permeability. The absolute permeability is the permeability if the porous medium is saturated with a single phase. Both phases are given the same viscosity in converting from flow rates to relative permeabilities. Equation 6 shows the relationship. The viscosity values and pressure-drop values are the same so these cancel out. The relative permeability is calculated by taking the sum of the surface integrals of the flow velocity over all the outlets of the network.

$$k_{ri} = \frac{k_i}{k} = \frac{Q_i \mu_i / P_i}{Q\mu/P} = \frac{Q_i}{Q} \tag{6}$$

3.3 Results

Bridges would be able to form within the pore throats over a range of values of capillary pressures (Cox et al., 2022). But it is unknown whether this range of capillary pressure is large enough to sustain the necessary number of bridges for a flow path for the WP. For every calculation of the WP and NWP flow through a network it is assumed that the given number of bridges can form to connect the network without any of them snapping off and blocking off the NWP from flowing. While we do not solve explicitly for the capillary-pressure range, it would be related to the p and $p_{network}$ values. Here $\Delta p_{network}$ means the difference between $p_{network}$ and $p_{threshold}$. Thus if the difference between the values of $p_{network}$ for the NWP flow paths to form and for the

WP flow paths is small enough, it is likely that enough WP bridges are stable enough for paths to form. The minimum value of allowing connection of the WP has been found numerically by simulating 100 networks per lattice size, and the results are shown in Fig. 20. As the lattice size increases, the median values of $p_{network}$ for NWP and WP flow converges toward a value, with the difference between the threshold of both phases being around 0.05. So the fraction of throats both invaded by NWP and able to maintain a stable bridge must be approximately 5% to allow any flow at all of WP.

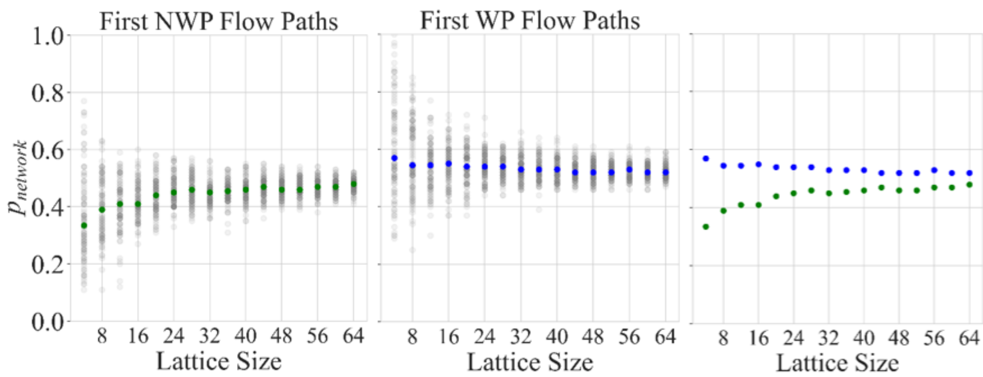


Figure 20: The results of determining the median $p_{network}$ values for when the NWP and WP are able to form continuous flow paths across a network. For a lattice size of 64 the median $p_{network}$ values for continuous flow paths are 0.48 for the NWP and 0.52 for the WP.

The relative permeability values are calculated for different fractions of bridges in the network. The results shown in Fig. 21 show the relative permeabilities at the lowest $\Delta p_{network}$ value that allows the WP to form paths across the network. This value varies for different network arrangements. These results show that the relative permeabilities of both phases decrease as the lattice size is increased. This might be because the flow paths become more tortuous as the network size is increased and the value of $p_{network}$ approaches 0.5, the theoretical value for infinite networks. The ratio of relative permeabilities appears to be unchanging as network size increases, with the relative permeability of the WP approximately a factor 10 to 50 times smaller than that of the NWP.

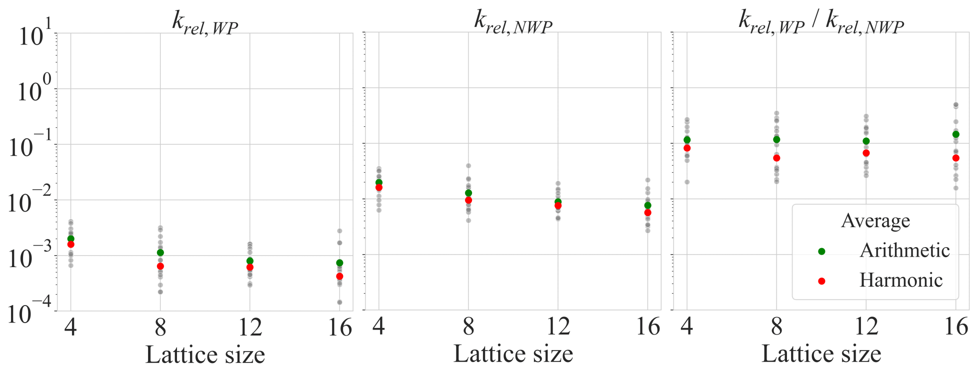


Figure 21: Average relative permeabilities calculated for networks differing in lattice size and the least amount of bridges necessary for flow paths to form in both directions across the network. 20 calculations are done for each lattice size.

The previous results show what the relative permeabilities would be at the minimum number of bridges. By placing more bridges in the networks, the WP has more possible flow paths, while also restricting the flow of the NWP. This has been investigated for networks with a lattice size of 12x12, to see how large the influence of a greater amount of bridges is on the relative permeabilities. For differing values of $\Delta p_{network}$ the flow is calculated through 20 different network arrangements, as shown in Fig. 22. As $\Delta p_{network}$ increases the relative permeability of the WP increases and decreases for the NWP. This results in the relative permeabilities being similar when the $\Delta p_{network}$ is 0.3.

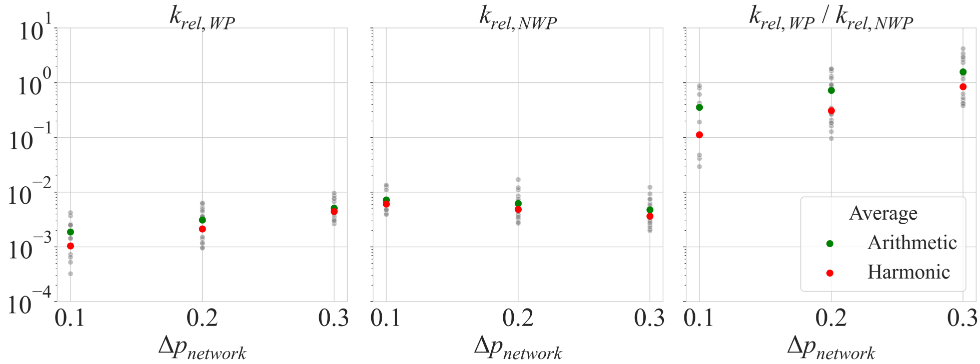


Figure 22: Average relative permeabilities with differing amounts of bridges in networks of lattice size 12x12.

3.4 Discussion

In this topic an estimate of the relative permeabilities for steady-state two-phase flow through a microfluidic device is made, one in which various assumptions favor the flow of the wetting phase. Simultaneous steady-state two-phase flow in an infinite 2D random network is possible only if the two phases are able to cross through a pore throat. This is possible with a bridge interface shape in some pore throats. The simulated microfluidic device consists of a square lattice of grains in the shape of round pillars. The fluid distribution around the individual pillars was determined with Surface Evolver. The simplification there is that the pillars have the same radius, which means that the capillary pressure is different in the individual interface shapes.

The flow of WP is constricted greatly by being split up around every pillar it flows around to get around an NWP-filled pore body (see Fig. 12a & 12b). Flow through these corner regions around the pillars would be the largest resistance to flow for the WP in a network. Therefore we modified the interface shape to be less restricting to the WP flow. This shape in turn is more restrictive to NWP flow than physically possible because shape the of the interface assumed for the flow calculations would result in snap-off. This has been shown to happen when the gap reaches a circular shape and the WP from the top and the bottom of the pillars meets (Cox et al., 2022).

In a network with uniform capillary pressure and differing pillar radii, the bridges would range in resistance to the WP flow. So if some pore throats could maintain such a liquid interface, it would only be for a fraction of the bridges throughout the network. These changes would benefit the relative permeability fraction in favor of the WP.

Other assumptions that benefit the flow of WP in our calculations are as follows: The WP is given a no-stress boundary condition on the interfacial boundary. For the NWP the maximum value of $p_{network}$ that connects both sides of the microfluidic device is chosen; allowing for more NWP flow paths would reduce the conductivity of the WP.

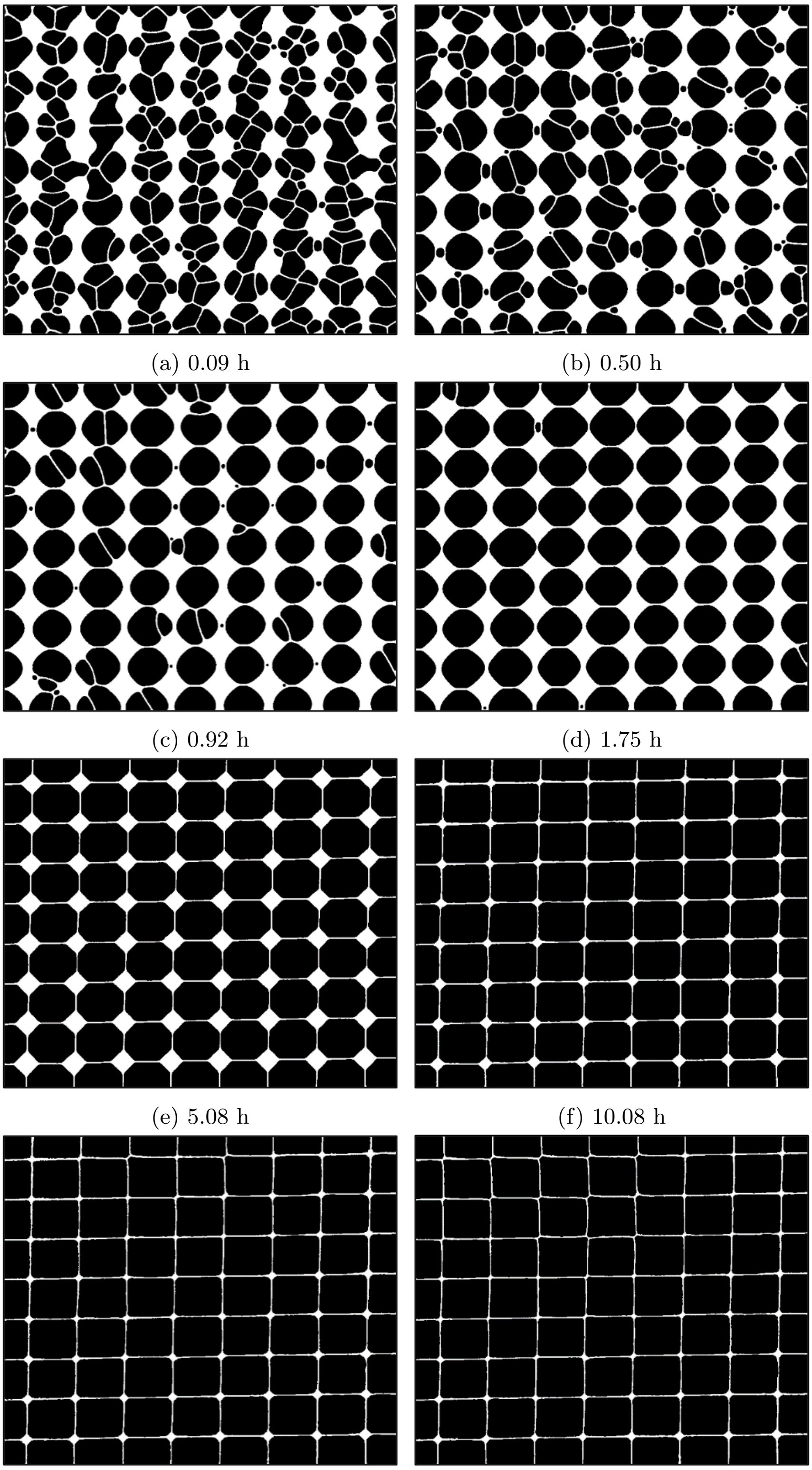
The limited size of the simulated microfluidic devices means that results for infinite networks may differ by a modest amount. Practical limitations in computer power prevented us from studying larger networks. Using periodic boundary conditions gives a somewhat better representation of an infinite network.

3.5 Conclusion

With all these benefits to the flow of the WP, it could perhaps be possible for two phases with similar viscosity values such as water and oil to maintain steady-state two-phase flow with comparable fractional flows imposed. If the NWP is gaseous instead and the viscosity is much smaller than that of the WP, the relative permeabilities would differ greatly (e.g., by a factor of 50 or 70). As a result, the wetting phase could not maintain a fractional flow greater than a small fraction of that of the NWP without a build-up in the WP, as it is unable to flow out of the microfluidic device quickly enough, accumulating WP in the network and forcing snap-off and fluctuating pore occupancy. This is exacerbated if there is a no-velocity boundary condition for the WP on the interfacial boundary, as is the case for surfactants with large surface viscosity. This fluctuation in pore occupancy does not occur in three-dimensional porous media and unless it is in the high-capillary-number flow regime. So microfluidic devices would be problematic for studying snap-off in gas-water flow in three-dimensional porous media.

A Appendix

Images of Model 1



(a) 0.09 h (b) 0.50 h

(c) 0.92 h (d) 1.75 h

(e) 5.08 h (f) 10.08 h

(g) 17.58 h (h) 24.08 h

Figure 23: Images of Model 1 Li et al. (2021).

Table of Model 1

Table 1: Calculations for Model 1. These values are calculated with a zero velocity BC on the interfacial boundary of the Plateau Borders. The results differ by a factor of 3.66 if a zero-stress BC is assigned. $Q_{network,2D}$ is the two-dimensional flow rate through the geometry created with the images of Model 1 with a pressure difference that is 10% of the capillary pressure. Q_{slit} is the flow rate of water at a temperature of 293.15K through a single slit with a width of 2 times B_{slit}, a length of 1 mm, the hydraulic aperture as the height, and a pressure difference of 1 Pa. Q_{PB} is the flow rate of water at a temperature of 293.15K through a Plateau border with a radius of R_{PB}, a length of 1 mm, and a pressure difference of 1 Pa. $Q_{network,slit}$ is $Q_{network,2D}$ multiplied with the hydraulic aperture. $Q_{network,slit}$ is then scaled down using the Scaling Factor to obtain $Q_{network,PB}$, which is the flow through a network of Plateau borders. The Equalization Time is obtained by dividing the Total Volume (of water in the image) by $Q_{network,PB}$.

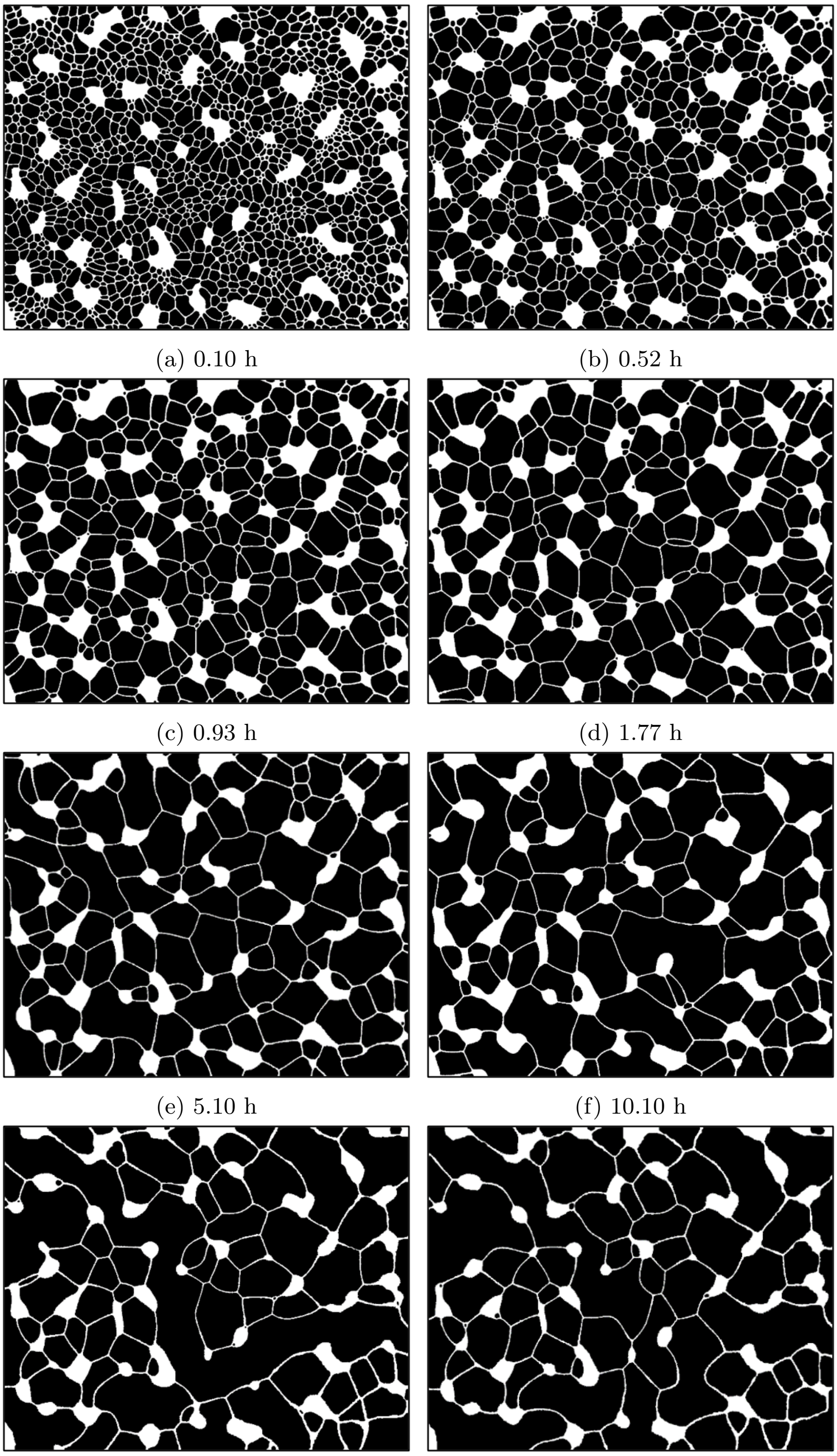
Time $[h]$	P_c $[Pa]$	Total Volume $[m^3]$	B_{slit} left-right $[m]$	StDev	B_{slit} up-down $[m]$	StDev
0.09	2010	6.23E-11	5.73E-05	1.23E-05	7.29E-05	3.80E-05
0.5	2210	4.30E-11	5.56E-05	7.23E-06	6.39E-05	2.58E-05
0.92	2460	3.83E-11	5.93E-05	1.00E-05	8.23E-05	3.54E-05
1.75	3370	1.20E-11	6.36E-05	9.01E-06	8.21E-05	2.59E-05

$Q_{network,2D}$ left-right $[m^2/s]$	$Q_{network,2D}$ up-down $[m^2/s]$	Q_{slit} left-right $[m^3/s]$	Q_{slit} up-down $[m^3/s]$	R_{PB} $[m]$	Q_{PB} $[m]$
1.22E-05	5.37E-05	7.20E-13	1.48E-12	1.60E-05	5.06E-16
1.55E-05	5.37E-05	6.58E-13	1.00E-12	1.46E-05	3.46E-16
1.80E-05	1.50E-04	7.99E-13	2.14E-12	1.31E-05	2.26E-16
1.76E-05	1.31E-04	9.88E-13	2.12E-12	9.55E-06	6.41E-17

Scaling Factor left-right $[m^3/s]$	Scaling Factor up-down $[m^3/s]$	$Q_{network,slit}$ left-right $[m^3/s]$	$Q_{network,slit}$ up-down $[m^3/s]$	$Q_{network,PB}$ left-right $[m^3/s]$	$Q_{network,PB}$ up-down $[m^3/s]$
7.03E-04	3.41E-04	5.61E-10	2.47E-09	3.95E-13	8.43E-13
5.26E-04	3.46E-04	7.13E-10	2.47E-09	3.75E-13	8.54E-13
2.83E-04	1.06E-04	8.28E-10	6.90E-09	2.34E-13	7.29E-13
6.49E-05	3.02E-05	8.10E-10	6.03E-09	5.25E-14	1.82E-13

Equalization Time left-right $[s]$	Equalization Time up-down $[s]$	Equalization Time StDev left-right	Equalization Time StDev up-down
1.58E+01	7.39E+00	3.39E+00	3.85E+00
1.15E+01	5.03E+00	1.49E+00	2.03E+00
1.64E+01	5.25E+00	2.76E+00	2.26E+00
2.29E+01	6.59E+00	3.24E+00	2.08E+00

Images of Model 2



(a) 0.10 h (b) 0.52 h (c) 0.93 h (d) 1.77 h (e) 5.10 h (f) 10.10 h (g) 17.10 h (h) 24.10 h

Figure 24: Images of Model 2 Li et al. (2021).

