

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	iii
List of Tables	v
1 General Introduction	1
2 Artificial Fracture Topic	2
2.1 Introduction	2
2.2 Method	4
2.3 Results	7
2.4 Discussion	8
2.5 Conclusion	9
3 Microfluidic Device Topic	10
3.1 Introduction	10
3.2 Method	11
3.3 Results	15
3.4 Discussion	17
3.5 Conclusion	18
A Appendix	19
B Appendix	26
References	102

List of Figures

1	The roughness pattern that is applied to the glass plate for Model 1 and Model 2 (Li et al., 2021).	2
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 $t = 0.09\text{ h}$ the image on the right is of Model 2 and taken at $t = 0.10\text{ h}$. The water is shown in white and the gas in black. The dimensions of the images are $7.8 \times 6.8\text{ mm}$ for Model 1 and $12.3 \times 9.8\text{ mm}$ for Model 2.	3
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.	3
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.	4
5	The resulting velocity contours (color scale) and pressure field (height) (a) for Model 1 at $t = 0.09\text{ hr}$ and (b) Model 2 at $t = 0.10\text{ hr}$. In this example the fluid flows in the negative x-direction. A pressure difference of 10% of the capillary pressure is assumed here.	5
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.	5
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.	6
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.	7
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.	7
10	The pressure equalization duration for both models with a zero-velocity interfacial BC.	8
11	The pressure equalization duration for both models with a zero-stress interfacial BC.	8
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.	11
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.	12
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.	12

15	Side view of the geometries shown in Fig. 14.	13
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.	13
17	Flow chart of the network arrangement procedure.	14
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.	15
19	Pressure and velocity plots for left-right flow in a lattice of 8x8 pillars.	15
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.	16
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.	16
22	Average relative permeabilities with differing amounts of bridges in networks of lattice size 12x12.	17
23	Images of Model 1 Li et al. (2021).	19
24	Images of Model 2 Li et al. (2021).	21
25	Cox bridge.	23
26	Modified bridge.	23
27	Cox straight.	24
28	Modified straight.	24
29	Cox corner.	24
30	Modified corner.	24
31	Lower half of a bridge with a circular gap.	25
32	Lower half of a bridge with the modified gap.	25

List of Tables

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 $Q_{network,PB}$	20
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 $Q_{network,PB}$	22
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.	23

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.

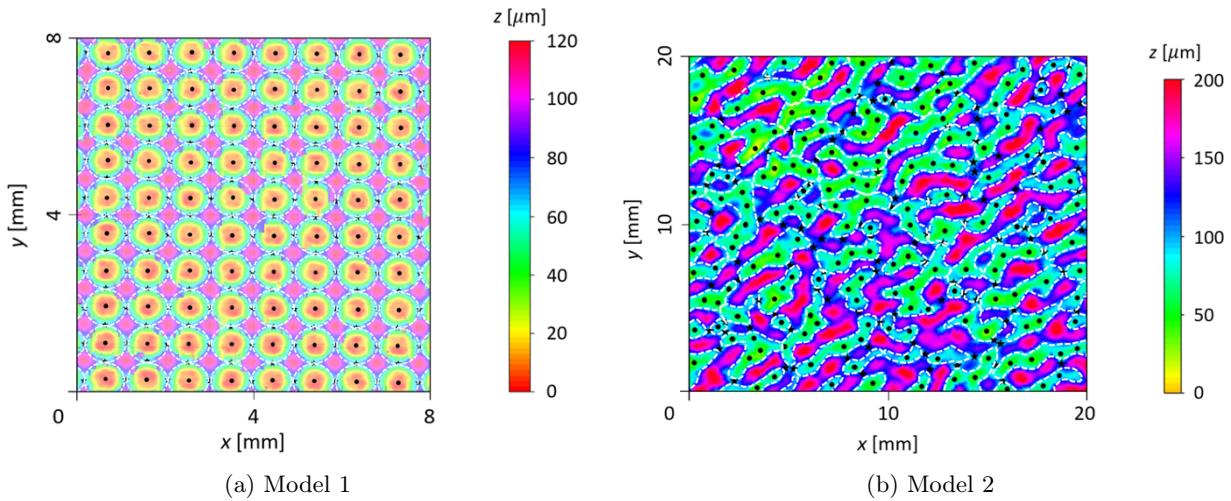
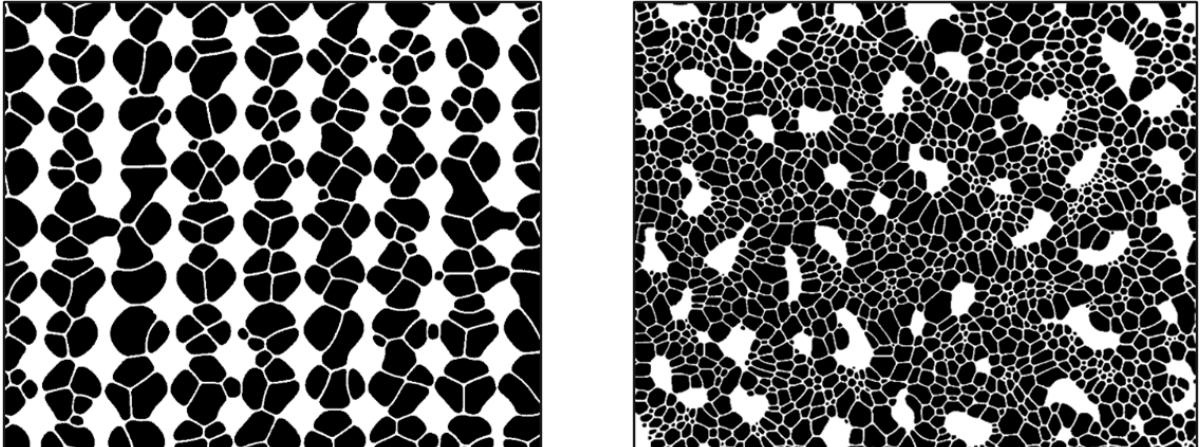


Figure 1: The roughness pattern that is applied to the glass plate for Model 1 and Model 2 (Li et al., 2021).

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 $t = 1.75\text{ h}$ because these WP clusters in the narrow locations became too small to calculate a capillary pressure, and the image of Model 2 at $t = 17.10\text{ h}$ because the water zones appear on the image to be disconnected from one side to another.

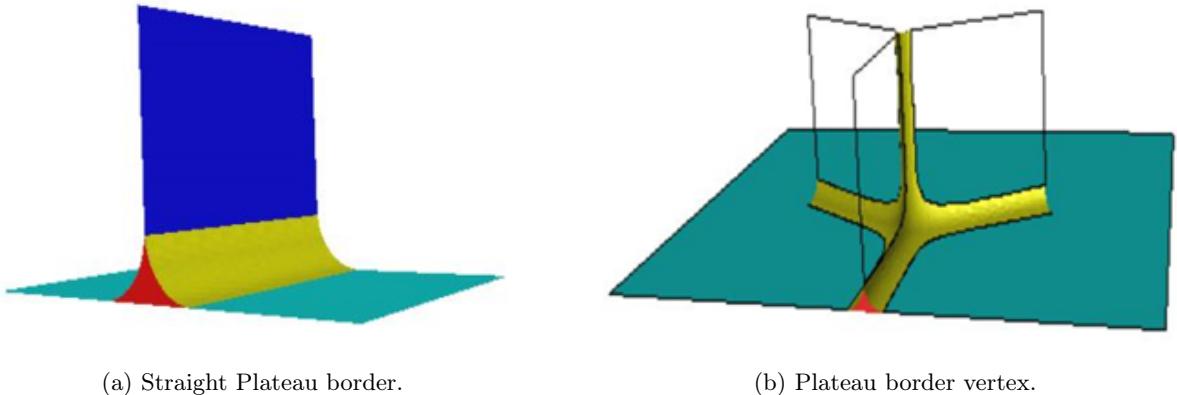
Although most lamellae appear to be nearly straight in the images, there is variation in the curvature of the lamellae. The smallest radius of curvature is approximately 1 mm for both models. The capillary pressure is roughly uniform within each image if the wetting phase is able to flow through the 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.