Table of Model 2

Table 2: Calculations for Model 2. These values are calculated with a zero-velocity BC on the interfacial boundary of the Plateau Borders. The results differ by a factor of 3.66 if a zero-stress BC is assigned. $Q_{network,2D}$ is the two-dimensional flow rate through the geometry created with the images of Model 1 with a pressure difference that is 10% of the capillary pressure. Q_{slit} is the flow rate of water at a temperature of 293.15K through a single slit with a width of 2 times B_{slit}, a length of 1 mm, the hydraulic aperture as the height, and a pressure difference of 1 Pa. Q_{PB} is the flow rate of water at a temperature of 293.15K through a Plateau border with a radius of R_{PB}, a length of 1 mm, and a pressure difference of 1 Pa. $Q_{network,slit}$ is $Q_{network,2D}$ multiplied with the hydraulic aperture. $Q_{network,slit}$ is then scaled down using the Scaling Factor to obtain $Q_{network,PB}$, which is the flow through a network of Plateau borders. The Equalization Time is obtained by dividing the Total Volume (of water in the image) by $Q_{network,PB}$.

Time $[h]$	P_c $[Pa]$	Total Volume $[m^3]$	B_{slit} left-right $[m]$	StDev	B_{slit} up-down $[m]$	StDev
0.1	840	1.34E-09	5.04E-05	7.84E-06	5.01E-05	6.99E-06
0.54	890	1.17E-09	5.39E-05	6.82E-06	5.54E-05	7.39E-06
0.93	900	1.14E-09	5.62E-05	5.86E-06	5.74E-05	6.40E-06
1.77	910	9.94E-10	6.11E-05	6.40E-06	6.30E-05	6.25E-06
5.10	930	8.31E-10	6.54E-05	5.70E-06	6.35E-05	5.99E-06
10.1	960	7.20E-10	6.33E-05	6.59E-06	6.38E-05	6.90E-06
24.1	1030	5.05E-10	7.40E-05	1.09E-05	7.39E-05	9.33E-06

$Q_{network,2D}$ left-right $[m^2/s]$	$Q_{network,2D}$ up-down $[m^2/s]$	Q_{slit} left-right $[m^3/s]$	Q_{slit} up-down $[m^3/s]$	R_{PB} $[m]$	Q_{PB} $[m]$
1.27E-05	1.85E-05	8.32E-13	8.16E-13	3.83E-05	1.66E-14
1.25E-05	1.73E-05	1.02E-12	1.11E-12	3.62E-05	1.32E-14
1.17E-05	1.51E-05	1.15E-12	1.23E-12	3.58E-05	1.26E-14
1.05E-05	1.40E-05	1.48E-12	1.63E-12	3.54E-05	1.21E-14
7.82E-06	1.12E-05	1.82E-12	1.67E-12	3.46E-05	1.10E-14
5.29E-06	1.00E-05	1.64E-12	1.68E-12	3.35E-05	9.73E-15
4.01E-06	5.58E-06	2.63E-12	2.62E-12	3.13E-05	7.34E-15

Scaling Factor left-right $[m^3/s]$	Scaling Factor up-down $[m^3/s]$	$Q_{network,slit}$ left-right $[m^3/s]$	$Q_{network,slit}$ up-down $[m^3/s]$	$Q_{network,PB}$ left-right $[m^3/s]$	$Q_{network,PB}$ up-down $[m^3/s]$
2.00E-02	2.03E-02	9.91E-10	1.44E-09	1.98E-11	2.93E-11
1.29E-02	1.19E-02	9.75E-10	1.35E-09	1.26E-11	1.61E-11
1.09E-02	1.03E-02	9.13E-10	1.18E-09	9.98E-12	1.21E-11
8.13E-03	7.40E-03	8.19E-10	1.09E-09	6.66E-12	8.09E-12
6.08E-03	6.63E-03	6.10E-10	8.74E-10	3.71E-12	5.79E-12
5.92E-03	5.78E-03	4.13E-10	7.80E-10	2.44E-12	4.51E-12
2.79E-03	2.80E-03	3.13E-10	4.35E-10	8.74E-13	1.22E-12

Equalization Time left-right $[s]$	Equalization Time up-down $[s]$	Equalization Time StDev left-right	Equalization Time StDev up-down
6.78E+00	4.57E+00	1.05E+00	6.37E-01
9.29E+00	7.28E+00	1.18E+00	9.71E-01
1.14E+01	9.44E+00	1.19E+00	1.05E+00
1.49E+01	1.23E+01	1.56E+00	1.22E+00
2.24E+01	1.44E+01	1.95E+00	1.35E+00
2.95E+01	1.60E+01	3.07E+00	1.73E+00
5.78E+01	4.14E+01	8.52E+00	5.23E+00

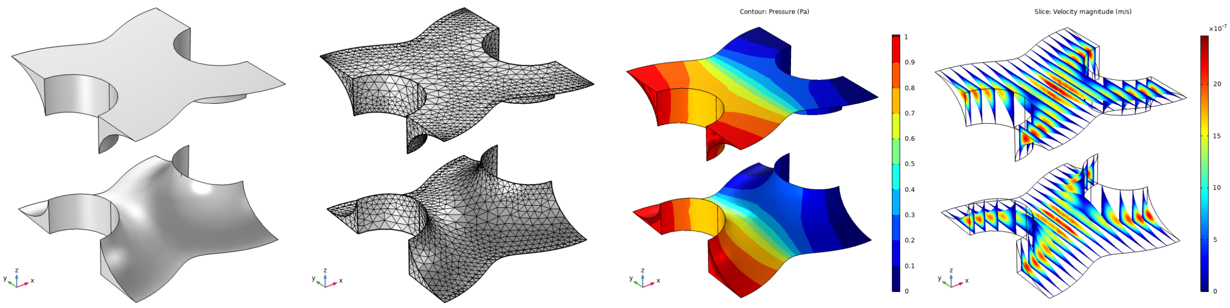
Separate COMSOL Multiphysics® shapes of the microfluidic device topic

All calculations are done using the standard water material in COMSOL Microfluidics® with $T = 293.15\,K$, $P = 1\,atm$, $\mu = 10^{-4}\,Pa \cdot s$, $W = L = H = 2\,\mu m$ and $\Delta P = 1\,Pa$. In the network calculations the pillars have a diameter of $1\,\mu m$ which is the same as the width of the gap in between two pillars. "Cox" in this table refers to the interface configuration produced by Prof. Simon Cox for the given connection, using Surface Evolver.

Table 3: The flow rates calculated for individual geometries, to compare the geometries obtained using Surface Evolver with the geometries used in the networks. In each case, "Cox" refers to the geometry and flow rate calculated through the shape determined by the Surface Evolver, and "Mod." refers to the modified geometry used in the COMSOL Multiphysics® solutions for flow through the network.

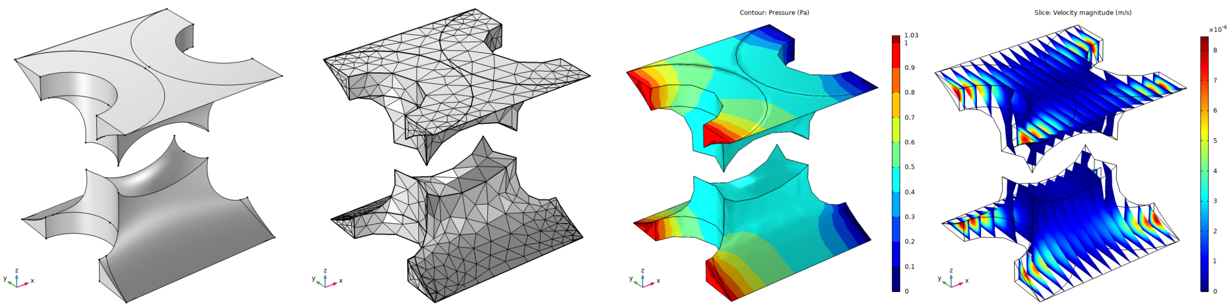
	Cox Bridge	Mod. Bridge	Cox Straight	Mod. Straight	Cox Corner	Mod. Corner
Q $[m^3/s]$	2.80E-19	9.13E-19	7.65E-19	8.78E-19	1.88E-17	1.51E-17

	Round Gap	Modified Gap
Q $[m^3/s]$	1,33E-17	8,52E-18



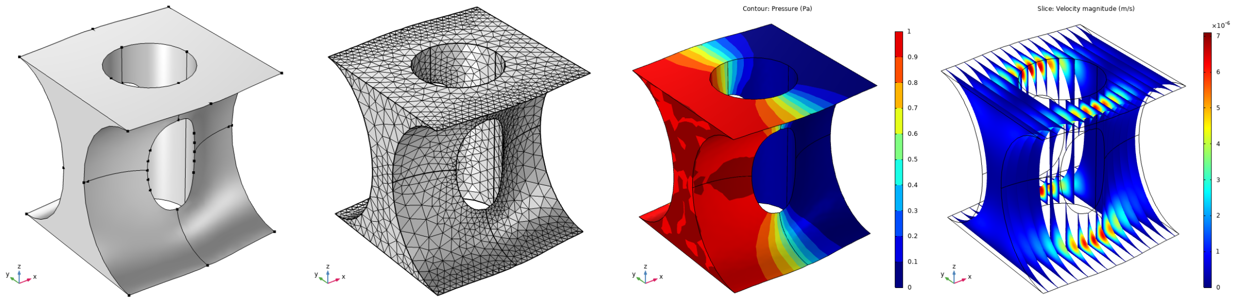
(a) Geometry (b) Mesh (c) Pressure $[Pa]$ (d) Velocity $[m/s^3]$

Figure 25: Cox bridge.



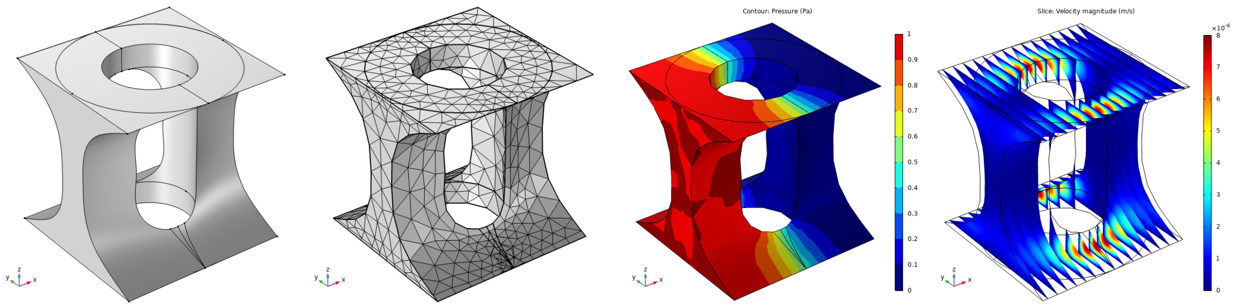
(a) Geometry (b) Mesh (c) Pressure $[Pa]$ (d) Velocity $[m/s^3]$

Figure 26: Modified bridge.



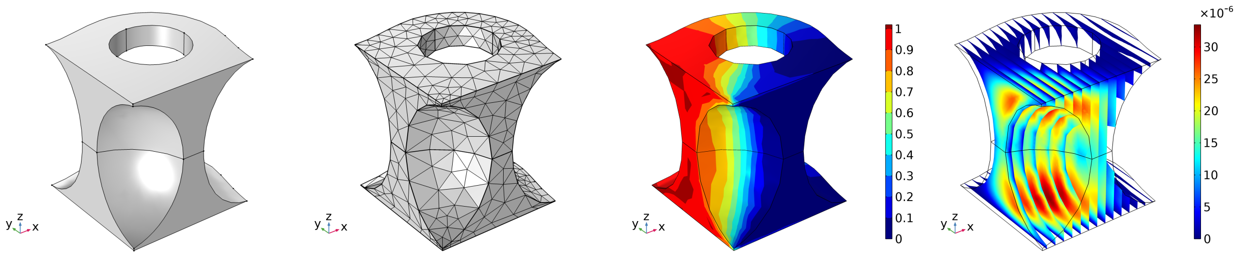
(a) Geometry (b) Mesh (c) Pressure $[Pa]$ (d) Velocity $[m/s^3]$

Figure 27: Cox straight.



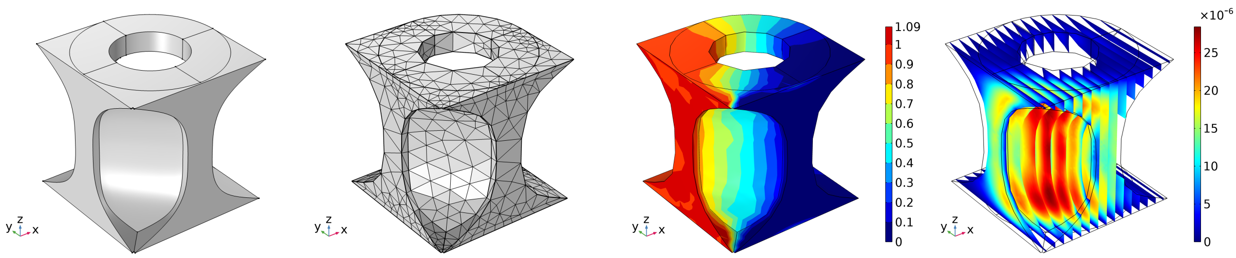
(a) Geometry (b) Mesh (c) Pressure $[Pa]$ (d) Velocity $[m/s^3]$

Figure 28: Modified straight.



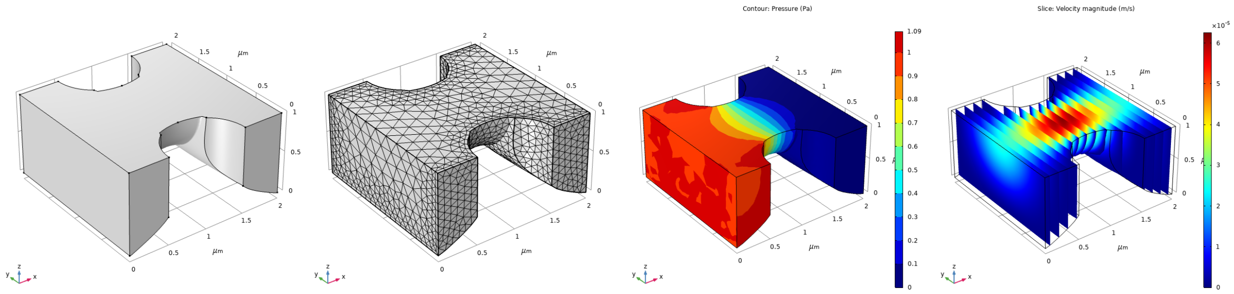
(a) Geometry (b) Mesh (c) Pressure $[Pa]$ (d) Velocity $[m/s^3]$

Figure 29: Cox corner.



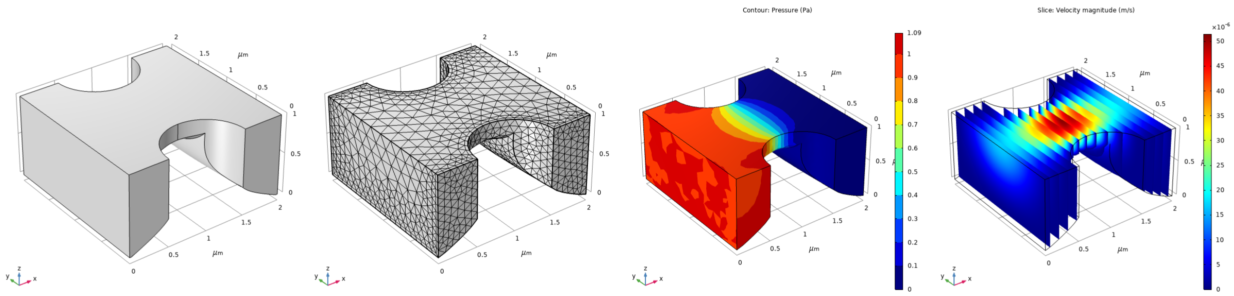
(a) Geometry (b) Mesh (c) Pressure $[Pa]$ (d) Velocity $[m/s^3]$

Figure 30: Modified corner.



(a) Geometry (b) Mesh (c) Pressure $[Pa]$ (d) Velocity $[m/s^3]$

Figure 31: Lower half of a bridge with a circular gap.



(a) (b) (c) Pressure $[Pa]$ (d) Velocity $[m/s^3]$

Figure 32: Lower half of a bridge with the modified gap.

B Appendix

Code: Plateau border flow rate curve fit

This code uses the flow rates obtained in COMSOL Multiphysics® (variables "No_Slip_Flow" and "Slip_Flow") for Plateau borders of varying widths (variable "Radius") to create Fig. 9 and derive Eq. 4 & 5.

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.optimize import curve_fit
from IPython.display import display, Latex
import seaborn as sns
import matplotlib as mpl

mpl.rcParams['figure.dpi']= 375

sns.set_style("whitegrid")
sns.set_style(rc={'ytick.left': True})

mpl.rc('font',family='Times New Roman')
mpl.rcParams['mathtext.fontset'] = 'custom'
mpl.rcParams['mathtext.rm'] = 'Times New Roman'
mpl.rcParams['mathtext.it'] = 'Times New Roman:italic'
mpl.rcParams['mathtext.bf'] = 'Times New Roman:bold'

mpl.rcParams.update({'figure.autolayout': True})

L = 0.001
visc = 0.0010000247767419923

Radius = np.array([0.001,0.0009,0.0008,0.0007,0.0006,0.0005,0.0004,0.0003,0.0002,0.0001])
No_Slip_Flow = np.array([1.9247E-9,1.2599E-9,7.8547E-10,4.5950E-10,2.4757E-10,1.1898E-10,4.8878E-11,1.5553E-11,3.0901E-12,2.2477E-13])
Slip_Flow = np.array([6.9825E-9,4.7269E-9,2.9074E-9,1.6358E-9,8.5499E-10,4.2804E-10,1.7438E-10,5.9057E-11,1.3465E-11,7.2433E-13])

No_Slip_Flow *= 4 # 4 times that flow rate in 1 PB
Slip_Flow *= 4
```

```python

def func(x,a):
    return a*x**4

parameters = curve_fit(func,Radius,No_Slip_Flow)
parameters2 = curve_fit(func,Radius, Slip_Flow)

display(Latex(f'Qnoslip = {parameters[0][0]}' + '$\cdot R^4_{PB}$'))
display(Latex(f'Qslip = {parameters2[0][0]}' + '$\cdot R^4_{PB}$'))

plt.figure(0, figsize =[6,6])

ax = sns.scatterplot(Radius,No_Slip_Flow)

plt.plot(Radius,func(Radius,parameters[0][0]),linestyle='dashed')

sns.scatterplot(Radius,Slip_Flow)

plt.plot(Radius,func(Radius,parameters2[0][0]),linestyle='dashed')

plt.legend(['Zero Velocity','Zero Stress'],title='Interfacial BC',fontsize=24)
plt.rcParams['legend.title_fontsize'] = 24
plt.xlabel('Radius (m)',fontsize=30)
plt.ylabel('Flow rate (m$^3$/s)',fontsize=30)
plt.xticks(size=30)
plt.yticks(size=30)
plt.ticklabel_format(axis="x", style="sci", scilimits=(0,0))
ax.yaxis.offsetText.set_fontsize(30)
ax.xaxis.offsetText.set_fontsize(30)
plt.savefig('PBflow.png')

print(parameters[0][0]*L*visc)
print(parameters2[0][0]*L*visc)
```

Code: Calculations Artificial Fracture Topic