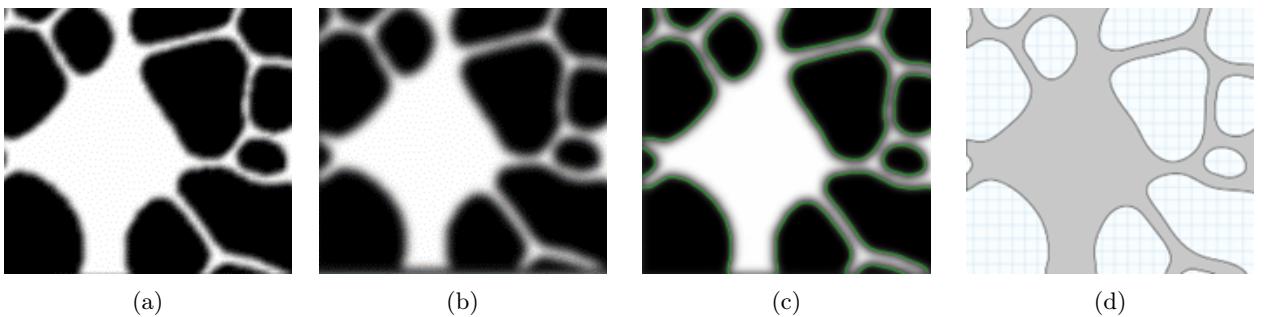


Figure 4: Detail of the bottom left corner of Model 1 at $t=0.09$ hours. (a) The original image. The resolution of the image causes the interfacial boundaries to appear jagged. (b) Gaussian filter is applied to smooth-out the image. (c) Contours (green lines) are detected at the interfacial boundaries. (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/m³ 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:

$$\text{Scaling Factor} = \frac{Q_{PB}}{Q_{slit}} \quad (1)$$

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.

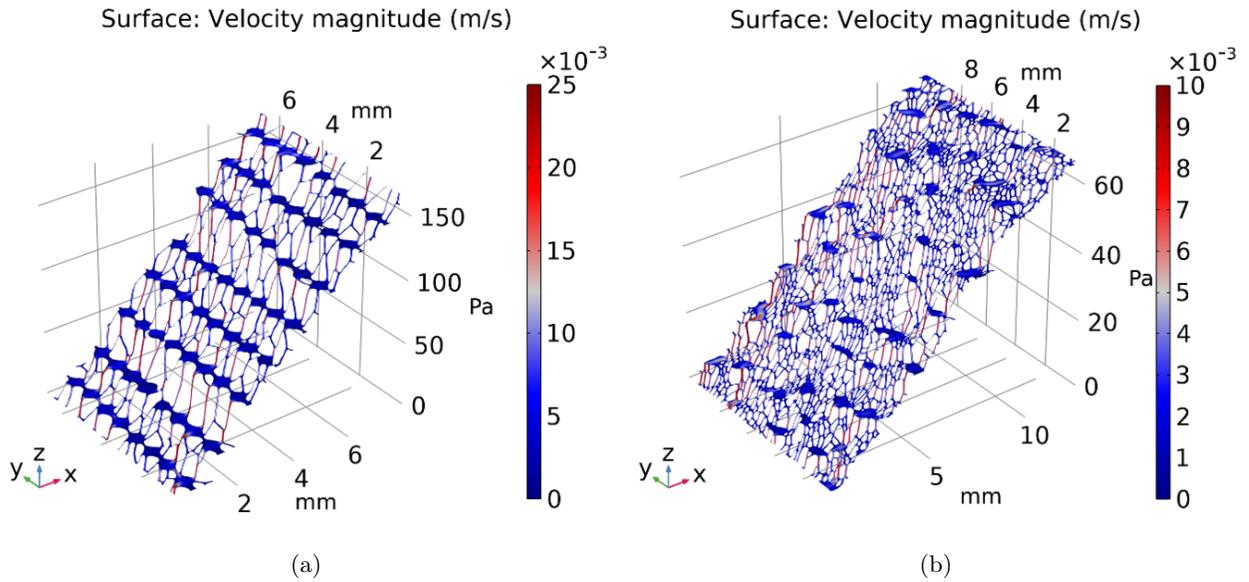


Figure 5: The resulting velocity contours (color scale) and pressure field (height) (a) for Model 1 at $t = 0.09$ hr and (b) Model 2 at $t = 0.10$ hr. In this example the fluid flows in the negative x-direction. A pressure difference of 10% of the capillary pressure is assumed here.

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

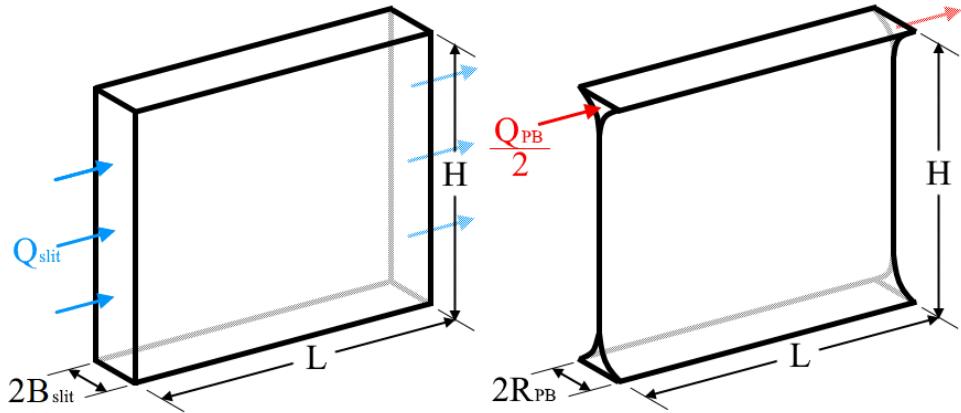


Figure 6: Schematic of the flow through a slit and flow through the Plateau borders. If width of the slit and Plateau borders is similar, then the Plateau borders constrict the flow of the liquid more.

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.

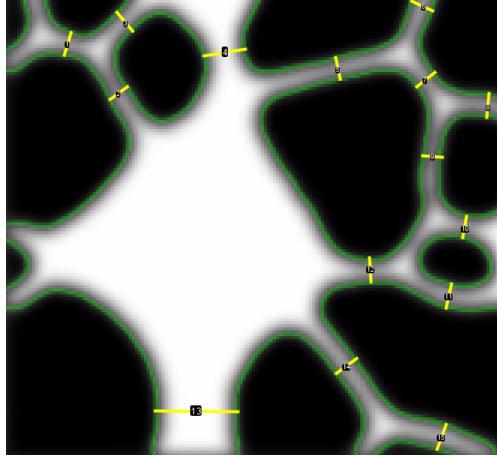


Figure 7: An example of a slit width measurements using ImageJ. The yellow lines indicate the extent of each measurement. These lines are manually drawn in the middle of the slits starting and ending on the contours recognized in COMSOL Multiphysics®. The measurements result in a width and an angle towards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

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

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.

$$P_c = \frac{\sigma}{R_{PB}} \quad (3)$$

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.

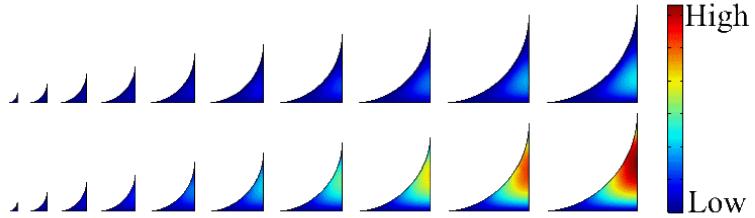


Figure 8: Velocity magnitude (m/s) through a Plateau border with an interfacial zero velocity BC (upper row) and an interfacial zero stress BC (lower row). Velocity is zero on the bottom (glass plate) and there is zero shear stress along the right, due to symmetry. The fluid flows in or out of the page.

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.

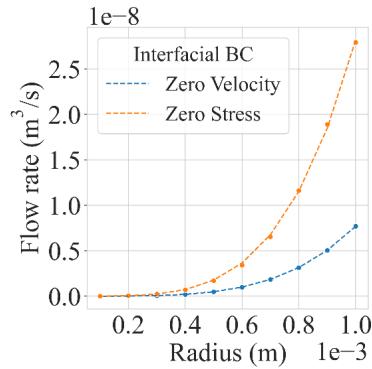


Figure 9: The flow rate values calculated for plateau borders (consisting of 4 times the shapes in Fig. 9) with a radius ranging from 0.1 to 1 mm and a pressure of 1 Pa.

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

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

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

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.

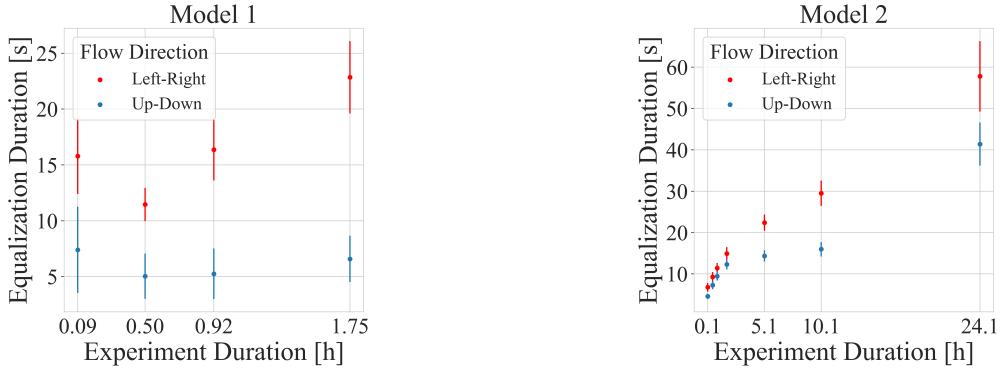


Figure 10: The pressure equalization duration for both models with a zero-velocity interfacial BC.

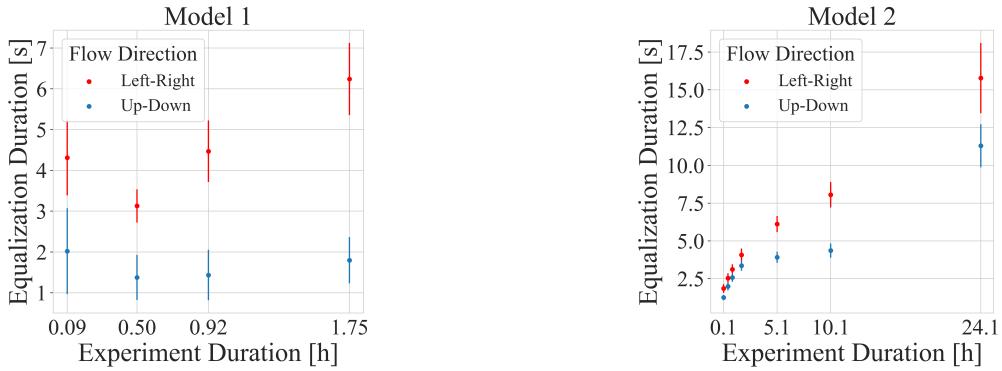


Figure 11: The pressure equalization duration for both models with a zero-stress interfacial BC.

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.

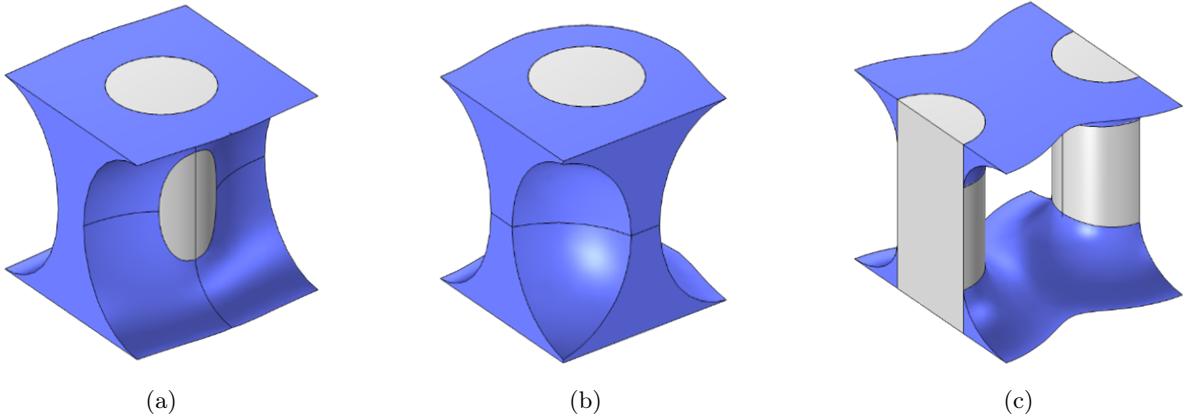


Figure 12: Interface shapes created with Surface Evolver for WP to flow around pore throats or pore bodies occupied by NWP. The grey volume represents cylindrical pillars (grains) in the microfluidic device. The blue volume is occupied by the WP, and the NWP occupies the remaining space.

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.

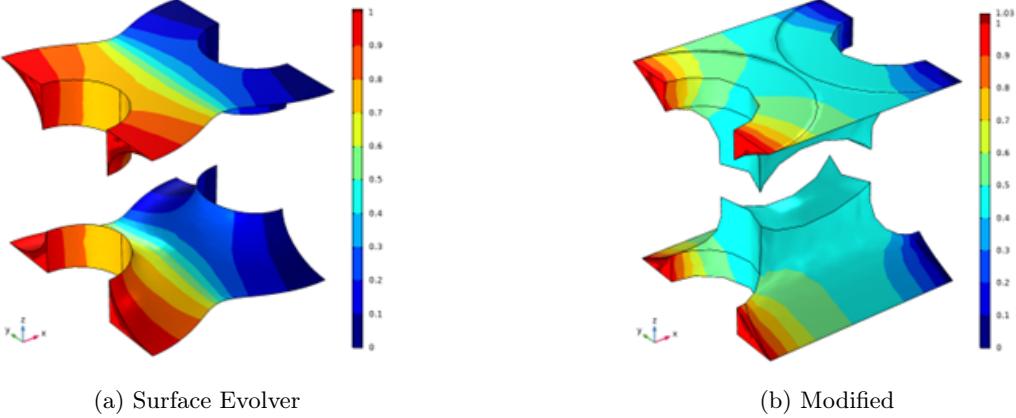


Figure 13: Pressure profiles for (a) WP flow through the bridge occupied pore throat for the geometry determined with Surface Evolver and (b) the geometry used in our network calculations. The largest pressure drop in the geometry determined with Surface Evolver is located in the middle of the pore throat. 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.

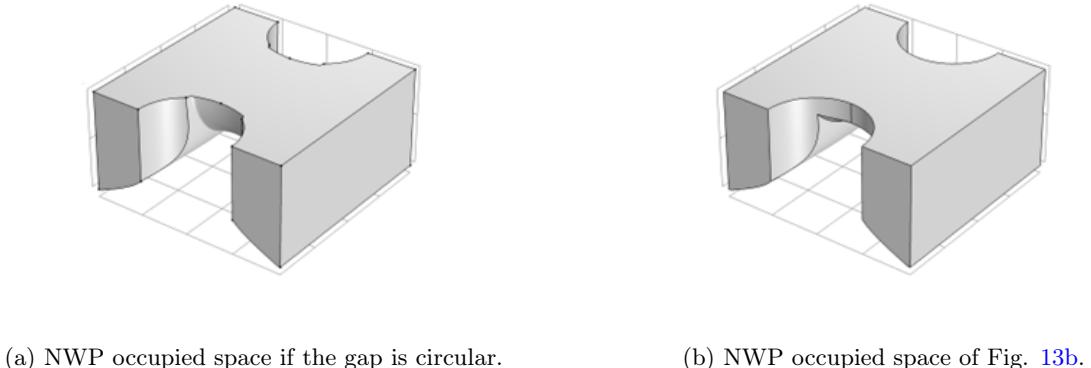
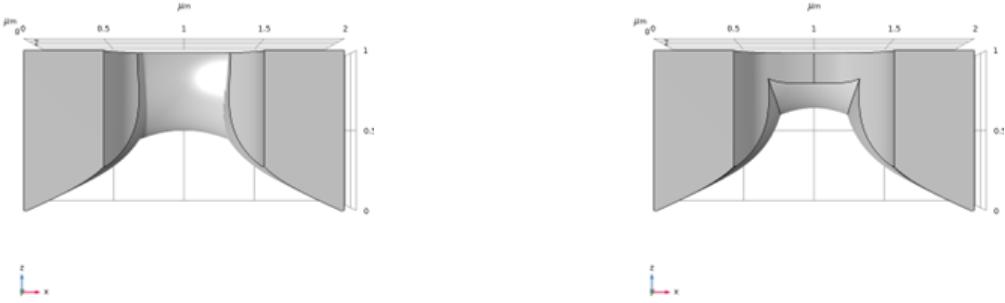


Figure 14: Lower half of the space occupied by the NWP as it flows through the gap in a pore throat with a WP bridge. The lower half is shown because it is symmetrical and details would be obscured otherwise, i.e., where the NWP channel is cylindrical. (a) Geometry with the smallest possible gap for 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.