```python
import numpy as np; import pandas as pd; import matplotlib as mpl; import matplotlib.pyplot as plt; import seaborn as sns

pd.set_option("display.max_columns", None)
pd.set_option('display.float_format', '{:.2E}'.format)

mpl.rcParams["figure.dpi"]= 375

sns.set_style("whitegrid")
sns.set_style(rc={"ytick.left": True})

mpl.rc("font",family="Times New Roman")
mpl.rcParams["mathtext.fontset"] = "custom"
mpl.rcParams["mathtext.rm"] = "Times New Roman"
mpl.rcParams["mathtext.it"] = "Times New Roman:italic"
mpl.rcParams["mathtext.bf"] = "Times New Roman:bold"
mpl.rcParams.update({"figure.autolayout": True})

hydraulic_aperture_model1 = 4.6E-5      # [m]
hydraulic_aperture_model2 = 7.8E-5      # [m]
density_fluid             = 998.0585    # [kg/m^3] Standard COMSOL water material at 293.15 K.
viscosity                 = 0.001000248 # [Pa*s] Standard COMSOL water material at 293.15 K.
surface_tension           = 0.0322      # [N/m]
L                         = 0.001       # [m] Length of the slit and Plateau border along the flow direction.
delta_P                   = 1           # [Pa] Pressure difference of inlet and outlet in slit/PB calculation.
volume_fraction           = 0.1         # Fraction of total water volume to flow out of the network.
coefficient_no_slip       = 0.007689340390189774  # Coefficients found for flow through a Plateau border.
coefficient_slip          = 0.02816471179666251
slip_noslip_ratio = coefficient_slip/coefficient_no_slip

data_model1 = {"time":         [0.09,                0.5,        0.92,        1.75], # [hour] Time of image creation since start of experiment.
               "P_c":          [2010,               2210,        2460,        3370], # Capillary Pressure[Pa] obtained from Li et al. (2021).
```

```
                "total␣volume": [6.23E-11,       4.30E-11,    3.83E-11,    1.20E-11], # [m^3] Total water volume in image.
                "width␣H":      [5.72805E-05, 5.55886E-05, 5.92850E-05, 6.36432E-05], # [m] Average lamellae width measured for horizontal flow.
                "width␣H␣StDev":[1.23E-05,       7.23E-06,    1.00E-05,    9.01E-06], # Standard Deviation.
                "width␣V":      [7.28896E-05, 6.39366E-05, 8.22884E-05, 8.20751E-05], # [m] Average lamellae width measured for vertical flow.
                "width␣V␣StDev":[3.80E-05,       2.58E-05,    3.54E-05,    2.59E-05], # Standard Deviation.
                "Q␣network␣H":  [1.22E-05,       1.55E-05,    1.80E-05,    1.76E-05], # [m^2/s] 2D flow rate in horizontal direction.
                "Q␣network␣V":  [5.37E-05,       5.37E-05,    1.50E-04,    1.31E-04]} # [m^2/s] 2D flow rate in vertical direction.


data_model2 = {"time":          [0.1,          0.52,        0.93,        1.77,        5.1,        10.1,        24.1],
                "P_c":           [840,          890,         900,         910,         930,         960,         1030],
                "total␣volume": [1.34E-09,    1.17E-09,    1.14E-09,    9.94E-10,    8.31E-10,    7.20E-10,    5.05E-10],
                "width␣H":      [5.0399E-05,  5.39459E-05, 5.61751E-05, 6.10897E-05, 6.53889E-05, 6.3258E-05,  7.39536E-05],
                "width␣H␣StDev":[7.84E-06,    6.82E-06,    5.86E-06,    6.40E-06,    5.70E-06,    6.59E-06,    1.09E-05],
                "width␣V":      [5.00837E-05, 5.54157E-05, 5.73899E-05, 6.30362E-05, 6.35442E-05, 6.37526E-05, 7.38641E-05],
                "width␣V␣StDev":[6.99E-06,    7.39E-06,    6.40E-06,    6.25E-06,    5.99E-06,    6.90E-06,    9.33E-06],
                "Q␣network␣H":  [1.27E-05,    1.25E-05,    1.17E-05,    1.05E-05,    7.82E-06,    5.29E-06,    4.01E-06],
                "Q␣network␣V":  [1.85E-05,    1.73E-05,    1.51E-05,    1.40E-05,    1.12E-05,    1.00E-05,    5.58E-06]}



```

```python
def Calculations(data, hydraulic_aperture):
    data_df = pd.DataFrame.from_dict(data)

    def Q_slit(slit_width):
        return (2/3)* (delta_P * (slit_width/2) ** 3 * hydraulic_aperture) / (viscosity * L)

    data_df["Width␣error␣H"]           = data_df["width␣H␣StDev"] / data_df["width␣H"]
    data_df["Width␣error␣V"]           = data_df["width␣V␣StDev"] / data_df["width␣V"]

    data_df["Q␣slit␣H"]          = Q_slit(data_df["width␣H"])
    data_df["Q␣slit␣V"]          = Q_slit(data_df["width␣V"])

    data_df["R␣Plateau␣border"] = surface_tension / data_df["P_c"]

    data_df["Q␣Plateau␣border"] = coefficient_no_slip/(viscosity*L) * (data_df["R␣Plateau␣border"] ** 4)

    data_df["Q␣ratio␣H"]         = data_df["Q␣Plateau␣border"] / data_df["Q␣slit␣H"]
    data_df["Q␣ratio␣V"]         = data_df["Q␣Plateau␣border"] / data_df["Q␣slit␣V"]

    data_df["Q␣slit␣network␣H"] = data_df["Q␣network␣H"] * hydraulic_aperture # [m^3/s]
    data_df["Q␣slit␣network␣V"] = data_df["Q␣network␣V"] * hydraulic_aperture

    data_df["Q␣PB␣H"]            = data_df["Q␣slit␣network␣H"] * data_df["Q␣ratio␣H"] # 3D flow corrected from slits to Plateau borders.
    data_df["Q␣PB␣V"]            = data_df["Q␣slit␣network␣V"] * data_df["Q␣ratio␣V"]

    data_df["eq.␣time␣H"]        = (volume_fraction * data_df["total␣volume"]) / data_df["Q␣PB␣H"] # [s] Pressure equalization duration.
    data_df["eq.␣time␣V"]        = (volume_fraction * data_df["total␣volume"]) / data_df["Q␣PB␣V"] # [s] Pressure equalization duration.

    data_df["eq.␣time␣H␣StDev"] = data_df["eq.␣time␣H"] * data_df["Width␣error␣H"]
    data_df["eq.␣time␣V␣StDev"] = data_df["eq.␣time␣V"] * data_df["Width␣error␣V"]
    return data_df

def plot(df, n, ylim, xticks):
```

```

    ax = df.plot(kind   = "scatter",
                         figsize= [6,6],
                         x      = "time",
                         y      = "eq.␣time␣H",
                         yerr   = "eq.␣time␣H␣StDev",
                         xticks = xticks,
                         #ylim   = ylim,
                         color  = "red")

    df.plot(ax     = ax,
            kind   = "scatter",
            x      = "time",
            y      = "eq.␣time␣V",
            yerr   = "eq.␣time␣V␣StDev")

    plt.title(f"Model␣{n}", size = 30)
    plt.xlabel("Experiment␣Duration␣[h]", fontsize = 30)
    plt.ylabel("Equalization␣Duration␣[s]", fontsize = 30)

    plt.xticks(xticks,fontsize=25)
    plt.yticks(fontsize=25)
    plt.legend(["Left-Right","Up-Down"], title = "Flow␣Direction",loc=2,fontsize=20)
    plt.rcParams["legend.title_fontsize"] = 24

    plt.savefig(f"model{n}.png")

data_df1 = Calculations(data_model1, hydraulic_aperture_model1)
data_df2 = Calculations(data_model2, hydraulic_aperture_model2)

plot(data_df1, 1, [0, 0.8], data_df1["time"])
plot(data_df2, 2, [0, 2  ], [0.1, 5.1, 10.1, 24.1])

print(data_df1)
print(data_df2)
```

Code: Functions used for the arrangement of the networks

```
def plot_grid(figcount,title,path):
    mesh = np.arange(len(Matrix))
    x, y = np.meshgrid(mesh, mesh)
    color = [[Matrix[i][j]['color'] for i in range(w)] for j in range(w)]
    color = [item for sublist in color for item in sublist]

    plt.figure(figcount,figsize=[12,12])

    plt.scatter(x,y,marker='s',s=s,color=color)
    plt.title(title)
    plt.savefig(f'Network_images_final/Gridsize_{gridsize_int}/Seed_{seed}/' + path)


def set_water_ini():
    #all bodies initially filled with water
    for x in range(w)[::2]:
        for y in range(h)[::2]:
                Matrix[x][y] = {'throat_size':1,'type':'body','fluid':'water', 'color':water}

def set_throat_size():
    #setting throat size depending on p
    for x in range(w+1)[::2]:
        for y in range(h):
            if (x - 1 -(y %2)) >= 0:
                Matrix[x - 1 - (y %2)][y] = {'throat_size':np.random.rand(),'type':'throat','fluid':
                    'water','color':water}

def set_gas_throats():
    # setting gas throats
    for x in range(w+1)[::2]:
        for y in range(h):
            if (x - 1 -(y %2)) >= 0:
                if (Matrix[x - 1 - (y %2)][y]['type'] == 'throat') and (Matrix[x - 1 - (y %2)][y]['
                    throat_size'] > p):
                    Matrix[x - 1 - (y %2)][y]['fluid'] = 'gas'
```

```
                    Matrix[x - 1 - (y %2)][y]['color'] = gas

def set_gas_bodies():
       #setting porebodies adjacent to gas throats to gas
    for x in range(0,w):
        for y in range(0,h):
            if Matrix[x][y]['fluid'] == 'gas':

                for i in [-1,1]:
                    if 1 < x < w - 1:
                        if Matrix[x+i][y]['type'] == 'body':
                            Matrix[x+i][y]['fluid'] = 'gas'
                            Matrix[x+i][y]['color'] = gas
                    if 1 < y < h - 1:
                        if Matrix[x][y+i]['type'] == 'body':
                            Matrix[x][y+i]['fluid'] = 'gas'
                            Matrix[x][y+i]['color'] = gas

def place_bridges():
    #place bridges
    for x in range(w):
        for y in range(h):
            if p_bridge[0] <= round(Matrix[x][y]['throat_size'],2) <= p_bridge[1]:
                Matrix[x][y]['fluid'] = 'bridge'
                Matrix[x][y]['color'] = bridge

def bfs_opposite_side(grid, start, end, fluid): #end is 'x-direction' or 'y-direction'
    queue = collections.deque([[start]])
    seen = set([start])
    while queue:
        path = queue.popleft()

        x, y = path[-1]

        for x2, y2 in ((x+1,y), (x-1,y), (x,y+1), (x,y-1)):

            if (0 <= x2 < w) and (0 <= y2 < h) and ((x2, y2) not in seen):
```

```
                fluid_condition = False

                if (fluid == 'gas') and (Matrix[x2][y2]['fluid'] == 'gas'):
                    fluid_condition = True

                if (fluid == 'water') and (Matrix[x2][y2]['fluid'] != 'gas'):
                    fluid_condition = True

                if fluid_condition is True:
                    queue.append(path + [(x2, y2)])
                    seen.add((x2, y2))

        if end == 'y-direction':
            if x == start[0] and y == h-1:
                #for (x , y) in seen:
                #    Matrix[x][y]['color'] = gas_path
                return path

        if end == 'x-direction':
            if x == w-1 and y == start[1]:
                #for (x , y) in seen:
                #    Matrix[x][y]['color'] = gas_path
                return path

def bfs_remove_gas(grid, start):
    queue = collections.deque([[start]])
    seen = set([start])
    while queue:
        path = queue.popleft()
        x, y = path[-1]
        if (x in [0,w-1]) or (y in [0,h-1]):
            return path

        for x2, y2 in ((x+1,y), (x-1,y), (x,y+1), (x,y-1)):
            if (0 <= x2 < w) and (0 <= y2 < h) and (Matrix[x2][y2]['fluid'] == 'gas') and (x2, y2)
                not in seen:
```

```
                queue.append(path + [(x2, y2)])
                seen.add((x2, y2))
    for x,y in seen:
        Matrix[x][y]['fluid'] = 'water'
        Matrix[x][y]['color'] = water
        checked.append((x,y))

def bfs_remove_bridge(grid, start):
    queue = collections.deque([[start]])
    seen = set([start])
    while queue:
        path = queue.popleft()
        x, y = path[-1]
        if (x in [0,w-1]) or (y in [0,h-1]):
            return path

        for x2, y2 in ((x+1,y), (x-1,y), (x,y+1), (x,y-1)):
            if (0 <= x2 < w) and (0 <= y2 < h) and ((Matrix[x2][y2]['fluid'] == 'gas') or (Matrix[x2
                ][y2]['fluid'] == 'bridge')) and (x2, y2) not in seen:
                queue.append(path + [(x2, y2)])
                seen.add((x2, y2))
    for x,y in seen:
        Matrix[x][y]['fluid'] = 'water'
        Matrix[x][y]['color'] = water
        checked.append((x,y))
```

Code: The results shown in Fig. 20

```
import numpy as np
import matplotlib
import matplotlib.pyplot as plt
import matplotlib as mpl
import collections
import json
import copy
import os
```

```
import seaborn as sns

mpl.rcParams["figure.dpi"]= 375

sns.set_style("whitegrid")
sns.set_style(rc={"ytick.left": True})

mpl.rc("font",family="Times␣New␣Roman")
mpl.rcParams["mathtext.fontset"] = "custom"
mpl.rcParams["mathtext.rm"] = "Times␣New␣Roman"
mpl.rcParams["mathtext.it"] = "Times␣New␣Roman:italic"
mpl.rcParams["mathtext.bf"] = "Times␣New␣Roman:bold"

mpl.rcParams.update({"figure.autolayout": True})

font_size = 30

p_dict = {}
grid_range = range(4,68,4)
xtick_range = range(8,72,8)
p_water_medians = []
p_gas_medians = []

#colors
pillar      = "silver"
pillar_path = "whitesmoke"
water       = "royalblue"
water_path  = "orange"     #orange
gas         = "black"
gas_path    = "lightgreen" #lightgreen for paths
bridge      = "yellow"
snap_off    = "red"

for i in grid_range:#,20,22,24,26,28,30,32,34,36,38,40]
    gridsize = i
    gridsize_int = int(gridsize/2)  #amount of pillars

```

```
    fign = 0
    p_gas_list = []
    p_water_list = []

    for seed in range(0,100):

        np.random.seed(seed)

        #Initial Matrix
        w, h = gridsize, gridsize #w = width
        Matrix = [[{"throat_size":0,"type":"pillar", "fluid":"pillar","color":pillar} for x in range
            (w)] for y in range(h)]

        #all bodies initially filled with water
        for x in range(w)[::2]:
            for y in range(h)[::2]:
                    Matrix[x][y] = {"throat_size":1,"type":"body","fluid":"water", "color":water}

        #setting throat size depending on p
        for x in range(w+1)[::2]:
            for y in range(h):
                if (x - 1 -(y %2)) >= 0:
                    Matrix[x - 1 - (y %2)][y] = {"throat_size":np.random.rand(),"type":"throat","
                        fluid":"water","color":water}



        for p in np.round(np.arange(0.1,1,0.01),2)[::-1]:
            # setting gas throats
            for x in range(w+1)[::2]:
                for y in range(h):
                    if (x - 1 -(y %2)) >= 0:
                        if (Matrix[x - 1 - (y %2)][y]["type"] == "throat") and (Matrix[x - 1 - (y
                            %2)][y]["throat_size"] > p):
                            Matrix[x - 1 - (y %2)][y]["fluid"] = "gas"
                            Matrix[x - 1 - (y %2)][y]["color"] = gas

            #setting porebodies adjacent to gas throats to gas
```

```
            for x in range(0,w):
                for y in range(0,h):
                    if Matrix[x][y]["fluid"] == "gas":

                        for i in [-1,1]:
                            if 0 <= x <= w - 1:
                                ix = i
                                if x == 0:
                                    ix = 1

                                if x == w -1:
                                    ix = -1
                                if Matrix[x+ix][y]["type"] == "body":
                                    Matrix[x+ix][y]["fluid"] = "gas"
                                    Matrix[x+ix][y]["color"] = gas

                                if x == 0: # make sure boundary has pore bodies adjacent vertically
                                    iy = i

                                    if y == h - 1:
                                        iy = -1
                                    if Matrix[x][y+iy]["type"] == "body":
                                        Matrix[x][y+iy]["fluid"] = "gas"
                                        Matrix[x][y+iy]["color"] = gas


                            if 0 <= y <= h - 1:
                                if y == 0:
                                    i = 1
                                if y == h - 1:
                                    i = -1
                                if Matrix[x][y+i]["type"] == "body":
                                    Matrix[x][y+i]["fluid"] = "gas"
                                    Matrix[x][y+i]["color"] = gas

                                if y == 0: # make sure boundary has pore bodies adjacent horizontally
```

```
                            ix = i

                            if x == w - 1:
                                ix = -1
                            if Matrix[x+ix][y]["type"] == "body":
                                Matrix[x+ix][y]["fluid"] = "gas"
                                Matrix[x+ix][y]["color"] = gas

                # make sure wrap around boundary condition is satisfied
                if Matrix[w - 1][y]["fluid"] == "gas":
                    Matrix[0][y]["type"] == "body"
                    Matrix[0][y]["fluid"] = "gas"
                    Matrix[0][y]["color"] = gas
                if Matrix[x][h-1]["fluid"] == "gas":
                    Matrix[x][0]["type"] == "body"
                    Matrix[x][0]["fluid"] = "gas"
                    Matrix[x][0]["color"] = gas


    x_path_exists_check, y_path_exists_check = False, False
    x_path_coord_list_gas, y_path_coord_list_gas = [], []
    x_dir_list_gas, y_dir_list_gas = [], [] #The whole gas backbone

    if not y_path_exists_check:
        for x in range(0,w):
            if (Matrix[x][0]["fluid"] == "gas"):
                y_dir = bfs_opposite_side(Matrix, (x,0),"y-direction","gas")
                if y_dir is not None:
                    y_path_coord_list_gas.append(x)
                    y_path_exists_check = True
                    y_dir_list_gas.append(y_dir)




    if not x_path_exists_check:

```

```
                for y in range(0,h):
                    if (Matrix[0][y]["fluid"] == "gas"):
                        x_dir = bfs_opposite_side(Matrix, (0,y),"x-direction","gas")

                        if x_dir is not None:
                            x_path_coord_list_gas.append(y)
                            x_path_exists_check = True
                            x_dir_list_gas.append(x_dir)



            if x_path_exists_check and y_path_exists_check:

                    #remove gas
                checked = []
                for x in range(w):
                    for y in range(h):
                        bfs_remove_gas(Matrix,(x,y))

                    #set pillars back
                for x in range(w)[1::2]:
                    for y in range(h)[1::2]:
                        Matrix[x][y] = {"throat_size":0,"type":"pillar", "fluid":"pillar","color":pillar}

                for y_dir in y_dir_list_gas:
                    for (i,j) in y_dir:
                        Matrix[i][j]["color"] = gas_path
                for x_dir in x_dir_list_gas:
                    for (i,j) in x_dir:
                        Matrix[i][j]["color"] = gas_path



                #create textfile for information and add p of the first gas flow paths
                if not os.path.exists(f"Network_images_final/Gridsize_{gridsize_int}/Seed_{seed}"):
                    os.makedirs(f"Network_images_final/Gridsize_{gridsize_int}/Seed_{seed}")

```

```python
with open(f"Network␣images␣final/Gridsize␣{gridsize_int}/Seed␣{seed}/x_paths_gas.txt", "a") as f:
    for xp in x_path_coord_list_gas:
        f.write(f"{xp}␣")

with open(f"Network␣images␣final/Gridsize␣{gridsize_int}/Seed␣{seed}/y_paths_gas.txt", "a") as f:
    for yp in y_path_coord_list_gas:

        f.write(f"{yp}␣")
p_first_gas = p

#creating file of locations of corner shapes to be added in COMSOL
cornergrid = np.zeros((w,h))
for j in range(1,h,2):
    for i in range(1,w,2):

        #if (i < len(coolgrid)-2) and (j < len(coolgrid)-2):

        x_north = i
        x_east = i+1
        x_south = i
        x_west = i-1

        y_north = j+1
        y_east = j
        y_south = j-1
        y_west = j

        #wrap around boundary condition
        if i == w - 1:
            x_east = 0
        if j == h - 1:
            y_north = 0

        # check if porethroats filled with water
        N_throat = (Matrix[y_north][x_north]["fluid"] == "water")
```

```
                        E_throat = (Matrix[y_east][x_east]["fluid"] == "water")
                        S_throat = (Matrix[y_south][x_south]["fluid"] == "water")
                        W_throat = (Matrix[y_west][x_west]["fluid"] == "water")