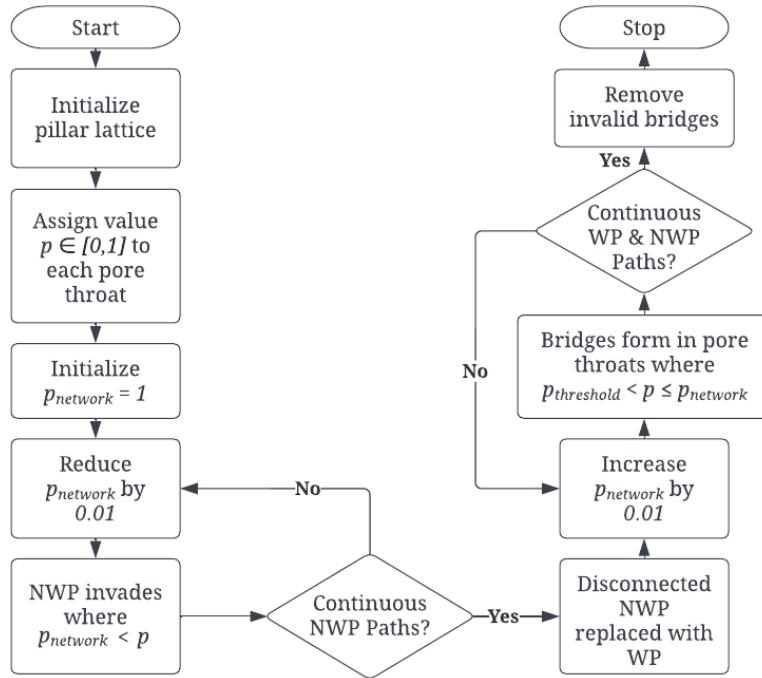


Figure 17: Flow chart of the network arrangement procedure.

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.

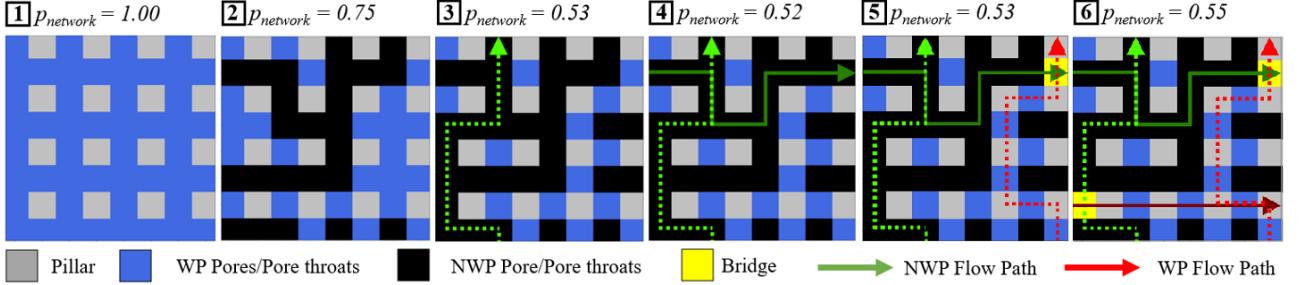


Figure 18: Schematic showing how the distribution of the interfaces is determined for a network with the p-value distribution of Fig. 16. Each pore throat is assigned a random value: $0 < p < 1$. (1) The device is saturated by WP. (2 & 3) NWP invades pores when $p_{network} < p$. (4) NWP flows from left→right & 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.

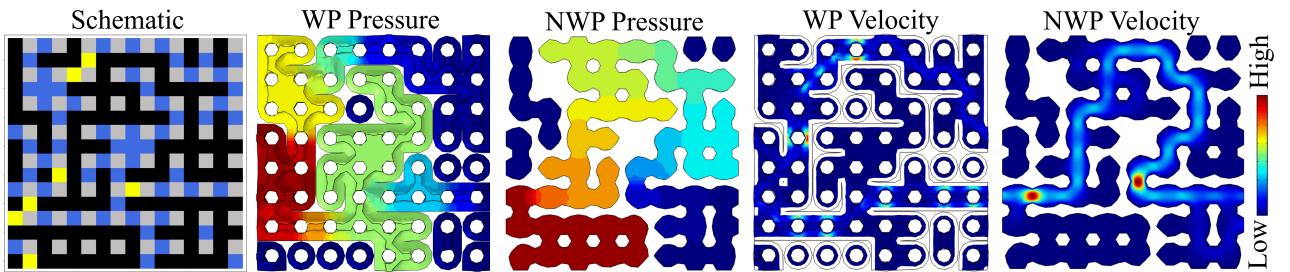


Figure 19: Pressure and velocity plots for left-right flow in a lattice of 8x8 pillars.

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.

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

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.

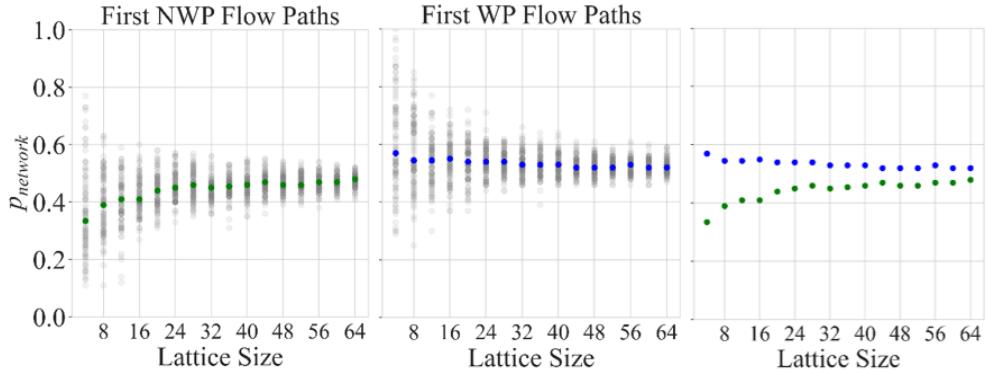


Figure 20: The results of determining the median $p_{network}$ values for when the NWP and WP are able to form continuous flow paths across a network. For a lattice size of 64 the median $p_{network}$ values for continuous flow paths are 0.48 for the NWP and 0.52 for the WP.

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.

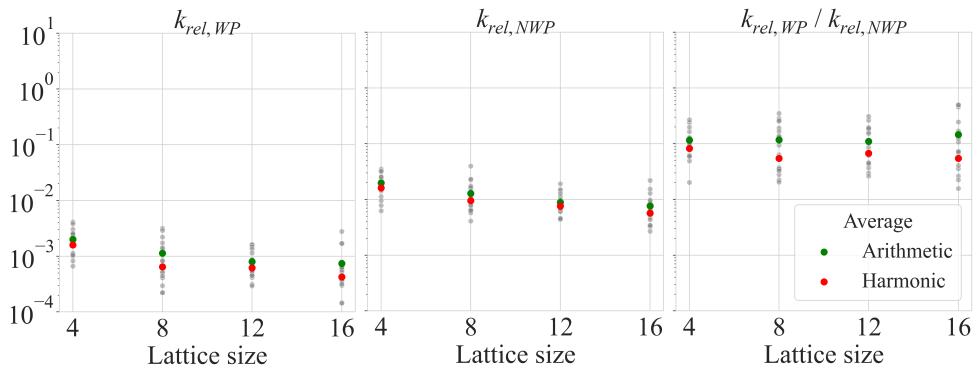


Figure 21: Average relative permeabilities calculated for networks differing in lattice size and the least amount of bridges necessary for flow paths to form in both directions across the network. 20 calculations are done for each lattice size.

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.

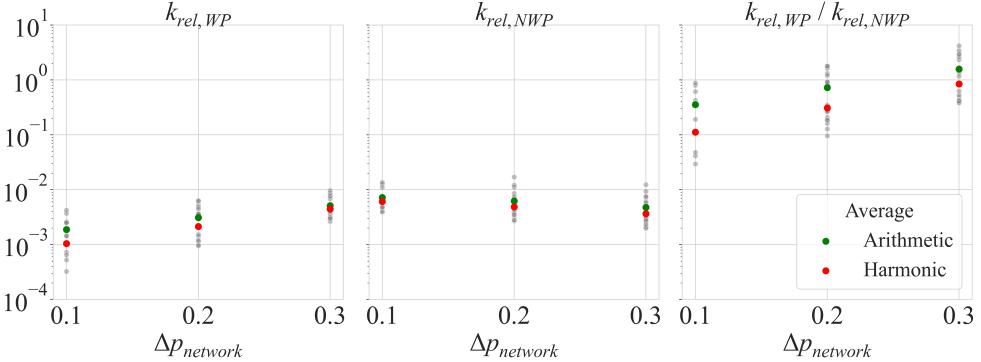


Figure 22: Average relative permeabilities with differing amounts of bridges in networks of lattice size 12x12.

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 restrictive 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

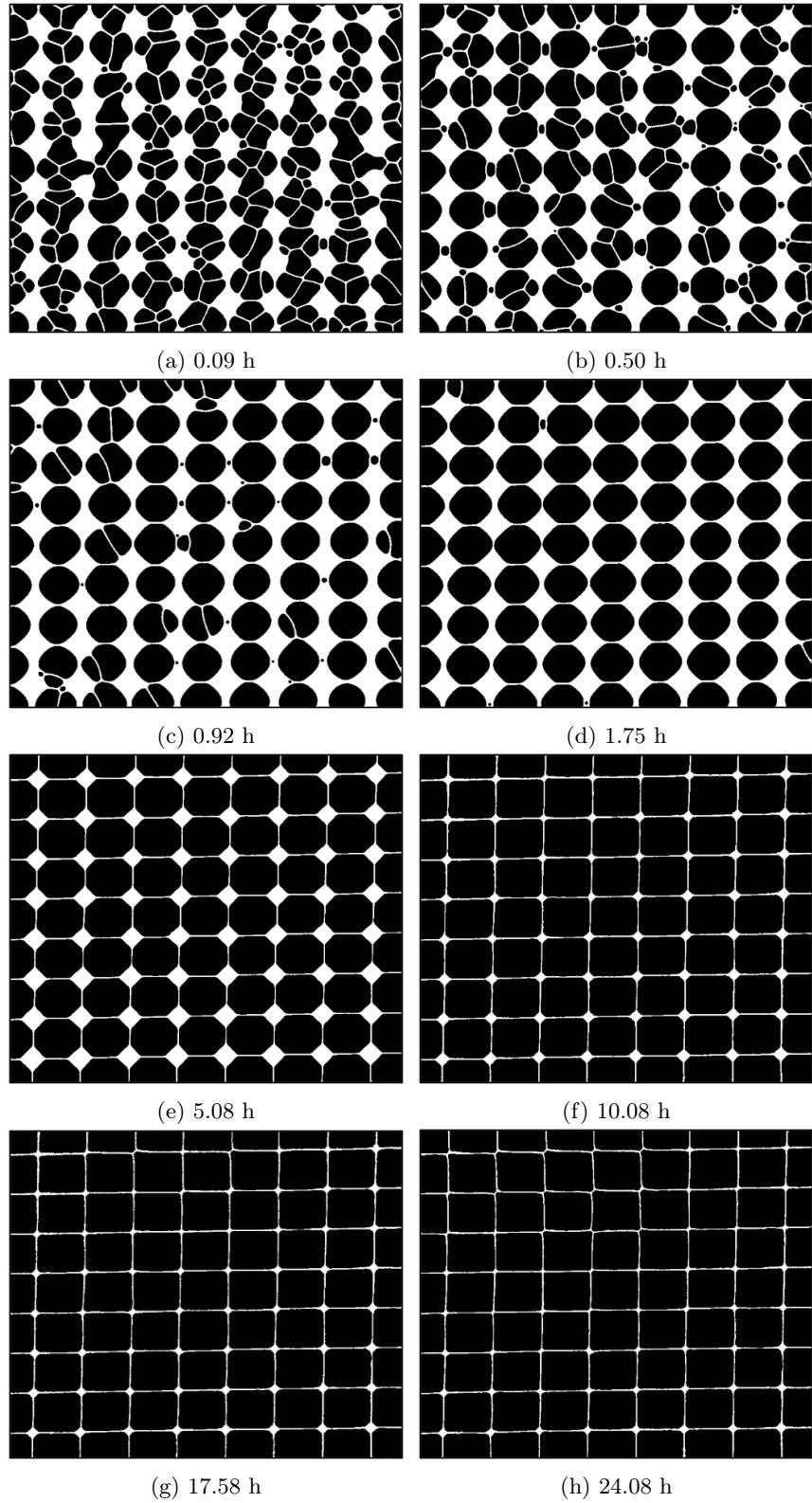


Figure 23: Images of Model 1 [Li et al. \(2021\)](#).

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. $Q_{network,PB}$ 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

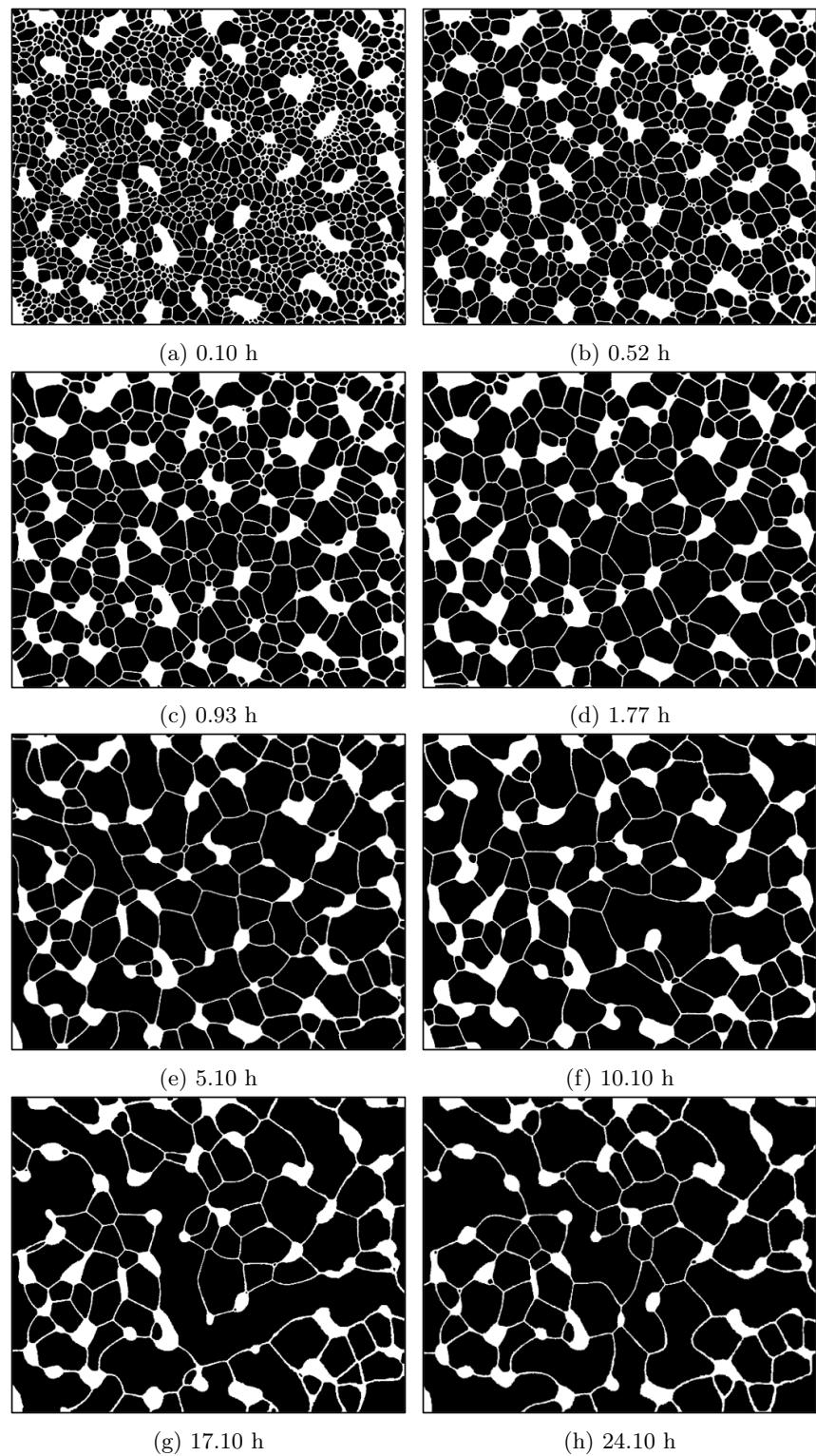


Figure 24: Images of Model 2 [Li et al. \(2021\)](#).