                        # check if porebodies not filled with water
                        NE_pb = (Matrix[y_north][x_east]["fluid"] == "water")
                        SE_pb = (Matrix[y_south][x_east]["fluid"] == "water")
                        SW_pb = (Matrix[y_south][x_west]["fluid"] == "water")
                        NW_pb = (Matrix[y_north][x_west]["fluid"] == "water")

                        if W_throat and S_throat and not SW_pb :
                            cornergrid[i-1][j-1] = int(1)

                        if E_throat and S_throat and not SE_pb:
                            cornergrid[i][j-1] = 2

                        if E_throat and N_throat and not NE_pb:
                            cornergrid[i][j] = 3

                        if W_throat and N_throat and not NW_pb:
                            cornergrid[i-1][j] = 4

                #print(cornergrid)
                cornergrid = cornergrid.astype(int)
                np.savetxt(f"Network␣images␣final/Gridsize␣{gridsize_int}/Seed␣{seed}/corners.txt", cornergrid, delimiter="␣",fmt="%s")
                break

        p_lowest= np.inf # first p to have bridges horizontally and vertically

        x_path_coord_list_water = []
        y_path_coord_list_water = []

        first_water = 0
        for p_i in np.round(p + np.arange(0,1.01,0.01), 2): #np.round(p + np.arange(0,0.90,0.01), 2):

```

```
        if (int(p_i*100) > 100):
            break

        p_bridge = [p, p_i]

        #place bridges
        for x in range(w):
            for y in range(h):
                if (p_bridge[0] <= round(Matrix[x][y]['throat_size'],2) <= p_bridge[1]) and (Matrix[x][y]['fluid'] == 'gas'):
                    if Matrix[x][y]['type'] == 'throat':
                        Matrix[x][y]['fluid'] = 'bridge'
                        Matrix[x][y]['color'] = bridge

        n_x_paths = 0
        n_y_paths = 0

        x_dir_list_water, y_dir_list_water = [], [] #The whole water backbone

        #water paths
        for x in range(0,w):
            if (Matrix[x][0]["fluid"] != "gas"):
                y_dir2 = bfs_opposite_side(Matrix, (x,0),"y-direction","water")
                if y_dir2 is not None:

                    y_dir_list_water.append(y_dir2)
                    n_y_paths += 1

                    if x not in y_path_coord_list_water:
                        y_path_coord_list_water.append(x)

                    for (i,j) in y_dir2:
                        if Matrix[i][j]["fluid"] != "bridge":
                            if Matrix[i][j]["fluid"] == "water":
                                Matrix[i][j]["color"] = water_path


```

```
        for y in range(0,h):
            if (Matrix[0][y]["fluid"] != "gas"):
                x_dir2 = bfs_opposite_side(Matrix, (0,y),"x-direction","water")

                if x_dir2 is not None:
                    x_dir_list_water.append(x_dir2)
                    n_x_paths += 1

                    if y not in x_path_coord_list_water:
                        x_path_coord_list_water.append(y)

                    for (i,j) in x_dir2:
                        if Matrix[i][j]["fluid"] != "bridge":
                            if Matrix[i][j]["fluid"] == "water":
                                Matrix[i][j]["color"] = water_path

        if (n_x_paths > 0) and (n_y_paths > 0):
            first_water += 1

            if first_water == 1 or p_i == 1:

                if p_i < p_lowest:
                    p_lowest = p_i #finding the lowest p value where gas and water both bridge
                        horizontally and vertically

                Output = np.array(copy.deepcopy(Matrix))
                for jj in range(h):
                    for ii in range(w):
                        Output[ii,jj] = Output[ii,jj]["fluid"]

                horizontals = Output[1::2,::2]
                verticals = Output[::2, 1::2]
                porebodies = Output[::2,::2]

                p_values = [p_first_gas, p_lowest]
                p_gas_list.append(p_first_gas)
                p_water_list.append(p_lowest)
```

```
                    pnumb = int(round(10*(p_i - p_lowest)))

                    if not os.path.exists(f"Network_images_final/Gridsize_{gridsize_int}/Seed_{seed}/P{pnumb}"):
                        os.makedirs(f"Network_images_final/Gridsize_{gridsize_int}/Seed_{seed}/P{pnumb}")



                    np.savetxt(f"Network_images_final/Gridsize_{gridsize_int}/Seed_{seed}/P{pnumb}/horizontals.txt", horizontals, delimiter=" ",fmt="%s")
                    np.savetxt(f"Network_images_final/Gridsize_{gridsize_int}/Seed_{seed}/P{pnumb}/verticals.txt", verticals, delimiter=" ",fmt="%s")
                    np.savetxt(f"Network_images_final/Gridsize_{gridsize_int}/Seed_{seed}/P{pnumb}/porebodies.txt", porebodies, delimiter=" ",fmt="%s")
                    #plot_grid(fign, f"Seed: {seed}, p: {p_i}, x_paths: {n_x_paths}, y_paths: {n_y_paths}",f"P{pnumb}/network_bridge_iteration.png")
                    #fign += 1

                    with open(f"Network_images_final/Gridsize_{gridsize_int}/Seed_{seed}/info.txt", "a") as f:
                        f.write(f"{p_i}\n")



                    with open(f"Network_images_final/Gridsize_{gridsize_int}/Seed_{seed}/P{pnumb}/x_paths_water.txt", "a") as f:
                        for xp in x_path_coord_list_water:
                            f.write(f"{xp} ")

                    with open(f"Network_images_final/Gridsize_{gridsize_int}/Seed_{seed}/P{pnumb}/y_paths_water.txt", "a") as f:
                        for yp in y_path_coord_list_water:
                            f.write(f"{yp} ")

                    break
    print(gridsize)
    fig1 = plt.figure(fign,figsize=(6, 6))
    plt.scatter(gridsize*np.ones(len(p_water_list)),p_water_list,s=None,alpha=0.1,c="grey")
```

```python
    plt.scatter(gridsize, np.median(p_water_list),c="blue")
    plt.title("First WP Flow Paths",size=font_size)
    plt.xlabel("Lattice Size",fontsize=font_size)
    plt.ylabel("$p_{network}$",fontsize=font_size)
    plt.xticks(xtick_range,fontsize=font_size-5)
    plt.yticks(fontsize=font_size)
    plt.ylim([0,1])

    fig2 = plt.figure(fign+1,figsize=(6, 6))
    plt.scatter(gridsize*np.ones(len(p_gas_list)),p_gas_list,s=None,alpha=0.1,c="grey")
    plt.scatter(gridsize, np.median(p_gas_list),c="green")
    plt.title("First NWP Flow Paths",size=font_size)
    plt.xlabel("Lattice Size",fontsize=font_size)
    plt.ylabel("$p_{network}$",fontsize=font_size)
    plt.xticks(xtick_range,fontsize=font_size-5)
    plt.yticks(fontsize=font_size)
    plt.ylim([0,1])
    #p_water_list

    p_water_medians.append(np.median(p_water_list))
    p_gas_medians.append(np.median(p_gas_list))


fig3 = plt.figure(fign+2,figsize=(6, 6))
plt.scatter(grid_range,p_water_medians,c="blue")
plt.title("Median Values")
plt.xlabel("Lattice Size",fontsize=font_size)
plt.ylabel("$p_{network}$",fontsize=font_size)
plt.xticks(xtick_range,fontsize=font_size-5)
plt.yticks(fontsize=font_size)
plt.ylim([0,1])

plt.scatter(grid_range,p_gas_medians,c="green")
plt.xlabel("Lattice Size",fontsize=font_size)
plt.ylabel("$p_{network}$",fontsize=font_size)
plt.xticks(xtick_range,fontsize=font_size-5)
plt.yticks(fontsize=font_size)
```

```
plt.ylim([0,1])

print(p_water_medians)
print(p_gas_medians)

#save figures
fig1.savefig("firstWP.png")
fig2.savefig("firstNWP.png")
fig3.savefig("firstmedians.png")
```

Code: To determine the arrangement of the networks used to obtain the results of Fig. 22

```
import numpy as np
import matplotlib
import matplotlib.pyplot as plt
import collections
import json
import copy
import os

gridsize = 24
gridsize_int = int(gridsize/2)  #amount of pillars

gridsizes = [gridsize]
ssizes =    [750] #size of the squares in

s = ssizes[gridsizes.index(gridsize)]

#colors
pillar      = 'silver'
pillar_path = 'whitesmoke'
water       = 'royalblue'
water_path  = 'royalblue'
gas         = 'black'
gas_path    = 'black'
```

```
bridge      = 'yellow'
snap_off    = 'red'

fign = 0

for seed in range(0,10):

    np.random.seed(seed)

    #Initial Matrix
    w, h = gridsize, gridsize #w = width
    Matrix = [[{'throat_size':0,'type':'pillar', 'fluid':'pillar','color':pillar} for x in range(w)]
         for y in range(h)]

    #all bodies initially filled with water
    for x in range(w)[::2]:
        for y in range(h)[::2]:
                Matrix[x][y] = {'throat_size':1,'type':'body','fluid':'water', 'color':water}

    #setting throat size depending on p
    for x in range(w+1)[::2]:
        for y in range(h):
            if (x - 1 -(y %2)) >= 0:
                Matrix[x - 1 - (y %2)][y] = {'throat_size':np.random.rand(),'type':'throat','fluid
                    ':'water','color':water}

    for p in np.round(np.arange(0.1,1,0.01),2)[::-1]:
        # setting gas throats
        for x in range(w+1)[::2]:
            for y in range(h):
                if (x - 1 -(y %2)) >= 0:
                    if (Matrix[x - 1 - (y %2)][y]['type'] == 'throat') and (Matrix[x - 1 - (y %2)][y
                        ]['throat_size'] > p):
                        Matrix[x - 1 - (y %2)][y]['fluid'] = 'gas'
                        Matrix[x - 1 - (y %2)][y]['color'] = gas

        #setting porebodies adjacent to gas throats to gas
```

```
        for x in range(0,w):
            for y in range(0,h):
                if Matrix[x][y][’fluid’] == ’gas’:

                    for i in [-1,1]:
                        if 0 <= x <= w - 1:
                            ix = i
                            if x == 0:
                                ix = 1

                            if x == w -1:
                                ix = -1
                            if Matrix[x+ix][y][’type’] == ’body’:
                                Matrix[x+ix][y][’fluid’] = ’gas’
                                Matrix[x+ix][y][’color’] = gas

                            if x == 0: # make sure boundary has pore bodies adjacent vertically
                                iy = i

                                if y == h - 1:
                                    iy = -1
                                if Matrix[x][y+iy][’type’] == ’body’:
                                    Matrix[x][y+iy][’fluid’] = ’gas’
                                    Matrix[x][y+iy][’color’] = gas


                        if 0 <= y <= h - 1:
                            if y == 0:
                                i = 1
                            if y == h - 1:
                                i = -1
                            if Matrix[x][y+i][’type’] == ’body’:
                                Matrix[x][y+i][’fluid’] = ’gas’
                                Matrix[x][y+i][’color’] = gas

                            if y == 0: # make sure boundary has pore bodies adjacent horizontally
                                ix = i
```

```

                                if x == w - 1:
                                    ix = -1
                                if Matrix[x+ix][y]['type'] == 'body':
                                    Matrix[x+ix][y]['fluid'] = 'gas'
                                    Matrix[x+ix][y]['color'] = gas

                    # make sure wrap around boundary condition is satisfied
                    if Matrix[w - 1][y]['fluid'] == 'gas':
                        Matrix[0][y]['type'] == 'body'
                        Matrix[0][y]['fluid'] = 'gas'
                        Matrix[0][y]['color'] = gas
                    if Matrix[x][h-1]['fluid'] == 'gas':
                        Matrix[x][0]['type'] == 'body'
                        Matrix[x][0]['fluid'] = 'gas'
                        Matrix[x][0]['color'] = gas

        x_path_exists_check, y_path_exists_check = False, False
        x_path_coord_list_gas, y_path_coord_list_gas = [], []
        x_dir_list_gas, y_dir_list_gas = [], [] #The whole gas backbone

        if not y_path_exists_check:
            for x in range(0,w):
                if (Matrix[x][0]['fluid'] == 'gas'):
                    y_dir = bfs_opposite_side(Matrix, (x,0),'y-direction','gas')
                    if y_dir is not None:
                        y_path_coord_list_gas.append(x)
                        y_path_exists_check = True
                        y_dir_list_gas.append(y_dir)




        if not x_path_exists_check:

            for y in range(0,h):
                if (Matrix[0][y]['fluid'] == 'gas'):
```

```
                x_dir = bfs_opposite_side(Matrix, (0,y),'x-direction','gas')

                if x_dir is not None:
                    x_path_coord_list_gas.append(y)
                    x_path_exists_check = True
                    x_dir_list_gas.append(x_dir)



        if x_path_exists_check and y_path_exists_check:

            #remove gas
            checked = []
            for x in range(w):
                for y in range(h):
                    bfs_remove_gas(Matrix,(x,y))

            #set pillars back
            for x in range(w)[1::2]:
                for y in range(h)[1::2]:
                    Matrix[x][y] = {'throat_size':0,'type':'pillar', 'fluid':'pillar','color':pillar}

            for y_dir in y_dir_list_gas:
                for (i,j) in y_dir:
                    Matrix[i][j]['color'] = gas_path
            for x_dir in x_dir_list_gas:
                for (i,j) in x_dir:
                    Matrix[i][j]['color'] = gas_path

            #create textfile for information and add p of the first gas flow paths
            if not os.path.exists(f'Network images final/Gridsize {gridsize_int}/Seed {seed}'):
                os.makedirs(f'Network images final/Gridsize {gridsize_int}/Seed {seed}')

            with open(f'Network images final/Gridsize {gridsize_int}/Seed {seed}/x_paths_gas.txt', 'a') as f:
                for xp in x_path_coord_list_gas:
```

```
                f.write(f’{xp} ’)

        with open(f’Network images final/Gridsize {gridsize_int}/Seed {seed}/y_paths_gas.txt’, ’
            a’) as f:
            for yp in y_path_coord_list_gas:

                f.write(f’{yp} ’)
        p_first_gas = p

        plot_grid(fign, f’Seed: {seed}, p: {p_first_gas}, First gas flow paths’,’
            network_first_gas.png’)
        fign += 1

        #creating file of locations of corner shapes to be added in COMSOL
        cornergrid = np.zeros((w,h))
        for j in range(1,h,2):
            for i in range(1,w,2):

                #if (i < len(coolgrid)-2) and (j < len(coolgrid)-2):

                x_north = i
                x_east = i+1
                x_south = i
                x_west = i-1

                y_north = j+1
                y_east = j
                y_south = j-1
                y_west = j

                #wrap around boundary condition
                if i == w - 1:
                    x_east = 0
                if j == h - 1:
                    y_north = 0

                # check if porethroats filled with water
```

```
                N_throat = (Matrix[y_north][x_north]['fluid'] == 'water')
                E_throat = (Matrix[y_east][x_east]['fluid'] == 'water')
                S_throat = (Matrix[y_south][x_south]['fluid'] == 'water')
                W_throat = (Matrix[y_west][x_west]['fluid'] == 'water')

                # check if porebodies not filled with water
                NE_pb = (Matrix[y_north][x_east]['fluid'] == 'water')
                SE_pb = (Matrix[y_south][x_east]['fluid'] == 'water')
                SW_pb = (Matrix[y_south][x_west]['fluid'] == 'water')
                NW_pb = (Matrix[y_north][x_west]['fluid'] == 'water')

                if W_throat and S_throat and not SW_pb :
                    cornergrid[i-1][j-1] = int(1)

                if E_throat and S_throat and not SE_pb:
                    cornergrid[i][j-1] = 2

                if E_throat and N_throat and not NE_pb:
                    cornergrid[i][j] = 3

                if W_throat and N_throat and not NW_pb:
                    cornergrid[i-1][j] = 4

        #save(cornergrid)
        cornergrid = cornergrid.astype(int)
        np.savetxt(f'Network images final/Gridsize {gridsize_int}/Seed {seed}/corners.txt',
            cornergrid, delimiter=' ',fmt='%s')
        break

p_lowest= np.inf # first p to have bridges horizontally and vertically

x_path_coord_list_water = []
y_path_coord_list_water = []

fign += 1

first_water = 0
```

```
pnumb = -1
for p_i in np.round(p + np.arange(0,1.01,0.01), 2): #np.round(p + np.arange(0,0.90,0.01), 2):

    if (int(p_i*100) > 100):
        break
    if pnumb > 2:

        break
    p_bridge = [p, p_i]

    #place bridges
    for x in range(w):
        for y in range(h):
            if (p_bridge[0] <= round(Matrix[x][y]['throat_size'],2) <= p_bridge[1]) and (Matrix[
                x][y]['fluid'] == 'gas'):
                if Matrix[x][y]['type'] == 'throat':
                    Matrix[x][y]['fluid'] = 'bridge'
                    Matrix[x][y]['color'] = bridge

    n_x_paths = 0
    n_y_paths = 0

    x_dir_list_water, y_dir_list_water = [], [] #The whole water backbone

    #water paths
    for x in range(0,w):
        if (Matrix[x][0]['fluid'] != 'gas'):
            y_dir2 = bfs_opposite_side(Matrix, (x,0),'y-direction','water')
            if y_dir2 is not None:

                y_dir_list_water.append(y_dir2)
                n_y_paths += 1

                if x not in y_path_coord_list_water:
                    y_path_coord_list_water.append(x)

                for (i,j) in y_dir2:
```

```
                        if Matrix[i][j]['fluid'] != 'bridge':
                            #if Matrix[i][j]['fluid'] == None:
                                #Matrix[i][j]['color'] = pillar_path
                            if Matrix[i][j]['fluid'] == 'water':
                                Matrix[i][j]['color'] = water_path



        for y in range(0,h):
            if (Matrix[0][y]['fluid'] != 'gas'):
                x_dir2 = bfs_opposite_side(Matrix, (0,y),'x-direction','water')

                if x_dir2 is not None:
                    x_dir_list_water.append(x_dir2)
                    n_x_paths += 1

                    if y not in x_path_coord_list_water:
                        x_path_coord_list_water.append(y)

                    for (i,j) in x_dir2:
                        if Matrix[i][j]['fluid'] != 'bridge':
                            #if Matrix[i][j]['fluid'] == None:
                                #Matrix[i][j]['color'] = pillar_path
                            if Matrix[i][j]['fluid'] == 'water':
                                Matrix[i][j]['color'] = water_path
        if (int(round((p_i*100 - p_first_gas*100)) % 10 == 0) or (int(round(p_i*100)) == 100)):
            pnumb += 1
            if (n_x_paths > 0) and (n_y_paths > 0):



                Output = np.array(copy.deepcopy(Matrix))
                for jj in range(h):
                    for ii in range(w):
                        Output[ii,jj] = Output[ii,jj]['fluid']

                horizontals = Output[1::2,::2]
                verticals = Output[::2, 1::2]
                porebodies = Output[::2,::2]
```

```

                if not os.path.exists(f'Network images final/Gridsize {gridsize_int}/Seed {seed}/P{pnumb}'):
                    os.makedirs(f'Network images final/Gridsize {gridsize_int}/Seed {seed}/P{pnumb}')


                np.savetxt(f'Network images final/Gridsize {gridsize_int}/Seed {seed}/P{pnumb}/horizontals.txt', horizontals, delimiter=' ',fmt='%s')
                np.savetxt(f'Network images final/Gridsize {gridsize_int}/Seed {seed}/P{pnumb}/verticals.txt', verticals, delimiter=' ',fmt='%s')
                np.savetxt(f'Network images final/Gridsize {gridsize_int}/Seed {seed}/P{pnumb}/porebodies.txt', porebodies, delimiter=' ',fmt='%s')
                plot_grid(fign, f'Seed: {seed}, p: {p_i}, x_paths: {n_x_paths}, y_paths: {n_y_paths}',f'P{pnumb}/network_bridge_iteration.png')
                fign += 1

                with open(f'Network images final/Gridsize {gridsize_int}/Seed {seed}/info.txt', 'a') as f:
                    f.write(f'{p_i}\n')


                with open(f'Network images final/Gridsize {gridsize_int}/Seed {seed}/P{pnumb}/x_paths_water.txt', 'a') as f:
                    for xp in x_path_coord_list_water:
                        f.write(f'{xp} ')

                with open(f'Network images final/Gridsize {gridsize_int}/Seed {seed}/P{pnumb}/y_paths_water.txt', 'a') as f:
                    for yp in y_path_coord_list_water:
                        f.write(f'{yp} ')

                #break
```

Code: COMSOL Multiphysics®/Java code to set the parameters and create the individual geometry parts

```java
package builder;

import com.comsol.api.*;
import com.comsol.model.*;
import com.comsol.model.physics.*;
import com.comsol.model.application.*;

public class partsbuilder extends ApplicationMethod {

  public void execute() {

    model.param().set("H", "2");
    model.param().descr("H", "Height");
    model.param().set("R", "1");
    model.param().descr("R", "Radius");
    model.param().set("D", "2");
    model.param().descr("D", "Displacement");

    model.geom().create("part0", "Part", 3);
    model.geom("part0").label("Prep␣Part");
    model.geom("part0").lengthUnit("\u00b5m");
    model.geom("part0").create("wp1", "WorkPlane");
    model.geom("part0").feature("wp1").set("unite", true);
    model.geom("part0").feature("wp1").geom().create("r1", "Rectangle");
    model.geom("part0").feature("wp1").geom().feature("r1").set("size", new int[]{2, 1});
    model.geom("part0").feature("wp1").geom().feature("r1").set("size", new double[]{2, 1.95});
    model.geom("part0").feature("wp1").geom().feature("r1").set("pos", new double[]{0, 0.025});
    model.geom("part0").create("copy1", "Copy");
    model.geom("part0").feature("copy1").set("displz", 2);
    model.geom("part0").feature("copy1").selection("input").set("wp1");
    model.geom("part0").create("wp2", "WorkPlane");
    model.geom("part0").feature("wp2").set("unite", true);
    model.geom("part0").feature("wp2").set("quickz", "11/40");
    model.geom("part0").feature("wp2").geom().create("r1", "Rectangle");
```

```
model.geom("part0").feature("wp2").geom().feature("r1").set("size", new int[]{2, 1});
model.geom("part0").feature("wp2").geom().feature("r1").set("type", "curve");
model.geom("part0").feature("wp2").geom().feature("r1").set("pos", new String[]{"0", "1/2"});
model.geom("part0").feature("wp2").geom().create("qb1", "QuadraticBezier");
model.geom("part0").feature("wp2").geom().feature("qb1").setIndex("p", "1/2", 1, 0);
model.geom("part0").feature("wp2").geom().feature("qb1").setIndex("p", 1, 0, 1);
model.geom("part0").feature("wp2").geom().feature("qb1").setIndex("p", "3/5", 1, 1);
model.geom("part0").feature("wp2").geom().feature("qb1").setIndex("p", 2, 0, 2);
model.geom("part0").feature("wp2").geom().feature("qb1").setIndex("p", "1/2", 1, 2);
model.geom("part0").feature("wp2").geom().feature("qb1").set("w", new int[]{1, 1, 1});
model.geom("part0").feature("wp2").geom().create("mir1", "Mirror");
model.geom("part0").feature("wp2").geom().feature("mir1").selection("input").set("qb1");
model.geom("part0").feature("wp2").geom().feature("mir1").set("pos", new int[]{0, 1});
model.geom("part0").feature("wp2").geom().feature("mir1").set("axis", new int[]{0, 0});
model.geom("part0").feature("wp2").geom().feature("mir1").set("axis", new int[]{0, 1});
model.geom("part0").feature("wp2").geom().feature("mir1").set("keep", true);
model.geom("part0").feature("wp2").geom().feature().create("del1", "Delete");
model.geom("part0").feature("wp2").geom().feature("del1").selection("input").init(1);
model.geom("part0").feature("wp2").geom().feature("del1").selection("input").set("r1", 1, 3);
model.geom("part0").feature("wp2").geom().create("csol1", "ConvertToSolid");
model.geom("part0").feature("wp2").geom().feature("csol1").selection("input").set("del1", "mir1"
    , "qb1");
model.geom("part0").create("copy2", "Copy");
model.geom("part0").feature("copy2").selection("input").set("wp2");
model.geom("part0").feature("copy2").set("displz", "58/40");
model.geom("part0").create("wp3", "WorkPlane");
model.geom("part0").feature("wp3").set("unite", true);
model.geom("part0").feature("wp3").set("quickz", 1);
model.geom("part0").feature("wp3").geom().create("r1", "Rectangle");
model.geom("part0").feature("wp3").geom().feature("r1").set("size", new int[]{2, 1});
model.geom("part0").feature("wp3").geom().feature("r1").set("pos", new String[]{"0", "1/2"});
model.geom("part0").feature("wp3").geom().create("qb1", "QuadraticBezier");
model.geom("part0").feature("wp3").geom().feature("qb1").setIndex("p", "1/2", 1, 0);
model.geom("part0").feature("wp3").geom().feature("qb1").setIndex("p", 1, 0, 1);
model.geom("part0").feature("wp3").geom().feature("qb1").setIndex("p", "9/8", 1, 1);
model.geom("part0").feature("wp3").geom().feature("qb1").setIndex("p", 2, 0, 2);
model.geom("part0").feature("wp3").geom().feature("qb1").setIndex("p", "1/2", 1, 2);
```

```
model.geom("part0").feature("wp3").geom().feature("qb1").set("w", new int[]{1, 1, 1});
model.geom("part0").feature("wp3").geom().create("mir1", "Mirror");
model.geom("part0").feature("wp3").geom().feature("mir1").set("pos", new int[]{0, 1});
model.geom("part0").feature("wp3").geom().feature("mir1").set("axis", new int[]{0, 0});
model.geom("part0").feature("wp3").geom().feature("mir1").set("axis", new int[]{0, 1});
model.geom("part0").feature("wp3").geom().feature("mir1").selection("input").set("qb1");
model.geom("part0").feature("wp3").geom().feature().create("del1", "Delete");
model.geom("part0").feature("wp3").geom().feature("del1").selection("input").init();
model.geom("part0").feature("wp3").geom().feature("mir1").set("keep", true);
model.geom("part0").feature("wp3").geom().feature("del1").selection("input").init(1);
model.geom("part0").feature("wp3").geom().feature("del1").selection("input").set("r1", 1, 3);
model.geom("part0").feature("wp3").geom().create("csol1", "ConvertToSolid");
model.geom("part0").feature("wp3").geom().feature("csol1").selection("input").set("del1", "mir1"
    , "qb1");
model.geom("part0").create("loft1", "Loft");
model.geom("part0").feature("loft1").selection("profile").set("copy1", "copy2", "wp1", "wp2", "
    wp3");

model.geom().create("part1", "Part", 3);
model.geom("part1").lengthUnit("\u00b5m");
model.geom("part1").create("pi1", "PartInstance");
model.geom("part1").feature("pi1").set("selkeepnoncontr", false);
model.geom("part1").feature("pi1").set("part", "part0");
model.geom("part1").create("wp1", "WorkPlane");
model.geom("part1").feature("wp1").set("unite", true);
model.geom("part1").feature("wp1").set("quickplane", "yz");
model.geom("part1").feature("wp1").geom().create("cro1", "CrossSection");
model.geom("part1").feature("wp1").geom().create("ls1", "LineSegment");
model.geom("part1").feature("wp1").geom().feature("ls1").set("specify1", "coord");
model.geom("part1").feature("wp1").geom().feature("ls1").set("coord1", new int[]{1, 0});
model.geom("part1").feature("wp1").geom().feature("ls1").set("specify2", "coord");
model.geom("part1").feature("wp1").geom().feature("ls1").set("coord2", new int[]{1, 2});
model.geom("part1").feature("wp1").geom().create("uni1", "Union");
model.geom("part1").feature("wp1").geom().feature("uni1").selection("input").set("cro1");
model.geom("part1").feature("wp1").geom().feature("uni1").selection("input").set("cro1", "ls1");
model.geom("part1").feature("wp1").geom().feature().create("del1", "Delete");
model.geom("part1").feature("wp1").geom().feature("del1").selection("input").init(1);
```

```
model.geom("part1").feature("wp1").geom().feature("del1").selection("input").set("uni1", 5, 6,
    7);
model.geom("part1").feature().create("rev1", "Revolve");
model.geom("part1").feature("rev1").set("workplane", "wp1");
model.geom("part1").feature("rev1").selection("input").set("wp1");
model.geom("part1").feature("rev1").set("angtype", "specang");
model.geom("part1").feature("rev1").set("pos", new int[]{1, 0});
model.geom("part1").feature().create("del1", "Delete");
model.geom("part1").feature("del1").selection("input").init();
model.geom("part1").feature("del1").selection("input").set("pi1");
model.geom("part1").create("mov1", "Move");
model.geom("part1").feature("mov1").set("disply", -1);
model.geom("part1").feature("mov1").selection("input").set("rev1");

model.geom().create("part2", "Part", 3);
model.geom("part2").lengthUnit("\u00b5m");
model.geom("part2").create("pi1", "PartInstance");
model.geom("part2").feature("pi1").set("selkeepnoncontr", false);
model.geom("part2").feature("pi1").set("part", "part0");
model.geom("part2").create("pi2", "PartInstance");
model.geom("part2").feature("pi2").set("selkeepnoncontr", false);
model.geom("part2").feature("pi2").set("part", "part1");
model.geom("part2").create("mov1", "Move");
model.geom("part2").feature("mov1").set("disply", 1);
model.geom("part2").feature("mov1").selection("input").set("pi2");
model.geom("part2").create("copy1", "Copy");
model.geom("part2").feature("copy1").selection("input").set("mov1");
model.geom("part2").feature("copy1").set("displx", 2);
model.geom("part2").create("cyl1", "Cylinder");
model.geom("part2").feature("cyl1").set("r", "1/2");
model.geom("part2").feature("cyl1").set("h", 2);
model.geom("part2").feature("cyl1").set("pos", new int[]{0, 1, 0});
model.geom("part2").create("copy2", "Copy");
model.geom("part2").feature("copy2").selection("input").set("cyl1");
model.geom("part2").feature("copy2").set("displx", 2);
model.geom("part2").create("dif1", "Difference");
model.geom("part2").feature("dif1").selection("input").set("pi1");
```

```
model.geom("part2").feature("dif1").selection("input2").set("copy1", "copy2", "cyl1", "mov1");
model.geom("part2").create("mov2", "Move");
model.geom("part2").feature("mov2").selection("input").set("dif1");
model.geom("part2").feature("mov2").set("disply", -1);
model.geom("part2").create("blk1", "Block");
model.geom("part2").feature("blk1").set("size", new int[]{2, 2, 1});
model.geom("part2").feature("blk1").set("pos", new int[]{0, -1, 1});
model.geom("part2").create("dif2", "Difference");
model.geom("part2").feature("dif2").selection("input").set("mov2");
model.geom("part2").feature("dif2").selection("input2").set("blk1");

model.geom().create("part3", "Part", 3);
model.geom("part3").lengthUnit("\u00b5m");
model.geom("part3").create("pi1", "PartInstance");
model.geom("part3").feature("pi1").set("selkeepnoncontr", false);
model.geom("part3").feature("pi1").set("part", "part2");
model.geom("part3").create("wp1", "WorkPlane");
model.geom("part3").feature("wp1").set("unite", true);
model.geom("part3").feature("wp1").geom().create("e1", "Ellipse");
model.geom("part3").feature("wp1").geom().run("e1");
model.geom("part3").create("wp2", "WorkPlane");
model.geom("part3").feature("wp2").set("unite", true);
model.geom("part3").feature("wp2").geom().create("e1", "Ellipse");
model.geom("part3").feature("wp2").geom().feature("e1").set("semiaxes", new String[]{"6/10", "1"
    });
model.geom("part3").feature("wp2").geom().feature("e1").set("semiaxes", new String[]{"6/10", "
    35/100"});
model.geom("part3").feature("wp2").set("quickz", 1);
model.geom("part3").create("copy1", "Copy");
model.geom("part3").feature("copy1").selection("input").set("wp1");
model.geom("part3").feature("copy1").set("displz", 2);
model.geom("part3").create("loft1", "Loft");
model.geom("part3").feature("loft1").selection("profile").set("wp1");
model.geom("part3").feature("loft1").selection("profile").set("wp1", "wp2");
model.geom("part3").feature("loft1").selection("profile").set("copy1", "wp1", "wp2");
model.geom("part3").create("rt1", "RigidTransform");
model.geom("part3").feature("rt1").selection("input").set("loft1");
```

```
model.geom("part3").feature("rt1").set("axistype", "x");
model.geom("part3").feature("rt1").set("rot", 90);
model.geom("part3").feature("rt1").set("displ", new int[]{1, 1, 1});
model.geom("part3").create("dif1", "Difference");
model.geom("part3").feature("dif1").selection("input").set("pi1");
model.geom("part3").feature("dif1").selection("input2").set("rt1");

model.geom().create("part4", "Part", 3);
model.geom("part4").label("Part␣4");
model.geom("part4").lengthUnit("\u00b5m");
model.geom("part4").create("pi1", "PartInstance");
model.geom("part4").feature("pi1").set("selkeepnoncontr", false);
model.geom("part4").feature("pi1").set("part", "part1");
model.geom("part4").create("cyl1", "Cylinder");
model.geom("part4").feature("cyl1").set("r", "1/2");
model.geom("part4").feature("cyl1").set("h", 2);
model.geom("part4").create("blk1", "Block");
model.geom("part4").feature("blk1").set("size", new String[]{"2", "-1", "1"});
model.geom("part4").feature("blk1").set("size", new int[]{2, 2, 1});
model.geom("part4").feature("blk1").set("pos", new int[]{-1, -1, 1});
model.geom("part4").create("dif1", "Difference");
model.geom("part4").feature("dif1").selection("input").set("pi1");
model.geom("part4").feature("dif1").selection("input2").set("blk1", "cyl1");

model.geom().create("part5", "Part", 3);
model.geom("part5").lengthUnit("\u00b5m");
model.geom("part5").create("pi1", "PartInstance");
model.geom("part5").feature("pi1").set("selkeepnoncontr", false);
model.geom("part5").feature("pi1").set("part", "part1");
model.geom("part5").create("cyl1", "Cylinder");
model.geom("part5").feature("cyl1").set("r", "1/2");
model.geom("part5").feature("cyl1").set("h", 2);
model.geom("part5").create("arr1", "Array");
model.geom("part5").feature("arr1").selection("input").set("pi1");
model.geom("part5").feature("arr1").selection("input").set("cyl1", "pi1");
model.geom("part5").feature("arr1").set("fullsize", new int[]{2, 2, 1});
model.geom("part5").feature("arr1").set("displ", new int[]{2, 2, 0});
```

```
    model.geom("part5").create("pi2", "PartInstance");
    model.geom("part5").feature("pi2").set("selkeepnoncontr", false);
    model.geom("part5").feature("pi2").set("part", "part2");
    model.geom("part5").create("pi3", "PartInstance");
    model.geom("part5").feature("pi3").set("selkeepnoncontr", false);
    model.geom("part5").feature("pi3").set("part", "part2");
    model.geom("part5").feature("pi3").set("displ", new int[]{0, 2, 0});
    model.geom("part5").create("pi4", "PartInstance");
    model.geom("part5").feature("pi4").set("selkeepnoncontr", false);
    model.geom("part5").feature("pi4").set("part", "part2");
    model.geom("part5").feature("pi4").set("displ", new int[]{1, 0, 0});
    model.geom("part5").feature("pi4").set("rot", 90);
    model.geom("part5").feature("pi4").set("displ", new int[]{0, 0, 0});
    model.geom("part5").create("pi5", "PartInstance");
    model.geom("part5").feature("pi5").set("selkeepnoncontr", false);
    model.geom("part5").feature("pi5").set("part", "part2");
    model.geom("part5").feature("pi5").set("rot", 90);
    model.geom("part5").feature("pi5").set("displ", new int[]{2, 0, 0});
    model.geom("part5").create("uni1", "Union");
    model.geom("part5").feature("uni1").selection("input").set("arr1", "pi2", "pi3", "pi4", "pi5");
    model.geom("part5").create("blk1", "Block");
    model.geom("part5").feature("blk1").set("size", new int[]{2, 2, 1});
    model.geom("part5").create("dif1", "Difference");
    model.geom("part5").feature("dif1").selection("input").set("blk1");
    model.geom("part5").feature("dif1").selection("input2").set("uni1");

    model.geom().create("part6", "Part", 3);
    model.geom("part6").lengthUnit("\u00b5m");
    model.geom("part6").create("pi1", "PartInstance");
    model.geom("part6").feature("pi1").set("selkeepnoncontr", false);
    model.geom("part6").feature("pi1").set("part", "part1");
    model.geom("part6").create("pi2", "PartInstance");
    model.geom("part6").feature("pi2").set("selkeepnoncontr", false);
    model.geom("part6").feature("pi2").set("part", "part2");
    model.geom("part6").create("pi3", "PartInstance");
    model.geom("part6").feature("pi3").set("selkeepnoncontr", false);
    model.geom("part6").feature("pi3").set("part", "part2");
```

```
model.geom("part6").feature("pi3").set("rot", 90);
model.geom("part6").create("mir1", "Mirror");
model.geom("part6").feature("mir1").selection("input").set("pi2", "pi3");
model.geom("part6").feature("mir1").set("axis", new int[]{0, 0, 1});
model.geom("part6").feature("mir1").set("pos", new int[]{0, 0, 1});
model.geom("part6").feature("mir1").set("keep", true);
model.geom("part6").create("cyl1", "Cylinder");
model.geom("part6").feature("cyl1").set("r", "1/2");
model.geom("part6").feature("cyl1").set("h", 2);
model.geom("part6").feature("cyl1").set("pos", new int[]{0, 0, 0});
model.geom("part6").create("wp1", "WorkPlane");
model.geom("part6").feature("wp1").set("unite", true);
model.geom("part6").feature("wp1").set("quickz", 1);
model.geom("part6").feature("wp1").geom().create("qb1", "QuadraticBezier");
model.geom("part6").feature("wp1").geom().feature("qb1").setIndex("p", 0.15, 0, 0);
model.geom("part6").feature("wp1").geom().feature("qb1").setIndex("p", 1.95, 1, 0);
model.geom("part6").feature("wp1").geom().feature("qb1").setIndex("p", 0.5, 0, 1);
model.geom("part6").feature("wp1").geom().feature("qb1").setIndex("p", 1.5, 1, 1);
model.geom("part6").feature("wp1").geom().feature("qb1").setIndex("p", 0.95, 0, 2);
model.geom("part6").feature("wp1").geom().feature("qb1").setIndex("p", 1.15, 1, 2);
model.geom("part6").feature("wp1").geom().feature("qb1").set("w", new String[]{"1", "1/10", "1"
    });
model.geom("part6").feature("wp1").geom().feature("qb1").setIndex("p", 0.95, 1, 0);
model.geom("part6").feature("wp1").geom().feature("qb1").setIndex("p", 0.5, 1, 1);
model.geom("part6").feature("wp1").geom().feature("qb1").setIndex("p", 0.15, 1, 2);
model.geom("part6").feature("wp1").geom().create("pol1", "Polygon");
model.geom("part6").feature("wp1").geom().feature("pol1").set("source", "table");
model.geom("part6").feature("wp1").geom().feature("pol1").set("type", "open");
model.geom("part6").feature("wp1").geom().feature("pol1").setIndex("table", 0.15, 0, 0);
model.geom("part6").feature("wp1").geom().feature("pol1").setIndex("table", 1.95, 0, 1);
model.geom("part6").feature("wp1").geom().feature("pol1").setIndex("table", 0, 1, 0);
model.geom("part6").feature("wp1").geom().feature("pol1").setIndex("table", 1.95, 1, 1);
model.geom("part6").feature("wp1").geom().feature("pol1").setIndex("table", 0, 2, 0);
model.geom("part6").feature("wp1").geom().feature("pol1").setIndex("table", 1, 2, 1);
model.geom("part6").feature("wp1").geom().feature("pol1").setIndex("table", 0.95, 3, 0);
model.geom("part6").feature("wp1").geom().feature("pol1").setIndex("table", 1, 3, 1);
model.geom("part6").feature("wp1").geom().feature("pol1").setIndex("table", 0.95, 4, 0);
```