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. $Q_{network,PB}$ 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 $T = 293.15\text{ K}$, $P = 1\text{ atm}$, $\mu = 10^{-4}\text{ Pa} \cdot \text{s}$, $W = L = H = 2\text{ }\mu\text{m}$ and $\Delta P = 1\text{ Pa}$. In the network calculations the pillars have a diameter of $1\text{ }\mu\text{m}$ 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				

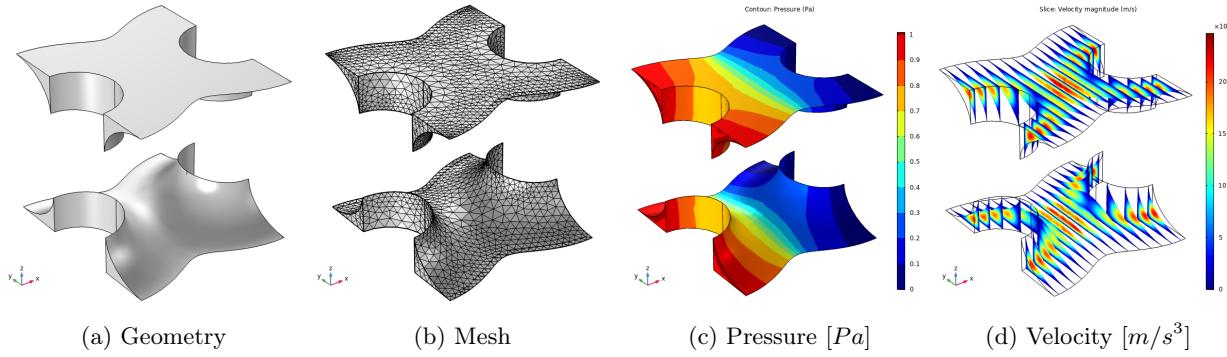


Figure 25: Cox bridge.

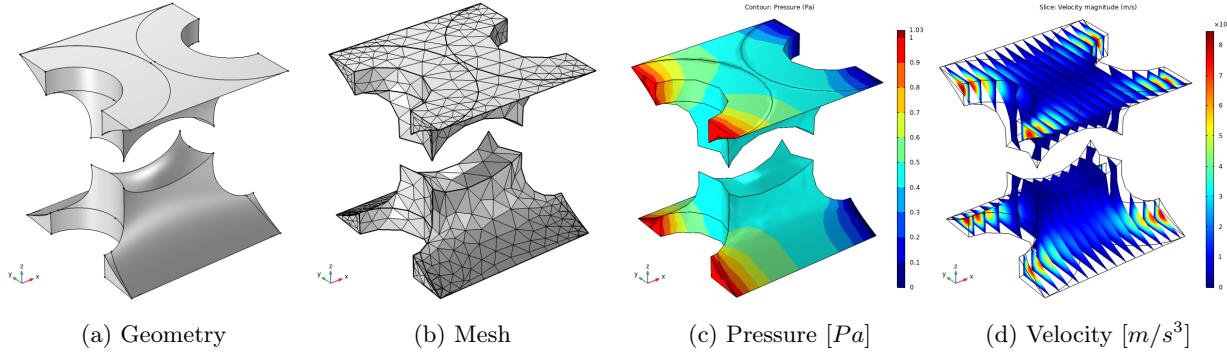


Figure 26: Modified bridge.

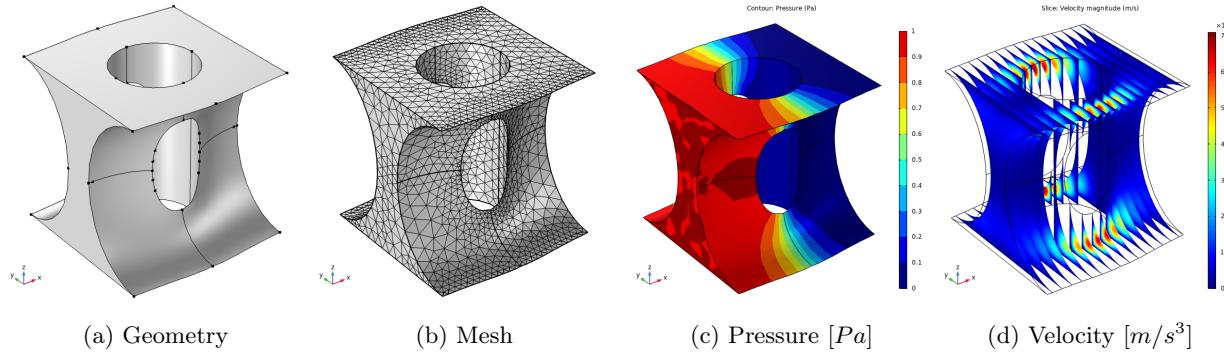


Figure 27: Cox straight.

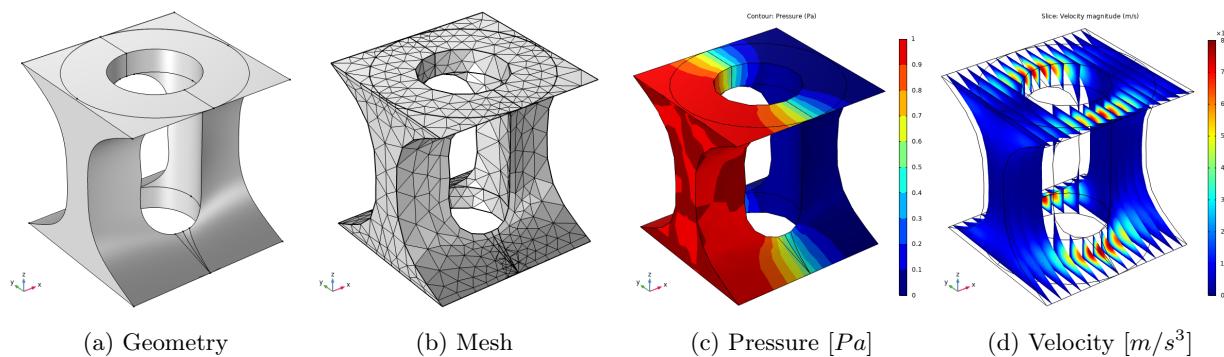


Figure 28: Modified straight.

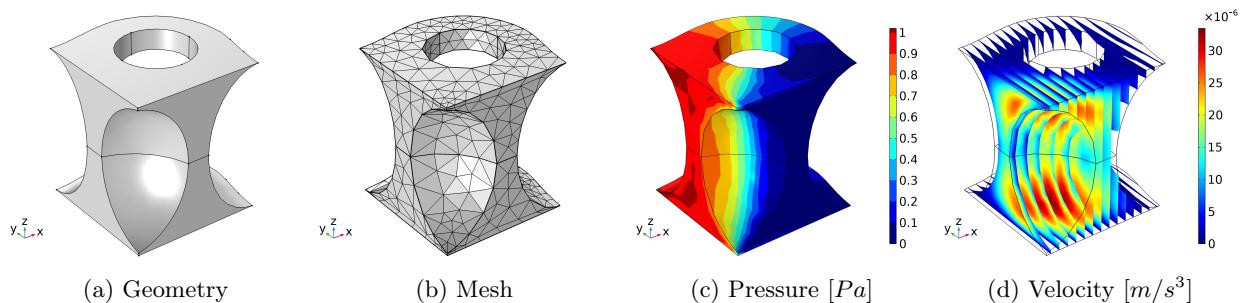


Figure 29: Cox corner.