```
model.geom("part6").feature("wp1").geom().feature("pol1").setIndex("table", 1.15, 4, 1);
model.geom("part6").feature("wp1").geom().feature("pol1").setIndex("table", 0.15, 0, 0);
model.geom("part6").feature("wp1").geom().feature("pol1").setIndex("table", 1.95, 0, 1);
model.geom("part6").feature("wp1").geom().feature("pol1").setIndex("table", 0, 1, 0);
model.geom("part6").feature("wp1").geom().feature("pol1").setIndex("table", 1.95, 1, 1);
model.geom("part6").feature("wp1").geom().feature("pol1").setIndex("table", 0, 2, 0);
model.geom("part6").feature("wp1").geom().feature("pol1").setIndex("table", 1, 2, 1);
model.geom("part6").feature("wp1").geom().feature("pol1").setIndex("table", 0.95, 3, 0);
model.geom("part6").feature("wp1").geom().feature("pol1").setIndex("table", 1, 3, 1);
model.geom("part6").feature("wp1").geom().feature("pol1").setIndex("table", 0.95, 4, 0);
model.geom("part6").feature("wp1").geom().feature("pol1").setIndex("table", 1.15, 4, 1);
model.geom("part6").feature("wp1").geom().feature("pol1").setIndex("table", 0.95, 0, 1);
model.geom("part6").feature("wp1").geom().feature("pol1").setIndex("table", 0.95, 1, 1);
model.geom("part6").feature("wp1").geom().feature("pol1").setIndex("table", 0, 2, 1);
model.geom("part6").feature("wp1").geom().feature("pol1").setIndex("table", 0, 3, 1);
model.geom("part6").feature("wp1").geom().feature("pol1").setIndex("table", 0.15, 4, 1);
model.geom("part6").feature("wp1").geom().create("csol1", "ConvertToSolid");
model.geom("part6").feature("wp1").geom().feature("csol1").selection("input").set("qb1");
model.geom("part6").feature("wp1").geom().feature("csol1").selection("input").set("pol1", "qb1")
    ;
model.geom("part6").create("wp2", "WorkPlane");
model.geom("part6").feature("wp2").set("unite", true);
model.geom("part6").feature("wp2").set("quickz", "1/2");
model.geom("part6").feature("wp2").geom().create("qb1", "QuadraticBezier");
model.geom("part6").feature("wp2").geom().feature("qb1").setIndex("p", 0.25, 0, 0);
model.geom("part6").feature("wp2").geom().feature("qb1").setIndex("p", 0.95, 1, 0);
model.geom("part6").feature("wp2").geom().feature("qb1").setIndex("p", 0.5, 0, 1);
model.geom("part6").feature("wp2").geom().feature("qb1").setIndex("p", 1.5, 1, 1);
model.geom("part6").feature("wp2").geom().feature("qb1").setIndex("p", 0.95, 0, 2);
model.geom("part6").feature("wp2").geom().feature("qb1").setIndex("p", 0.25, 1, 2);
model.geom("part6").feature("wp2").geom().feature("qb1").setIndex("p", 0.5, 1, 1);
model.geom("part6").feature("wp2").geom().feature("qb1").set("w", new double[]{1, 0.1, 1});
model.geom("part6").feature("wp2").geom().create("pol1", "Polygon");
model.geom("part6").feature("wp2").geom().feature("pol1").set("source", "table");
model.geom("part6").feature("wp2").geom().feature("pol1").set("type", "open");
model.geom("part6").feature("wp2").geom().feature("pol1").setIndex("table", 0.25, 0, 0);
model.geom("part6").feature("wp2").geom().feature("pol1").setIndex("table", 0.95, 0, 1);
```

```
model.geom("part6").feature("wp2").geom().feature("pol1").setIndex("table", 0, 1, 0);
model.geom("part6").feature("wp2").geom().feature("pol1").setIndex("table", 0.95, 1, 1);
model.geom("part6").feature("wp2").geom().feature("pol1").setIndex("table", 0, 2, 0);
model.geom("part6").feature("wp2").geom().feature("pol1").setIndex("table", 1, 2, 1);
model.geom("part6").feature("wp2").geom().feature("pol1").setIndex("table", 0.95, 3, 0);
model.geom("part6").feature("wp2").geom().feature("pol1").setIndex("table", 0, 2, 1);
model.geom("part6").feature("wp2").geom().feature("pol1").setIndex("table", 0, 3, 1);
model.geom("part6").feature("wp2").geom().feature("pol1").setIndex("table", 0.95, 4, 0);
model.geom("part6").feature("wp2").geom().feature("pol1").setIndex("table", 0.25, 4, 1);
model.geom("part6").feature("wp2").geom().create("csol1", "ConvertToSolid");
model.geom("part6").feature("wp2").geom().feature("csol1").selection("input").set("pol1");
model.geom("part6").feature("wp2").geom().feature("csol1").selection("input").set("pol1", "qb1")
    ;
model.geom("part6").create("copy1", "Copy");
model.geom("part6").feature("copy1").set("displz", 1);
model.geom("part6").feature("copy1").selection("input").set("wp2");
model.geom("part6").create("wp3", "WorkPlane");
model.geom("part6").feature("wp3").set("unite", true);
model.geom("part6").feature("wp3").geom().create("pol1", "Polygon");
model.geom("part6").feature("wp3").geom().feature("pol1").set("source", "table");
model.geom("part6").feature("wp3").geom().feature("pol1").setIndex("table", 0.75, 0, 0);
model.geom("part6").feature("wp3").geom().feature("pol1").setIndex("table", 0.9, 0, 1);
model.geom("part6").feature("wp3").geom().feature("pol1").setIndex("table", 0, 1, 0);
model.geom("part6").feature("wp3").geom().feature("pol1").setIndex("table", 0.9, 1, 1);
model.geom("part6").feature("wp3").geom().feature("pol1").setIndex("table", 0, 2, 0);
model.geom("part6").feature("wp3").geom().feature("pol1").setIndex("table", 0, 2, 1);
model.geom("part6").feature("wp3").geom().feature("pol1").setIndex("table", 0.9, 3, 0);
model.geom("part6").feature("wp3").geom().feature("pol1").setIndex("table", 0, 3, 1);
model.geom("part6").feature("wp3").geom().feature("pol1").setIndex("table", 0.9, 4, 0);
model.geom("part6").feature("wp3").geom().feature("pol1").setIndex("table", 0.75, 4, 1);
model.geom("part6").feature("wp3").geom().feature("pol1").setIndex("table", 0.75, 5, 0);
model.geom("part6").feature("wp3").geom().feature("pol1").setIndex("table", 0.9, 5, 1);
model.geom("part6").create("copy2", "Copy");
model.geom("part6").feature("copy2").selection("input").set("wp3");
model.geom("part6").feature("copy2").set("displz", 2);
model.geom("part6").create("loft1", "Loft");
model.geom("part6").feature("loft1").selection("profile").set("copy1", "copy2", "wp1", "wp2", "
```

```
        wp3");
    model.geom("part6").create("dif1", "Difference");
    model.geom("part6").feature("dif1").selection("input").set("loft1");
    model.geom("part6").feature("dif1").selection("input2").set("cyl1", "mir1", "pi1", "pi2", "pi3")
        ;
    model.geom("part6").create("mov1", "Move");
    model.geom("part6").feature("mov1").set("disply", 1);
    model.geom("part6").feature("mov1").selection("input").set("dif1");
    model.geom("part6").create("rt1", "RigidTransform");
    model.geom("part6").feature("rt1").selection("input").set("mov1");
    model.geom("part6").feature("rt1").set("displ", new int[]{1, 2, 0});
    model.geom("part6").feature("rt1").set("rot", 180);
    model.geom("part6").create("blk1", "Block");
    model.geom("part6").feature("blk1").set("pos", new int[]{0, 0, 1});
    model.geom("part6").create("dif2", "Difference");
    model.geom("part6").feature("dif2").selection("input").set("rt1");
    model.geom("part6").feature("dif2").selection("input2").set("blk1");
  }
}
```

Code: COMSOL Multiphysics®/Java code to arrange the interface shapes in networks and compute the flow

```
package builder;

import com.comsol.api.*;
import com.comsol.model.*;
import com.comsol.model.physics.*;
import com.comsol.model.application.*;
import java.util.Random;
import javax.swing.JOptionPane;
import java.util.ArrayList;
import java.io.*;
import java.util.List;
import java.nio.file.*;
import javax.swing.*;
```

```java



public class Build extends ApplicationMethod {

  public String[][] loadFileToArray(int gridsize, String filename) {

    //JOptionPane.showMessageDialog(null, ""+System.getProperty("java.io.tmpdir"));
    try {
      File file = new File("C:\\Users\\eobbens\\Master␣Thesis␣Grids\\Gridsize␣"+gridsize+"\\"+
          filename);


      BufferedReader br = new BufferedReader(new FileReader(file));

      String st;
      int lineCount = 0;
      while ((st = br.readLine()) != null) {
        lineCount++;
      }
      br.close();
      br = new BufferedReader(new FileReader(file));

      String[][] fileArray = new String[lineCount][];
      lineCount = 0;
      while ((st = br.readLine()) != null) {
        fileArray[lineCount++] = st.split("␣");
      }
      br.close();

      return fileArray;
    } catch (Exception e) {
      e.printStackTrace();
      //JOptionPane.showMessageDialog(null);
    }
    return null;
  }
```

```java


public void execute() {

  //Deletes all the settings of the previous iteration
  Boolean deletenodes = new Boolean("true");

  for (int gridsize = 16; gridsize < 17; gridsize++) {
    for (int seedn = 7; seedn < 10; seedn++) {
      for (int pn = 0; pn < 5; pn++) {
        for (int directionn = 0; directionn < 1; directionn++) { //0 = x-direction, 1 = y-direction
          for (int fluidn = 0; fluidn < 1; fluidn++) { // 0 = water, 1 = gas
            try {
              //model.component("comp1").geom("geom1").feature().clear();

              String[][] horizontals = loadFileToArray(gridsize, "Seed "+seedn+"\\P"+pn+"\\horizontals.txt");
              String[][] porebodies = loadFileToArray(gridsize, "Seed "+seedn+"\\P"+pn+"\\porebodies.txt");
              String[][] verticals = loadFileToArray(gridsize, "Seed "+seedn+"\\P"+pn+"\\verticals.txt");
              String[][] corners = loadFileToArray(gridsize, "Seed "+seedn+"\\corners.txt");

              if (horizontals == null || porebodies == null || verticals == null || corners == null) {
                continue;
              }

              int count = 1;

              model.component().create("comp1", true);
              model.component("comp1").geom().create("geom1", 3);
              model.component("comp1").geom("geom1").lengthUnit("\u00b5m"); //set lengthscale to microns
              model.component("comp1").mesh().create("mesh1");
              model.component("comp1").physics().create("spf", "CreepingFlow", "geom1"); //add
```

```
creeping flow physics



//whole network selection list
ArrayList<String> selectionlist = new ArrayList<String>();

// create first pillar
model.component("comp1").geom("geom1").create("pi"+count, "PartInstance");
model.component("comp1").geom("geom1").feature("pi"+count).set("part", "part4");
count++;
// create array of pillars
model.component("comp1").geom("geom1").create("arr1", "Array");
model.component("comp1").geom("geom1").feature("arr1").selection("input").set("pi1")
    ;
model.component("comp1").geom("geom1").feature("arr1").set("fullsize", new int[]{
    gridsize, gridsize, 1});
model.component("comp1").geom("geom1").feature("arr1").set("displ", new String[]{"D"
    , "D", "0"});
selectionlist.add("arr1");

// Arrange the geometry parts according to the interface text files created with the
     network arrangement code.
// "verticals.txt": WP pore throat or bridge or NWP pore throat with a top-bottom
    orientation.
// "horizontals.txt": WP pore throat or bridge or NWP pore throat with a left-right
    orientation.
// "porebodies.txt": WP or NWP pore body.
// "corners.txt": locations and orientation to add a WP part geometry where two WP
    filled pore throats meet at a 90 degree angle.

try {
  for (int i = 0; i < gridsize; i++) {
    for (int j = 0; j < gridsize; j++) {
      //String name = null;
      System.out.println("i:␣"+i+",␣j:"+j);
      if (verticals[i][j].equals("water")) {
        model.component("comp1").geom("geom1").create("pi"+count, "PartInstance");
```

```
      model.component("comp1").geom("geom1").feature("pi"+count).set("part", "part2");
      model.component("comp1").geom("geom1").feature("pi"+count).set("displ", new String[]{i+"*D-2", "D*"+j, "0"});
      model.component("comp1").geom("geom1").feature("pi"+count).set("rot", 0);
      selectionlist.add("pi"+count);


      if (i == 0) {
        model.component("comp1").geom("geom1").create("copy"+count, "Copy");
        model.component("comp1").geom("geom1").feature("copy"+count).selection("input").set("pi"+count);
        model.component("comp1").geom("geom1").feature("copy"+count).set("displx", gridsize+"*D");
        selectionlist.add("copy"+count);
      }
    }
    if (verticals[i][j].equals("bridge")) {
      model.component("comp1").geom("geom1").create("pi"+count, "PartInstance");
      model.component("comp1").geom("geom1").feature("pi"+count).set("part", "part3");
      model.component("comp1").geom("geom1").feature("pi"+count).set("displ", new String[]{i+"*D-2", "D*"+j, "0"});
      model.component("comp1").geom("geom1").feature("pi"+count).set("rot", 0);
      selectionlist.add("pi"+count);


      if (i == 0) {
        model.component("comp1").geom("geom1").create("copy"+count, "Copy");
        model.component("comp1").geom("geom1").feature("copy"+count).selection("input").set("pi"+count);
        model.component("comp1").geom("geom1").feature("copy"+count).set("displx", gridsize+"*D");
        selectionlist.add("copy"+count);
      }
    }
    count++;
```

```
                    }
                  }
                } catch (Exception e) {
                  JOptionPane.showMessageDialog(null, ""+e);
                }
                // adding the
                for (int i = 0; i < gridsize; i++) {
                  for (int j = 0; j < gridsize; j++) {

                    if (horizontals[i][j].equals("water")) {
                      model.component("comp1").geom("geom1").create("pi"+count, "PartInstance");
                      model.component("comp1").geom("geom1").feature("pi"+count).set("part", "part2"
                          );
                      model.component("comp1").geom("geom1").feature("pi"+count).set("displ", new
                          String[]{"D*"+i, j+"*D-2", "0"});
                      model.component("comp1").geom("geom1").feature("pi"+count).set("rot", 90);
                      selectionlist.add("pi"+count);

                      if (j == 0) {
                        model.component("comp1").geom("geom1").create("copy"+count, "Copy");
                        model.component("comp1").geom("geom1").feature("copy"+count).selection("
                            input").set("pi"+count);
                        model.component("comp1").geom("geom1").feature("copy"+count).set("disply",
                            gridsize+"*D");
                        selectionlist.add("copy"+count);
                      }
                    }
                    if (horizontals[i][j].equals("bridge")) {
                      model.component("comp1").geom("geom1").create("pi"+count, "PartInstance");
                      model.component("comp1").geom("geom1").feature("pi"+count).set("part", "part3"
                          );
                      model.component("comp1").geom("geom1").feature("pi"+count).set("displ", new
                          String[]{"D*"+i, j+"*D-2", "0"});
                      model.component("comp1").geom("geom1").feature("pi"+count).set("rot", 90);
                      selectionlist.add("pi"+count);

                      if (j == 0) {
```

```
        model.component("comp1").geom("geom1").create("copy"+count, "Copy");
        model.component("comp1").geom("geom1").feature("copy"+count).selection("input").set("pi"+count);
        model.component("comp1").geom("geom1").feature("copy"+count).set("disply", gridsize+"*D");
        selectionlist.add("copy"+count);
      }
    }

    count++;
  }
}
// adding the water filled pore bodies
for (int i = 0; i < gridsize; i++) {
  for (int j = 0; j < gridsize; j++) {
    if (porebodies[i][j].equals("water")) {
      model.component("comp1").geom("geom1").create("pi"+count, "PartInstance");
      model.component("comp1").geom("geom1").feature("pi"+count).set("part", "part5");
      model.component("comp1").geom("geom1").feature("pi"+count).set("displ", new String[]{"D*"+(i-1), "D*"+(j-1), "0"});
      selectionlist.add("pi"+count);


      if (i == 0 && j != 0) {
        model.component("comp1").geom("geom1").create("copy"+count, "Copy");
        model.component("comp1").geom("geom1").feature("copy"+count).selection("input").set("pi"+count);
        model.component("comp1").geom("geom1").feature("copy"+count).set("displx", gridsize+"*D");
        selectionlist.add("copy"+count);
      }

      if (i != 0 && j == 0) {
        model.component("comp1").geom("geom1").create("copy"+count, "Copy");
        model.component("comp1").geom("geom1").feature("copy"+count).selection("input").set("pi"+count);
```

```
        model.component("comp1").geom("geom1").feature("copy"+count).set("disply", gridsize+"*D");
        selectionlist.add("copy"+count);
      }

      if (i == 0 && j == 0) {
        model.component("comp1").geom("geom1").create("copy"+count, "Copy");
        model.component("comp1").geom("geom1").feature("copy"+count).selection("input").set("pi"+count);
        model.component("comp1").geom("geom1").feature("copy"+count).set("displx", gridsize+"*D");
        selectionlist.add("copy"+count);

        int part_number = count;
        count++;

        model.component("comp1").geom("geom1").create("copy"+count, "Copy");
        model.component("comp1").geom("geom1").feature("copy"+count).selection("input").set("pi"+part_number);
        model.component("comp1").geom("geom1").feature("copy"+count).set("disply", gridsize+"*D");
        selectionlist.add("copy"+count);
      }
      count++;
    }

  }

}

for (int i = 0; i < 2*gridsize; i++) {
  for (int j = 0; j < 2*gridsize; j++) {
    if (corners[i][j].equals("1")) {
      model.component("comp1").geom("geom1").create("pi"+count, "PartInstance");
      model.component("comp1").geom("geom1").feature("pi"+count).set("part", "part6");
      model.component("comp1").geom("geom1").feature("pi"+count).set("displ", new
```

```
                    String[]{"D*"+(j/2-0.5), "D*"+(i/2-0.5), "0"});
                  selectionlist.add("pi"+count);

                  count++;
                }
                if (corners[i][j].equals("2")) {
                  model.component("comp1").geom("geom1").create("pi"+count, "PartInstance");
                  model.component("comp1").geom("geom1").feature("pi"+count).set("part", "part6");
                  model.component("comp1").geom("geom1").feature("pi"+count).set("displ", new String[]{"D*"+(j/2-0.5), "D*"+(i/2+0.5), "0"});
                  model.component("comp1").geom("geom1").feature("pi"+count).set("rot", -90);
                  selectionlist.add("pi"+count);

                  count++;
                }
                if (corners[i][j].equals("3")) {
                  model.component("comp1").geom("geom1").create("pi"+count, "PartInstance");
                  model.component("comp1").geom("geom1").feature("pi"+count).set("part", "part6");
                  model.component("comp1").geom("geom1").feature("pi"+count).set("displ", new String[]{"D*"+(j/2+0.5), "D*"+(i/2+0.5), "0"});
                  model.component("comp1").geom("geom1").feature("pi"+count).set("rot", -180);
                  selectionlist.add("pi"+count);

                  count++;
                }
                if (corners[i][j].equals("4")) {
                  model.component("comp1").geom("geom1").create("pi"+count, "PartInstance");
                  model.component("comp1").geom("geom1").feature("pi"+count).set("part", "part6");
                  model.component("comp1").geom("geom1").feature("pi"+count).set("displ", new String[]{"D*"+(j/2+0.5), "D*"+(i/2-0.5), "0"});
                  model.component("comp1").geom("geom1").feature("pi"+count).set("rot", -270);
                  selectionlist.add("pi"+count);

                  count++;
```

```
    }


  }

}

//adding side cutting blocks
model.component("comp1").geom("geom1").create("blk1", "Block");
model.component("comp1").geom("geom1").feature("blk1").set("pos", new String[]{"-D", "-D/2", "0"});
model.component("comp1").geom("geom1").feature("blk1").set("size", new String[]{"1", gridsize+"*D", "1"});

model.component("comp1").geom("geom1").create("blk2", "Block");
model.component("comp1").geom("geom1").feature("blk2").set("pos", new String[]{"-D", "-D", "0"});
model.component("comp1").geom("geom1").feature("blk2").set("size", new String[]{gridsize+1+"*D", "1", "1"});

model.component("comp1").geom("geom1").create("blk3", "Block");
model.component("comp1").geom("geom1").feature("blk3").set("pos", new String[]{gridsize+"*D-D/2", "-D/2", "0"});
model.component("comp1").geom("geom1").feature("blk3").set("size", new String[]{"1", gridsize+"*D", "1"});

model.component("comp1").geom("geom1").create("blk4", "Block");
model.component("comp1").geom("geom1").feature("blk4").set("pos", new String[]{"-D", gridsize+"*D-D/2", "0"});
model.component("comp1").geom("geom1").feature("blk4").set("size", new String[]{gridsize+1+"*D", "1", "1"});

model.component("comp1").geom("geom1").create("dif1", "Difference");
model.component("comp1").geom("geom1").feature("dif1").selection("input").set(selectionlist.toArray(new String[0]));
model.component("comp1").geom("geom1").feature("dif1").selection("input2").set("blk1", "blk2", "blk3", "blk4");
```

```
model.component("comp1").geom("geom1").feature("dif1").set("intbnd", false);

int boxselcount = 0;

//invert geometry for gas flow
if (fluidn == 1) {
  model.component("comp1").geom("geom1").create("cyl1", "Cylinder");
  model.component("comp1").geom("geom1").feature("cyl1").set("r", "1/2");
  model.component("comp1").geom("geom1").create("arr2", "Array");
  model.component("comp1").geom("geom1").feature("arr2").set("fullsize", new int[]{gridsize, gridsize, 1});
  model.component("comp1").geom("geom1").feature("arr2").set("displ", new int[]{2, 2, 0});
  model.component("comp1").geom("geom1").feature("arr2").selection("input").set("cyl1");

  ArrayList<String> selectionlist2 = new ArrayList<String>();
  selectionlist2.add("dif1");
  selectionlist2.add("arr2");

  model.component("comp1").geom("geom1").create("boxsel"+boxselcount, "BoxSelection");
  boxselcount++;

  model.component("comp1").geom("geom1").create("blk5", "Block");
  model.component("comp1").geom("geom1").feature("blk5").set("pos", new int[]{-1, -1, 0});
  model.component("comp1").geom("geom1").feature("blk5").set("size", new int[]{2*gridsize, 2*gridsize, 1});
  model.component("comp1").geom("geom1").create("dif2", "Difference");
  model.component("comp1").geom("geom1").feature("dif2").selection("input").set("blk5");
  model.component("comp1").geom("geom1").feature("dif2").selection("input2").set(selectionlist2.toArray(new String[0]));

  //move part of the network to the otherside to complete gas backbone
  model.component("comp1").geom("geom1").create("arr3", "Array");
```

```
            model.component("comp1").geom("geom1").feature("arr3").selection("input").set("dif2");
            model.component("comp1").geom("geom1").feature("arr3").set("fullsize", new int[]{2, 2, 1});
            model.component("comp1").geom("geom1").feature("arr3").set("displ", new int[]{-2*gridsize, -2*gridsize, 0});

            model.component("comp1").geom("geom1").create("blk6", "Block");
            model.component("comp1").geom("geom1").feature("blk6").set("pos", new int[]{-2, -2, 0});
            model.component("comp1").geom("geom1").feature("blk6").set("size", new int[]{2*gridsize, 2*gridsize, 1});

            model.component("comp1").geom("geom1").create("uni1", "Union");
            model.component("comp1").geom("geom1").feature("uni1").selection("input").set("arr3");

            // remove the edge that is created by the union
            model.component("comp1").geom("geom1").create("int1", "Intersection");
            model.component("comp1").geom("geom1").feature("int1").selection("input").set("blk6");
            model.component("comp1").geom("geom1").feature("int1").selection("input").set("blk6", "uni1");
            model.component("comp1").geom("geom1").feature("int1").set("intbnd", false);

            model.component("comp1").geom("geom1").create("boxsel"+boxselcount, "BoxSelection");
            model.component("comp1").geom("geom1").feature("boxsel"+boxselcount).set("entitydim", 1);
            model.component("comp1").geom("geom1").feature("boxsel"+boxselcount).set("xmin", -1.01);
            model.component("comp1").geom("geom1").feature("boxsel"+boxselcount).set("xmax", -0.99);
            model.component("comp1").geom("geom1").feature("boxsel"+boxselcount).set("condition", "inside");

            model.component("comp1").geom("geom1").create("ige1", "IgnoreEdges");
```

```
        model.component("comp1").geom("geom1").feature("ige1").selection("input").named("boxsel"+boxselcount);
        boxselcount++;

        model.component("comp1").geom("geom1").create("boxsel"+boxselcount, "BoxSelection");
        model.component("comp1").geom("geom1").feature("boxsel"+boxselcount).set("entitydim", 1);
        model.component("comp1").geom("geom1").feature("boxsel"+boxselcount).set("ymin", -1.01);
        model.component("comp1").geom("geom1").feature("boxsel"+boxselcount).set("ymax", -0.99);
        model.component("comp1").geom("geom1").feature("boxsel"+boxselcount).set("condition", "inside");

        model.component("comp1").geom("geom1").create("ige2", "IgnoreEdges");
        model.component("comp1").geom("geom1").feature("ige2").selection("input").named("boxsel"+boxselcount);
        boxselcount++;

        if (porebodies[0][0].equals("water")) {
          model.component("comp1").geom("geom1").create("blk7", "Block");
          model.component("comp1").geom("geom1").feature("blk7").set("pos", new int[]{-2, 0, 0});
          model.component("comp1").geom("geom1").feature("blk7").set("pos", new int[]{-2, -2, 0});

          model.component("comp1").geom("geom1").create("dif3", "Difference");
          model.component("comp1").geom("geom1").feature("dif3").selection("input").set("int1");
          model.component("comp1").geom("geom1").feature("dif3").selection("input2").set("blk7");
        }
      }


      model.component("comp1").geom("geom1").selection().create("csel1", "
```

```
CumulativeSelection"); //create cumulative selection for the upper boundary
                model.component("comp1").geom("geom1").selection("csel1").label("Cumulative Selection 1");

                model.component("comp1").geom("geom1").selection().create("csel2", "CumulativeSelection"); //create cumulative selection for the outlets if left-right flow direction
                model.component("comp1").geom("geom1").selection("csel2").label("Cumulative Selection 2");

                model.component("comp1").geom("geom1").selection().create("csel3", "CumulativeSelection"); //create cumulative selection for the outlets if top-bottom flow direction
                model.component("comp1").geom("geom1").selection("csel3").label("Cumulative Selection 3");


                //upper boundary
                model.component("comp1").geom("geom1").create("boxsel"+boxselcount, "BoxSelection");
                model.component("comp1").geom("geom1").feature("boxsel"+boxselcount).set("zmin", 0.9);
                model.component("comp1").geom("geom1").feature("boxsel"+boxselcount).set("entitydim", 2);
                model.component("comp1").geom("geom1").feature("boxsel"+boxselcount).set("condition", "inside");

                model.component("comp1").geom("geom1").feature("boxsel"+boxselcount).set("contributeto", "csel1"); //contribute to cumulative selection used for inlets

                boxselcount++;

                if (fluidn == 0) {

                  //merging faces and removing edges

                  model.component("comp1").geom("geom1").create("boxsel"+boxselcount, "BoxSelection");
```

```
            model.component("comp1").geom("geom1").feature("boxsel"+boxselcount).set("entitydim", 2);
            model.component("comp1").geom("geom1").feature("boxsel"+boxselcount).set("zmax", 0.01);
            model.component("comp1").geom("geom1").feature("boxsel"+boxselcount).set("condition", "inside");

            // merge all bottom horizontal faces
            model.component("comp1").geom("geom1").create("cmf1", "CompositeFaces");
            model.component("comp1").geom("geom1").feature("cmf1").selection("input").named("boxsel"+boxselcount);

            boxselcount++;

            model.component("comp1").geom("geom1").create("boxsel"+boxselcount, "BoxSelection");
            model.component("comp1").geom("geom1").feature("boxsel"+boxselcount).set("zmin", 0.99);
            model.component("comp1").geom("geom1").feature("boxsel"+boxselcount).set("condition", "inside");
            model.component("comp1").geom("geom1").feature("boxsel"+boxselcount).set("entitydim", 2);

            // merge all top horizontal faces
            model.component("comp1").geom("geom1").create("cmf2", "CompositeFaces");
            model.component("comp1").geom("geom1").feature("cmf2").selection("input").named("boxsel"+boxselcount);

            boxselcount++;


            //create cumulative selection to select unimportant edges
            model.component("comp1").geom("geom1").selection().create("csel4", "CumulativeSelection");


            model.component("comp1").geom("geom1").create("boxsel"+boxselcount, "BoxSelection"
```

```
                        );
                        model.component("comp1").geom("geom1").feature("boxsel"+boxselcount).set("entitydim", 1);
                        model.component("comp1").geom("geom1").feature("boxsel"+boxselcount).set("xmin", -0.9);
                        model.component("comp1").geom("geom1").feature("boxsel"+boxselcount).set("xmax", 2*gridsize-1.1);
                        model.component("comp1").geom("geom1").feature("boxsel"+boxselcount).set("ymin", -0.9);
                        model.component("comp1").geom("geom1").feature("boxsel"+boxselcount).set("ymax", 2*gridsize-1.1);
                        model.component("comp1").geom("geom1").feature("boxsel"+boxselcount).set("zmin", 0.3);
                        model.component("comp1").geom("geom1").feature("boxsel"+boxselcount).set("zmax", 0.9);
                        model.component("comp1").geom("geom1").feature("boxsel"+boxselcount).set("contributeto", "csel4");

                        boxselcount++;

                        model.component("comp1").geom("geom1").create("boxsel"+boxselcount, "BoxSelection");
                        model.component("comp1").geom("geom1").feature("boxsel"+boxselcount).set("entitydim", 1);
                        model.component("comp1").geom("geom1").feature("boxsel"+boxselcount).set("xmin", -0.9);
                        model.component("comp1").geom("geom1").feature("boxsel"+boxselcount).set("xmax", 2*gridsize-1.1);
                        model.component("comp1").geom("geom1").feature("boxsel"+boxselcount).set("ymin", -0.9);
                        model.component("comp1").geom("geom1").feature("boxsel"+boxselcount).set("ymax", 2*gridsize-1.1);
                        model.component("comp1").geom("geom1").feature("boxsel"+boxselcount).set("zmin", 0.01);
                        model.component("comp1").geom("geom1").feature("boxsel"+boxselcount).set("zmax", 0.2);
                        model.component("comp1").geom("geom1").feature("boxsel"+boxselcount).set("
```

```
contributeto", "csel4");

      boxselcount++;

      model.component("comp1").geom("geom1").create("ige1", "IgnoreEdges");
      model.component("comp1").geom("geom1").feature("ige1").selection("input").named("csel4");

      //selection for the no slip boundary condition
      model.component("comp1").geom("geom1").create("boxsel"+boxselcount, "BoxSelection");
      model.component("comp1").geom("geom1").feature("boxsel"+boxselcount).set("entitydim", 2);
      model.component("comp1").geom("geom1").feature("boxsel"+boxselcount).set("condition", "inside");
      model.component("comp1").geom("geom1").feature("boxsel"+boxselcount).set("zmax", 0.3);

      //setting Slip BC on all boundaries
      model.component("comp1").physics("spf").feature("wallbc1").set("BoundaryCondition", "Slip");

      //setting No-Slip on bottom plate and pillars which overrides the slip BC.
      model.component("comp1").physics("spf").create("wallbc2", "WallBC", 2);
      model.component("comp1").physics("spf").feature("wallbc2").selection().named("geom1_boxsel"+boxselcount);
      model.component("comp1").physics("spf").feature("wallbc2").set("BoundaryCondition", "NoSlip");

      boxselcount++;
    }

    if (fluidn == 1) {
      //merging faces and removing edges

      model.component("comp1").geom("geom1").create("boxsel"+boxselcount, "BoxSelection");
```

```
          model.component("comp1").geom("geom1").feature("boxsel"+boxselcount).set("entitydim", 1);
          model.component("comp1").geom("geom1").feature("boxsel"+boxselcount).set("xmin", -1.999);
          model.component("comp1").geom("geom1").feature("boxsel"+boxselcount).set("xmax", 2*gridsize-2.001);
          model.component("comp1").geom("geom1").feature("boxsel"+boxselcount).set("ymin", -1.999);
          model.component("comp1").geom("geom1").feature("boxsel"+boxselcount).set("ymax", 2*gridsize-2.001);
          model.component("comp1").geom("geom1").feature("boxsel"+boxselcount).set("zmax", 0.5);

          model.component("comp1").geom("geom1").feature("boxsel"+boxselcount).set("condition", "intersects");

          // merge all bottom horizontal faces
          model.component("comp1").geom("geom1").create("ige3", "IgnoreEdges");
          model.component("comp1").geom("geom1").feature("ige3").selection("input").named("boxsel"+boxselcount);

          boxselcount++;

        }

        //ignore vertices to aid mesh creation
        model.component("comp1").geom("geom1").create("boxsel"+boxselcount, "BoxSelection");
        model.component("comp1").geom("geom1").feature("boxsel"+boxselcount).set("entitydim", 0);


        model.component("comp1").geom("geom1").create("igv1", "IgnoreVertices");
        model.component("comp1").geom("geom1").feature("igv1").selection("input").named("boxsel"+boxselcount);

        boxselcount++;

```

```java

        ArrayList<Integer> boxsellist1 = new ArrayList<Integer>();
        ArrayList<Integer> boxsellist2 = new ArrayList<Integer>();

        //ArrayList<String> surfaceIntegrals = new ArrayList<String>(); //boxselection name list, input at surface integral node

        Integer fluidoffset = 0; //Setting the offset based on fluid type. used for determining the amount of selections needed and locations of the selections
        if (fluidn == 1) {
          fluidoffset = -1;
        }

        char direction_indicator1 = 'x'; //Rotating the selections 90 degrees around the center of the network depending on the flow direction.
        char direction_indicator2 = 'y';
        if (directionn == 1) {
          direction_indicator1 = 'y';
          direction_indicator2 = 'x';
        }

        for (int boundn = 0; boundn < gridsize; boundn++) {

          //inlet side selections
          model.component("comp1").geom("geom1").create("boxsel"+boxselcount, "BoxSelection");
          model.component("comp1").geom("geom1").feature("boxsel"+boxselcount).set("entitydim", 2);
          model.component("comp1").geom("geom1").feature("boxsel"+boxselcount).set(direction_indicator1+"max", -0.99+fluidoffset);
          model.component("comp1").geom("geom1").feature("boxsel"+boxselcount).set(direction_indicator2+"min", 2*boundn-1.1+fluidoffset);
          model.component("comp1").geom("geom1").feature("boxsel"+boxselcount).set(direction_indicator2+"max", 2*boundn+1.1+fluidoffset);
          model.component("comp1").geom("geom1").feature("boxsel"+boxselcount).set("condition", "inside");

```

```
            model.component("comp1").geom("geom1").feature("boxsel"+boxselcount).set("
                contributeto", "csel2"); //contribute to cumulative selection used for inlets

            boxsellist1.add(boxselcount);
            boxselcount++;

            //outlet side selections
            model.component("comp1").geom("geom1").create("boxsel"+boxselcount, "BoxSelection"
                );
            model.component("comp1").geom("geom1").feature("boxsel"+boxselcount).set("
                entitydim", 2);
            model.component("comp1").geom("geom1").feature("boxsel"+boxselcount).set(
                direction_indicator1+"min", gridsize-1+"*D+0.99"+fluidoffset);
            model.component("comp1").geom("geom1").feature("boxsel"+boxselcount).set(
                direction_indicator2+"min", 2*boundn-1.1+fluidoffset);
            model.component("comp1").geom("geom1").feature("boxsel"+boxselcount).set(
                direction_indicator2+"max", 2*boundn+1.1+fluidoffset);
            model.component("comp1").geom("geom1").feature("boxsel"+boxselcount).set("
                condition", "inside");

            model.component("comp1").geom("geom1").feature("boxsel"+boxselcount).set("
                contributeto", "csel3"); //contribute to cumulative selection used for inlets

            boxsellist2.add(boxselcount);
            boxselcount++;

          }
          //JFrame jFrame = new JFrame();
          //JOptionPane.showMessageDialog(jFrame, surfaceIntegrals);


          // ADD Material properties: water & CO2
          model.component("comp1").material().create("mat1", "Common");

          if (fluidn == 0) {
            model.component("comp1").material("mat1").label("H2O");
            model.component("comp1").material("mat1").propertyGroup("def").set("density", new
```

```
            String[]{"1000"});
        model.component("comp1").material("mat1").propertyGroup("def").set("
            dynamicviscosity", new String[]{"0.00100935"});
      }

      if (fluidn == 1) {
        model.component("comp1").material("mat1").label("H2O");
        model.component("comp1").material("mat1").propertyGroup("def").set("density", new
            String[]{"1000"});
        model.component("comp1").material("mat1").propertyGroup("def").set("
            dynamicviscosity", new String[]{"0.00100935"});
        //If NWP is CO2
        //model.component("comp1").material("mat1").label("CO2");
        //model.component("comp1").material("mat1").propertyGroup("def").set("density",
            new String[]{"1.84104"});
        //model.component("comp1").material("mat1").propertyGroup("def").set("
            dynamicviscosity", new String[]{"0.0000146885"});
      }

      ArrayList<Integer> selectn = new ArrayList<Integer>();
      selectn.add(1);
      selectn.add(2);
      selectn.add(3);
      selectn.add(4);

      String path_filename = "";

      if (fluidn == 0) {
        if (directionn == 0) {
          path_filename = "\\P"+pn+"\\x_paths_water";
        }
        if (directionn == 1) {
          path_filename = "\\P"+pn+"\\y_paths_water";
        }
      }
      if (fluidn == 1) {
        if (directionn == 0) {
```

```
            path_filename = "x_paths_gas";
          }
          if (directionn == 1) {
            path_filename = "y_paths_gas";
          }
        }

        String[][] connected_indexes = loadFileToArray(gridsize, "Seed␣"+seedn+"\\"+
            path_filename+".txt");


        // set symmetry on top surface
        model.component("comp1").physics("spf").create("sym1", "Symmetry", 2);
        model.component("comp1").physics("spf").feature("sym1").selection().named("
            geom1_csel1_bnd");

        int counter = 1;

        for (Integer boxsel_n = 0; boxsel_n < boxsellist1.size(); boxsel_n++) {
          for (int i = 0; i < connected_indexes[0].length; i++) {

            int path_index = Integer.parseInt(connected_indexes[0][i])/2;

            if (boxsel_n == path_index) {

              //set inlets
              //JFrame jFrame = new JFrame();
              //JOptionPane.showMessageDialog(jFrame, boxsel_n.toString()+connected_indexes
                  [0][i]);

              model.component("comp1").physics("spf").create("inl"+counter, "InletBoundary",
                   2);
              model.component("comp1").physics("spf").feature("inl"+counter).set("
                  BoundaryCondition", "Pressure");
              model.component("comp1").physics("spf").feature("inl"+counter).set("p0",
                  gridsize);
              model.component("comp1").physics("spf").feature("inl"+counter).selection().
```

```
                named("geom1_boxsel"+boxsellist1.get(boxsel_n));
            //set outlets
            model.component("comp1").physics("spf").create("out"+counter, "OutletBoundary", 2);
            model.component("comp1").physics("spf").feature("out"+counter).selection().named("geom1_boxsel"+boxsellist2.get(boxsel_n));
            counter++;
            break;
          }
        }
      }
      //box selection used in mesh creation
      model.component("comp1").geom("geom1").create("boxsel"+boxselcount, "BoxSelection");
      model.component("comp1").geom("geom1").feature("boxsel"+boxselcount).set("entitydim", 2);
      model.component("comp1").geom("geom1").feature("boxsel"+boxselcount).set("zmax", 0.99);
      model.component("comp1").geom("geom1").feature("boxsel"+boxselcount).set("xmin", -0.99);
      model.component("comp1").geom("geom1").feature("boxsel"+boxselcount).set("xmax", 2*gridsize-1.001);
      model.component("comp1").geom("geom1").feature("boxsel"+boxselcount).set("ymin", -0.99);
      model.component("comp1").geom("geom1").feature("boxsel"+boxselcount).set("ymax", 2*gridsize-1.001);
      //set mesh
      model.component("comp1").mesh("mesh1").automatic(false);
      model.component("comp1").mesh("mesh1").feature("size").set("hauto", 9);
```

```

        model.component("comp1").mesh("mesh1").create("size1", "Size");
        model.component("comp1").mesh("mesh1").feature("size1").set("hauto", 9);
        model.component("comp1").mesh("mesh1").feature("size1").selection().named("
            geom1_boxsel"+boxselcount);
        model.component("comp1").mesh("mesh1").feature("size1").set("table", "cfd");

        model.component("comp1").mesh("mesh1").create("ftet1", "FreeTet");