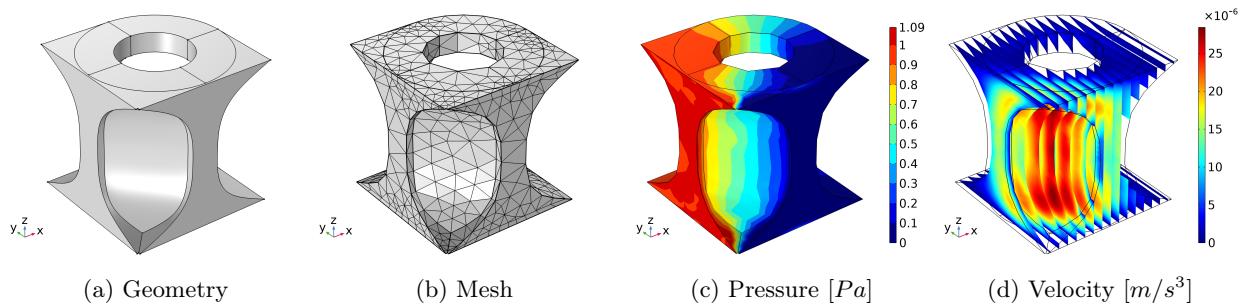


Figure 30: Modified corner.

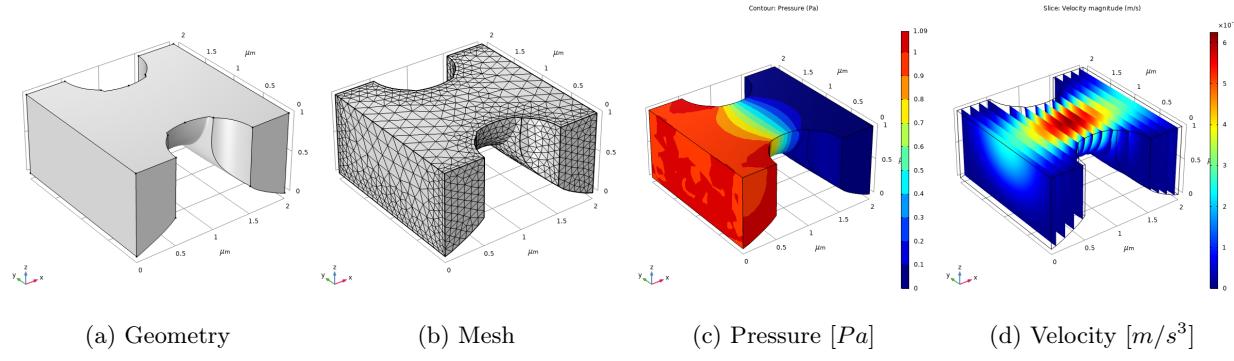


Figure 31: Lower half of a bridge with a circular gap.

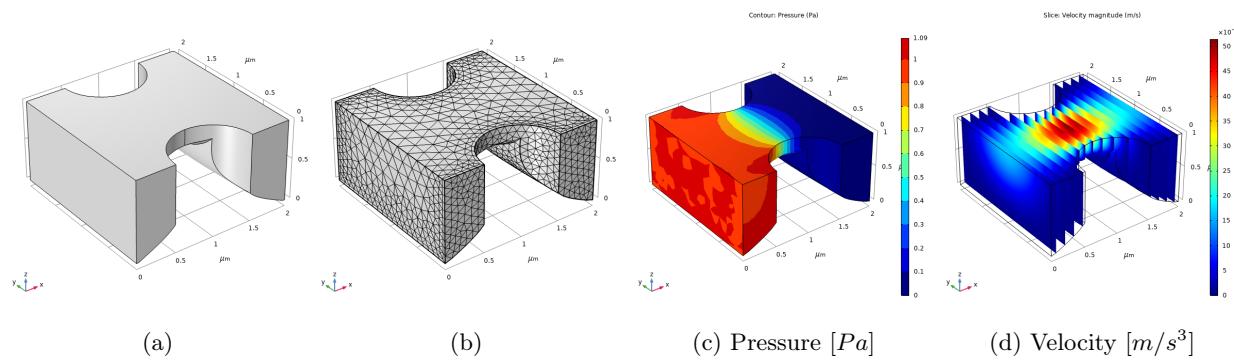


Figure 32: Lower half of a bridge with the modified gap.

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.

```
1 import numpy as np
2 import matplotlib.pyplot as plt
3 from scipy.optimize import curve_fit
4 from IPython.display import display, Latex
5 import seaborn as sns
6 import matplotlib as mpl
7
8 mpl.rcParams['figure.dpi']= 375
9
10 sns.set_style("whitegrid")
11 sns.set_style(rc={'ytick.left': True})
12
13 mpl.rc('font',family='Times_New_Roman')
14 mpl.rcParams['mathtext.fontset'] = 'custom'
15 mpl.rcParams['mathtext.rm'] = 'Times_New_Roman'
16 mpl.rcParams['mathtext.it'] = 'Times_New_Roman:italic'
17 mpl.rcParams['mathtext.bf'] = 'Times_New_Roman:bold'
18
19 mpl.rcParams.update({'figure.autolayout': True})
20
21 L = 0.001
22 visc = 0.0010000247767419923
23
24 Radius = np.array([0.001,0.0009,0.0008,0.0007,0.0006,0.0005,0.0004,0.0003,0.0002,0.0001])
25 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])
26 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])
27
28 No_Slip_Flow *= 4 # 4 times that flow rate in 1 PB
29 Slip_Flow *= 4
```

```

30
31 def func(x,a):
32     return a*x**4
33
34 parameters = curve_fit(func,Radius,No_Slip_Flow)
35 parameters2 = curve_fit(func,Radius, Slip_Flow)
36
37 display(Latex(f'Q_{no\_slip}={parameters[0][0]} \cdot R^4_{(PB)}'))
38 display(Latex(f'Q_{slip}={parameters2[0][0]} \cdot R^4_{(PB)}'))
39
40 plt.figure(0, figsize =[6,6])
41
42 ax = sns.scatterplot(Radius,No_Slip_Flow)
43
44 plt.plot(Radius,func(Radius,parameters[0][0]),linestyle='dashed')
45
46 sns.scatterplot(Radius,Slip_Flow)
47
48 plt.plot(Radius,func(Radius,parameters2[0][0]),linestyle='dashed')
49
50 plt.legend(['Zero_Velocity','Zero_Stress'],title='Interfacial BC',fontsize=24)
51 plt.rcParams['legend.title_fontsize'] = 24
52 plt.xlabel('Radius_(m)',fontsize=30)
53 plt.ylabel('Flow_rate_(m$^3$/s)',fontsize=30)
54 plt.xticks(size=30)
55 plt.yticks(size=30)
56 plt.ticklabel_format(axis="x", style="sci", scilimits=(0,0))
57 ax.yaxis.offsetText.set_fontsize(30)
58 ax.xaxis.offsetText.set_fontsize(30)
59 plt.savefig('PBflow.png')
60
61 print(parameters[0][0]*L*visc)
62 print(parameters2[0][0]*L*visc)

```

Code: Calculations Artificial Fracture Topic

```

1 import numpy as np; import pandas as pd; import matplotlib as mpl; import matplotlib.pyplot as plt;
    import seaborn as sns
2
3 pd.set_option("display.max_columns", None)
4 pd.set_option('display.float_format', '{:.2E}'.format())
5
6 mpl.rcParams["figure.dpi"] = 375
7
8 sns.set_style("whitegrid")
9 sns.set_style(rc={"ytick.left": True})
10
11 mpl.rc("font", family="Times_New_Roman")
12 mpl.rcParams["mathtext.fontset"] = "custom"
13 mpl.rcParams["mathtext.rm"] = "Times_New_Roman"
14 mpl.rcParams["mathtext.it"] = "Times_New_Roman:italic"
15 mpl.rcParams["mathtext.bf"] = "Times_New_Roman:bold"
16 mpl.rcParams.update({"figure.autolayout": True})
17
18 hydraulic_aperture_model1 = 4.6E-5      # [m]
19 hydraulic_aperture_model2 = 7.8E-5      # [m]
20 density_fluid           = 998.0585     # [kg/m^3] Standard COMSOL water material at 293.15 K.
21 viscosity                = 0.001000248 # [Pa*s] Standard COMSOL water material at 293.15 K.
22 surface_tension          = 0.0322       # [N/m]
23 L                      = 0.001        # [m] Length of the slit and Plateau border along the flow
    direction.
24 delta_P                 = 1            # [Pa] Pressure difference of inlet and outlet in slit/PB
    calculation.
25 volume_fraction          = 0.1          # Fraction of total water volume to flow out of the network.
26 coefficient_no_slip      = 0.007689340390189774 # Coefficients found for flow through a Plateau
    border.
27 coefficient_slip         = 0.02816471179666251
28 slip_noslip_ratio = coefficient_slip/coefficient_no_slip
29
30 data_model1 = {"time": [0.09,          0.5,          0.92,          1.75], # [hour] Time
    of image creation since start of experiment.
31         "P_c": [2010,          2210,          2460,          3370], # Capillary
    Pressure[Pa] obtained from Li et al. (2021).