        //Solve
        model.study().create("std1");
        model.study("std1").create("stat", "Stationary");
        model.study("std1").feature("stat").activate("spf", true);

        model.sol().create("sol1");
        model.sol("sol1").study("std1");
        model.study("std1").feature("stat").set("notlistsolnum", 1);
        model.study("std1").feature("stat").set("notsolnum", "1");
        model.study("std1").feature("stat").set("listsolnum", 1);
        model.study("std1").feature("stat").set("solnum", "1");
        model.sol("sol1").create("st1", "StudyStep");
        model.sol("sol1").feature("st1").set("study", "std1");
        model.sol("sol1").feature("st1").set("studystep", "stat");
        model.sol("sol1").create("v1", "Variables");
        model.sol("sol1").feature("v1").set("control", "stat");
        model.sol("sol1").create("s1", "Stationary");
        model.sol("sol1").feature("s1").set("stol", 0.2);
        model.sol("sol1").feature("s1").feature("aDef").set("cachepattern", true);
        model.sol("sol1").feature("s1").create("fc1", "FullyCoupled");
        model.sol("sol1").feature("s1").feature("fc1").set("dtech", "auto");
        model.sol("sol1").feature("s1").feature("fc1").set("initstep", 0.01);
        model.sol("sol1").feature("s1").feature("fc1").set("minstep", 1.0E-4);
        model.sol("sol1").feature("s1").feature("fc1").set("maxiter", 100);
        model.sol("sol1").feature("s1").create("i1", "Iterative");
        model.sol("sol1").feature("s1").feature("i1").set("linsolver", "gmres");
        model.sol("sol1").feature("s1").feature("i1").set("prefuntype", "left");
        model.sol("sol1").feature("s1").feature("i1").set("itrestart", 50);
```

```
        model.sol("sol1").feature("s1").feature("i1").set("rhob", 20);
        model.sol("sol1").feature("s1").feature("i1").set("maxlinit", 1000);
        model.sol("sol1").feature("s1").feature("i1").set("nlinnormuse", "on");
        model.sol("sol1").feature("s1").feature("i1").label("AMG, fluid flow variables (spf)");
        model.sol("sol1").feature("s1").feature("i1").create("mg1", "Multigrid");
        model.sol("sol1").feature("s1").feature("i1").feature("mg1").set("prefun", "saamg");
        model.sol("sol1").feature("s1").feature("i1").feature("mg1").set("mgcycle", "v");
        model.sol("sol1").feature("s1").feature("i1").feature("mg1").set("maxcoarsedof", 80000);
        model.sol("sol1").feature("s1").feature("i1").feature("mg1").set("strconn", 0.02);
        model.sol("sol1").feature("s1").feature("i1").feature("mg1").set("nullspace", "constant");
        model.sol("sol1").feature("s1").feature("i1").feature("mg1").set("usesmooth", false);
        model.sol("sol1").feature("s1").feature("i1").feature("mg1").set("saamgcompwise", true);
        model.sol("sol1").feature("s1").feature("i1").feature("mg1").set("loweramg", true);
        model.sol("sol1").feature("s1").feature("i1").feature("mg1").feature("pr").create("va1", "Vanka");
        model.sol("sol1").feature("s1").feature("i1").feature("mg1").feature("pr").feature("va1")
          .set("linesweeptype", "ssor");
        model.sol("sol1").feature("s1").feature("i1").feature("mg1").feature("pr").feature("va1").set("iter", 0);
        model.sol("sol1").feature("s1").feature("i1").feature("mg1").feature("pr").feature("va1").set("vankarelax", 0.8);
        model.sol("sol1").feature("s1").feature("i1").feature("mg1").feature("pr").feature("va1")
          .set("vankavars", new String[]{"comp1_p"});
        model.sol("sol1").feature("s1").feature("i1").feature("mg1").feature("pr").feature("va1").set("seconditer", 1);
        model.sol("sol1").feature("s1").feature("i1").feature("mg1").feature("pr").feature("va1").set("relax", 0.5);
        model.sol("sol1").feature("s1").feature("i1").feature("mg1").feature("po").create("va1", "Vanka");
        model.sol("sol1").feature("s1").feature("i1").feature("mg1").feature("po").feature("
```

```
va1")
  .set("linesweeptype", "ssor");
model.sol("sol1").feature("s1").feature("i1").feature("mg1").feature("po").feature("va1").set("iter", 1);
model.sol("sol1").feature("s1").feature("i1").feature("mg1").feature("po").feature("va1").set("vankarelax", 0.8);
model.sol("sol1").feature("s1").feature("i1").feature("mg1").feature("po").feature("va1")
  .set("vankavars", new String[]{"comp1_p"});
model.sol("sol1").feature("s1").feature("i1").feature("mg1").feature("po").feature("va1").set("seconditer", 2);
model.sol("sol1").feature("s1").feature("i1").feature("mg1").feature("po").feature("va1").set("relax", 0.5);
model.sol("sol1").feature("s1").feature("i1").feature("mg1").feature("cs").create("d1", "Direct");
model.sol("sol1").feature("s1").feature("i1").feature("mg1").feature("cs").feature("d1")
  .set("linsolver", "pardiso");
model.sol("sol1").feature("s1").feature("i1").feature("mg1").feature("cs").feature("d1")
  .set("pivotperturb", 1.0E-13);
model.sol("sol1").feature("s1").create("d1", "Direct");
model.sol("sol1").feature("s1").feature("d1").set("linsolver", "pardiso");
model.sol("sol1").feature("s1").feature("d1").set("pivotperturb", 1.0E-13);
model.sol("sol1").feature("s1").feature("d1").label("Direct, fluid flow variables (spf)");
model.sol("sol1").feature("s1").feature("fc1").set("linsolver", "i1");
model.sol("sol1").feature("s1").feature("fc1").set("dtech", "auto");
model.sol("sol1").feature("s1").feature("fc1").set("initstep", 0.01);
model.sol("sol1").feature("s1").feature("fc1").set("minstep", 1.0E-4);
model.sol("sol1").feature("s1").feature("fc1").set("maxiter", 100);
model.sol("sol1").feature("s1").feature().remove("fcDef");
model.sol("sol1").attach("std1");

//create velocity plot
model.result().dataset("dset1").set("geom", "geom1");
model.result().create("pg1", "PlotGroup3D");
```

```java
            model.result("pg1").label("Velocity␣(spf)");
            model.result("pg1").set("frametype", "spatial");
            model.result("pg1").set("data", "dset1");
            model.result("pg1").feature().create("slc1", "Slice");
            model.result("pg1").feature("slc1").label("Slice");
            model.result("pg1").feature("slc1").set("smooth", "internal");
            model.result("pg1").feature("slc1").set("data", "parent");

            //create pressure plot

            model.result().create("pg2", "PlotGroup3D");
            model.result("pg2").label("Pressure␣(spf)");
            model.result("pg2").set("frametype", "spatial");

            model.result("pg2").feature().create("con1", "Contour");
            model.result("pg2").feature("con1").label("Pressure");
            model.result("pg2").feature("con1").set("expr", "p");
            model.result("pg2").feature("con1").set("number", 20);
            model.result("pg2").feature("con1").set("levelrounding", true);
            model.result("pg2").feature("con1").set("data", "dset1");
            model.result("pg2").feature("con1").set("contourtype", "filled");

            model.sol("sol1").runAll();

            //create horizontal velocity plane
            model.result("pg1").feature("slc1").set("quickplane", "xy");
            model.result("pg1").feature("slc1").set("quickznumber", 1);
            model.result("pg1").feature("slc1").set("interactive", true);

            String sliceHeight = "-3.8E-7";

            if (fluidn == 1) {
              sliceHeight = "5.0E-7";
            }
            model.result("pg1").feature("slc1").set("shift", sliceHeight);

            //create surface integral east side
```

```
model.result().numerical().create("int1", "IntSurface");
model.result().numerical("int1").set("intvolume", true);

if (fluidn == 0) {
  model.result().numerical("int1").selection().named("geom1_csel2_bnd");
}
if (fluidn == 1) {
  model.result().numerical("int1").selection().named("geom1_csel3_bnd");
}

model.result("pg1").run();

//save result to text file
model.result().numerical("int1").setIndex("expr", "spf.U", 0);
model.result().table().create("tbl1", "Table");
model.result().table("tbl1").comments("Surface␣Integration␣2");
model.result().numerical("int1").set("table", "tbl1");
model.result().numerical("int1").setResult();
model.result().table("tbl1")
  .save("C:\\Users\\eobbens\\Master␣Thesis␣Grids\\Gridsize␣"+gridsize+"\\Seed␣"+
      seedn+"\\P"+pn+"\\table␣"+directionn+"␣"+fluidn+".txt");



//export first image
model.result().export().create("img1", "pg1", "Image");
model.result().export("img1").set("size", "manualweb");
model.result().export("img1").set("zoomextents", true);
model.result().export("img1").set("options3d", true);
model.result().export("img1").set("title3d", false);
model.result().export("img1").set("grid", false);
model.result().export("img1").set("axisorientation", false);
model.result().export("img1").set("logo3d", false);
model.result().export("img1").set("lockview", "on");
model.result().export("img1").set("view", "view2");

```

```
        model.result().export("img1")
          .set("pngfilename", "C:\\Users\\eobbens\\Master␣Thesis␣Grids\\Gridsize␣"+gridsize+
              "\\Seed␣"+seedn+"\\P"+pn+"\\Velocity␣Profile␣"+directionn+"␣"+fluidn+".png");
        model.result().export("img1").run();



        //export second image
        model.result().export().create("img2", "pg2", "Image");
        model.result().export("img2").set("size", "manualweb");
        model.result().export("img2").set("zoomextents", true);
        model.result().export("img2").set("options3d", true);
        model.result().export("img2").set("title3d", false);
        model.result().export("img2").set("grid", false);
        model.result().export("img2").set("axisorientation", false);
        model.result().export("img2").set("logo3d", false);
        model.result().export("img2").set("lockview", "on");
        model.result().export("img2").set("view", "view2");

        model.result().export("img2")
          .set("pngfilename", "C:\\Users\\eobbens\\Master␣Thesis␣Grids\\Gridsize␣"+gridsize+
              "\\Seed␣"+seedn+"\\P"+pn+"\\Pressure␣Profile␣"+directionn+"␣"+fluidn+".png");
        model.result().export("img2").run();



        if (deletenodes == true) {
          model.component().remove("comp1");
          model.study().remove("std1");
          model.result().table().remove("tbl1");
          model.result().export().remove("img1");
          model.result().remove("pg1");
          model.result().numerical().remove("int1");
          model.result().remove("pg2");
          model.result().remove("pg1");
          model.result().export().remove("img2");
          model.result().dataset().remove("surf1");
          model.result().dataset().remove("dset1");
        }
```

```

              }
              catch (Exception e) {
                JOptionPane.showMessageDialog(null, ""+e);
                model.component().remove("comp1");
                model.study().remove("std1");
                model.result().table().remove("tbl1");
                model.result().export().remove("img1");
                model.result().remove("pg1");
                model.result().numerical().remove("int1");
                model.result().remove("pg2");
                model.result().remove("pg1");
                model.result().export().remove("img2");
                model.result().dataset().remove("surf1");
                model.result().dataset().remove("dset1");

                continue;
              }

            } //directionn


          } //fluidn

        } //pn
      } //seed
    }
  }
}
```

Code: Creation of Fig. 22 with the data generated in COMSOL

```
import numpy as np
import matplotlib as mpl
import matplotlib.pyplot as plt
import seaborn as sns
```

```
import os
import pandas as pd
from scipy.stats import hmean

#figure resolution
mpl.rcParams['figure.dpi']= 375

sns.set_style("whitegrid")
sns.set_style(rc={'ytick.left': True})

mpl.rc('font',family='Times New Roman')
mpl.rcParams['mathtext.fontset'] = 'custom'
mpl.rcParams['mathtext.rm'] = 'Times New Roman'
mpl.rcParams['mathtext.it'] = 'Times New Roman:italic'
mpl.rcParams['mathtext.bf'] = 'Times New Roman:bold'

mpl.rcParams.update({'figure.autolayout': True})

font_size = 30

directory = 'Network images brdif'
all_folders = [folder[0] for folder in os.walk(directory)]

df_water = pd.DataFrame()
df_gas = pd.DataFrame()
grid_n = 0
seed_n = 0

p_n = 0

info_dict = {}

for folder in all_folders:
    info_path = folder + '\\info.txt'
    if os.path.exists(info_path):
        with open(info_path,'r') as fin:
            grid_n = int(folder[30])
```

```
            seed_n = int(folder[37])
            lines = fin.readlines()

    for i in range(2):
        for j in range(2):
            fpath = folder + f'\\table_{i}_{j}.txt'
            if os.path.exists(fpath):
                with open(fpath ,'r') as fin:
                    lines = fin.readlines()
                    flow_measurement = float(lines[5][:-1])

                    grid_n = int(fpath[30])
                    seed_n = int(fpath[37])
                    p_n = int(fpath[40])
                    #
                    if i ==0:
                        direction = 'hor'

                    if i == 1:
                        direction = 'ver'

                    #fluid = 'water'
                    if j == 0:
                        df_water = df_water.append(({'Grid':grid_n,
                                                     'Seed':seed_n,
                                                     'P_n':p_n,
                                                     'Direction':direction,
                                                     'Q_water':flow_measurement}),ignore_index=True)


                    if j == 1:
                        #fluid = 'gas'
                        Q = 'Q_gas'
                        df_gas = df_gas.append(({'Grid':grid_n,
                                                 'Seed':seed_n,
                                                 'P_n':p_n,
                                                 'Direction':direction,
```

```
                                                 'Q_gas':flow_measurement}),ignore_index=True)

df_water = df_water.reindex(columns= ['Grid','Seed','P_n', 'Direction','Q_water'])

df_gas = df_gas.reindex(columns= ['Grid','Seed','P_n', 'Direction','Q_gas'])

df = pd.concat([df_water, df_gas], axis=1)#.drop_duplicates()

df = df.loc[:,~df.columns.duplicated()]

#flow rate through a microfluidic device of gridsize 1 filled completely with one fluid (water or
    co2)
fullwater = 2.1473E-16
fullgas = 1.4755E-14
df['Grid'] = df['Grid']*4 #The Gridsize Folder names were divided by 4 to give the number one index,
     e.g. "Gridsize 4" becomes "Gridsize 1".
#df['Q_gas_full'] = df['Grid'] * fullgas
df['Q_water_full'] = df['Grid'] * fullwater

df['gas_rel_perm'] = df['Q_gas']/df['Q_water_full']
df['water_rel_perm'] = df['Q_water']/df['Q_water_full']

df['rel_perm_frac'] = df['water_rel_perm']/df['gas_rel_perm']

df['P_n'] = df['P_n']/10

fign= 0

figsize = (6,6)

tick_list = [0.1,0.2,0.3]

legend_fontsize = 24
plt.rcParams['legend.title_fontsize'] = legend_fontsize
### WETTING PHASE
plt.figure(fign,figsize=figsize)
sns.scatterplot(data=df,x='P_n',y='water_rel_perm',color='grey',alpha=0.5)
```

```

plt.figure(fign,figsize=figsize)
sns.regplot(data=df,x='P_n',y='water_rel_perm',x_estimator=np.mean, ci=None,fit_reg=False,color='green',label='Arithmetic')

fig = plt.figure(fign,figsize=figsize)
ax = sns.regplot(data=df,x='P_n',y='water_rel_perm',x_estimator=hmean, ci=None,fit_reg=False,color='red', label='Harmonic')
plt.title('$k_{rel,WP}$',size=font_size)
plt.xlabel('$\Delta p_{network}$',fontsize=font_size)
plt.xticks(size=font_size)
plt.yticks(size=font_size)
plt.yscale('log')

plt.ylim([10e-5,10e0])

ax.set_xticks(tick_list)
#ax.set_yticks(np.geomspace(10e-4, 10e-2 ,31))

#plt.legend(title = "Average")
fig.savefig('gz12WP.png')
fign += 1

### NON WETTING PHASE
plt.figure(fign,figsize=figsize)
sns.scatterplot(data=df,x='P_n',y='gas_rel_perm',color='grey',alpha=0.5)

plt.figure(fign,figsize=figsize)
sns.regplot(data=df,x='P_n',y='gas_rel_perm',x_estimator=np.mean, ci=None,fit_reg=False,color='green',label='Arithmetic')

fig = plt.figure(fign,figsize=figsize)
ax = sns.regplot(data=df,x='P_n',y='gas_rel_perm',x_estimator=hmean, ci=None,fit_reg=False,color='red', label='Harmonic')
plt.title('$k_{rel,NWP}$',size=font_size)
plt.xlabel('$\Delta p_{network}$',fontsize=font_size)
plt.xticks(size=font_size)
```

```python
plt.yticks(size=font_size)
plt.yscale('log')
plt.ylim([10e-5,10e0])
ax.set_xticks(tick_list)
#plt.legend(title = "Average",loc=4)
fig.savefig('gz12NWP.png')
fign += 1

### Fraction of rel perms
plt.figure(fign,figsize=figsize)
sns.scatterplot(data=df,x='P_n',y='rel_perm_frac',color='grey',alpha=0.5)

plt.figure(fign,figsize=figsize)
sns.regplot(data=df,x='P_n',y='rel_perm_frac',x_estimator=np.mean, ci=None,fit_reg=False,color='
    green',label='Arithmetic')

fig = plt.figure(fign,figsize=figsize)
ax = sns.regplot(data=df,x='P_n',y='rel_perm_frac',x_estimator=hmean, ci=None,fit_reg=False,color='
    red', label='Harmonic')
plt.title('$k_{rel,WP}$ / $k_{rel,NWP}$',size=font_size)
plt.xlabel('$\Delta p_{network}$',fontsize=font_size)
plt.xticks(size=font_size)
plt.yticks(size=font_size)
plt.yscale('log')
plt.ylim([10e-5,10e0])
ax.set_xticks(tick_list)
plt.legend(title = "Average",loc=4,fontsize = legend_fontsize)
fig.savefig('gz12frac.png')
fign += 1
```

References

L. Alonso, G. Bradley, S. J. Cox., and S. Hutzler. Flow through borders and vertices in foam drainage. Poster presented at EUFoam 2002 Conference. 2002.

B. R. Bird, W. E. Stewart, E. N. Lightfoot, and D. J. Klingenberg. *Introductory Transport Phenomena*. Wiley, 2014.

K. Brakke. The surface evolver. *Exp. Math. Volume 1, 1992 - Issue 2*, 1992. doi: https://doi.org/10.1080/10586458.1992.10504253.

COMSOL. Comsol multiphysics® microfluidics module users guide v. 5.6. 2020.

S. J. Cox, A. Davarpanah, and W. R. Rossen. Challenges for microfluidic devices in representing flow in geological formations. *poster presentation at Interpore 2022 conference. (Manuscript in preparation)*, 2022.

W. Drenckhan, H. Ritacco, A. Saint-Jalmes, A. Saugey, P. McGuinness, A. van der Net, D. Langevin, and D. Weaire. Fluid dynamics of rivulet flow between plates. *Physics of Fluids 19, 102101*, 2007. doi: https://doi.org/10.1063/1.2757153.

M. E. Fisher. Critical probabilities for cluster size and percolation problems. *J. Math Phys., 2 620–627*, 1961. doi: https://doi.org/10.1201/9781482272444.

G. Hadjisotiriou. Fluid conductivity of steady two-phase flow in a 2d micromodel. BSc thesis, Delft U. of Technology. 2020.

G. Hirasaki. Personal communication. 2022.

J. Holstvoogd. Analysis of steady multiphase flow in porous media. BSc thesis, Delft U. of Technology. 2020.

K. Li, M. Sharifnik, K.-H. A. Wolf, and W. R. Rossen. Coarsening of foam in two model fractures with different roughness. *Colloids and Surfaces A: Physicochemical and Engineering Aspects*, 631:127666, 2021. ISSN 0927-7757. doi: https://doi.org/10.1016/j.colsurfa.2021.127666.

E. J. M. Obbens. Steady-state two-phase flow conductance in a 2d micromodel. BSc thesis, Delft U. of Technology. 2020.

W. R. Rossen. *Foams in Enhanced Oil Recovery. In R. K. Prud'homme S. A. Khan (Eds.), Foams theory, measurements, and applications (pp. 413–445).* Marcel Dekker, Inc., 1996.

M. Sahimi. *Applications Of Percolation Theory.* Taylor and Francis, 2014. doi: https://doi.org/10.1201/9781482272444.