```

```

32     "total_volume": [6.23E-11,           4.30E-11,           3.83E-11,           1.20E-11], # [m^3] Total
33             water volume in image.
34     "width_H":      [5.72805E-05, 5.55886E-05, 5.92850E-05, 6.36432E-05], # [m] Average
35             lamellae width measured for horizontal flow.
36     "width_H_StdDev": [1.23E-05,           7.23E-06,           1.00E-05,           9.01E-06], # Standard
37             Deviation.
38     "width_V":      [7.28896E-05, 6.39366E-05, 8.22884E-05, 8.20751E-05], # [m] Average
39             lamellae width measured for vertical flow.
40     "width_V_StdDev": [3.80E-05,           2.58E-05,           3.54E-05,           2.59E-05], # Standard
41             Deviation.
42     "Q_network_H": [1.22E-05,           1.55E-05,           1.80E-05,           1.76E-05], # [m^2/s] 2D
43             flow rate in horizontal direction.
44     "Q_network_V": [5.37E-05,           5.37E-05,           1.50E-04,           1.31E-04]} # [m^2/s] 2D
45             flow rate in vertical direction.

46
47
48 data_model2 = {"time":          [0.1,           0.52,           0.93,           1.77,           5.1,
49               10.1,          24.1],
50
51     "P_c":          [840,            890,            900,            910,            930,
52               960,          1030],
53
54     "total_volume": [1.34E-09,        1.17E-09,        1.14E-09,        9.94E-10,        8.31E-10,
55               7.20E-10,        5.05E-10],
56
57     "width_H":       [5.0399E-05, 5.39459E-05, 5.61751E-05, 6.10897E-05, 6.53889E-05,
58               6.3258E-05, 7.39536E-05],
59
60     "width_H_StdDev": [7.84E-06,        6.82E-06,        5.86E-06,        6.40E-06,        5.70E-06,
61               6.59E-06,        1.09E-05],
62
63     "width_V":       [5.00837E-05, 5.54157E-05, 5.73899E-05, 6.30362E-05, 6.35442E-05,
64               6.37526E-05, 7.38641E-05],
65
66     "width_V_StdDev": [6.99E-06,        7.39E-06,        6.40E-06,        6.25E-06,        5.99E-06,
67               6.90E-06,        9.33E-06],
68
69     "Q_network_H": [1.27E-05,        1.25E-05,        1.17E-05,        1.05E-05,        7.82E-06,
70               5.29E-06,        4.01E-06],
71
72     "Q_network_V": [1.85E-05,        1.73E-05,        1.51E-05,        1.40E-05,        1.12E-05,
73               1.00E-05,        5.58E-06]} }
```

```

53 def Calculations(data, hydraulic_aperture):
54     data_df = pd.DataFrame.from_dict(data)
55
56     def Q_slit(slit_width):
57         return (2/3) * (delta_P * (slit_width/2) ** 3 * hydraulic_aperture) / (viscosity * L)
58
59     data_df["Width_error_H"] = data_df["width_H_StDev"] / data_df["width_H"]
60     data_df["Width_error_V"] = data_df["width_V_StDev"] / data_df["width_V"]
61
62     data_df["Q_slit_H"] = Q_slit(data_df["width_H"])
63     data_df["Q_slit_V"] = Q_slit(data_df["width_V"])
64
65     data_df["R_Plateau_border"] = surface_tension / data_df["P_c"]
66
67     data_df["Q_Plateau_border"] = coefficient_no_slip/(viscosity*L) * (data_df["R_Plateau_border"]
68                           ** 4)
69
70     data_df["Q_ratio_H"] = data_df["Q_Plateau_border"] / data_df["Q_slit_H"]
71     data_df["Q_ratio_V"] = data_df["Q_Plateau_border"] / data_df["Q_slit_V"]
72
73     data_df["Q_slit_network_H"] = data_df["Q_network_H"] * hydraulic_aperture # [m^3/s]
74     data_df["Q_slit_network_V"] = data_df["Q_network_V"] * hydraulic_aperture
75
76     data_df["Q_PB_H"] = data_df["Q_slit_network_H"] * data_df["Q_ratio_H"] # 3D flow
77     corrected from slits to Plateau borders.
78     data_df["Q_PB_V"] = data_df["Q_slit_network_V"] * data_df["Q_ratio_V"]
79
80     data_df["eq._time_H"] = (volume_fraction * data_df["total_volume"]) / data_df["Q_PB_H"] #
81     [s] Pressure equalization duration.
82     data_df["eq._time_V"] = (volume_fraction * data_df["total_volume"]) / data_df["Q_PB_V"] #
83     [s] Pressure equalization duration.
84
85     data_df["eq._time_H_StDev"] = data_df["eq._time_H"] * data_df["Width_error_H"]
86     data_df["eq._time_V_StDev"] = data_df["eq._time_V"] * data_df["Width_error_V"]
87
88     return data_df
89
90
91 def plot(df, n, ylim, xticks):

```

```

86
87     ax = df.plot(kind    = "scatter",
88                  figsize= [6,6],
89                  x       = "time",
90                  y       = "eq._time_H",
91                  yerr   = "eq._time_H_StdDev",
92                  xticks = xticks,
93                  #ylim   = ylim,
94                  color  = "red")
95
96
97     df.plot(ax      = ax,
98              kind    = "scatter",
99              x       = "time",
100             y       = "eq._time_V",
101             yerr   = "eq._time_V_StdDev")
102
103     plt.title(f"Model_{n}", size = 30)
104     plt.xlabel("Experiment_Duration_[h]", fontsize = 30)
105     plt.ylabel("Equalization_Duration_[s]", fontsize = 30)
106
107     plt.xticks(xticks,fontsize=25)
108     plt.yticks(fontsize=25)
109     plt.legend(["Left-Right","Up-Down"], title = "Flow_Direction",loc=2,fontsize=20)
110     plt.rcParams["legend.title_fontsize"] = 24
111
112
113     plt.savefig(f"model{n}.png")
114
115
116     data_df1 = Calculations(data_model1, hydraulic_aperture_model1)
117     data_df2 = Calculations(data_model2, hydraulic_aperture_model2)
118
119     print(data_df1)
120     print(data_df2)

```

Code: Functions used for the arrangement of the networks

```
1 def plot_grid(figcount,title,path):
2
3     mesh = np.arange(len(Matrix))
4
5     x, y = np.meshgrid(mesh, mesh)
6
7     color = [[Matrix[i][j]['color'] for i in range(w)] for j in range(w)]
8
9     color = [item for sublist in color for item in sublist]
10
11
12
13
14 def set_water_ini():
15
16     #all bodies initially filled with water
17
18     for x in range(w)[::2]:
19
20         for y in range(h)[::2]:
21
22             Matrix[x][y] = {'throat_size':1,'type':'body','fluid':'water', 'color':water}
23
24
25 def set_throat_size():
26
27     #setting throat size depending on p
28
29     for x in range(w+1)[::2]:
30
31         for y in range(h):
32
33             if (x - 1 - (y % 2)) >= 0:
34
35                 Matrix[x - 1 - (y % 2)][y] = {'throat_size':np.random.rand(),'type':'throat','fluid':
36
37                     'water','color':water}
38
39
40 def set_gas_throats():
41
42     # setting gas throats
43
44     for x in range(w+1)[::2]:
45
46         for y in range(h):
47
48             if (x - 1 - (y % 2)) >= 0:
49
50                 if (Matrix[x - 1 - (y % 2)][y]['type'] == 'throat') and (Matrix[x - 1 - (y % 2)][y][
51
52                     'throat_size'] > p):
53
53                     Matrix[x - 1 - (y % 2)][y]['fluid'] = 'gas'
```

```

34             Matrix[x - 1 - (y % 2)][y]['color'] = gas
35
36 def set_gas_bodies():
37     #setting porebodies adjacent to gas throats to gas
38     for x in range(0,w):
39         for y in range(0,h):
40             if Matrix[x][y]['fluid'] == 'gas':
41
42                 for i in [-1,1]:
43                     if 1 < x < w - 1:
44                         if Matrix[x+i][y]['type'] == 'body':
45                             Matrix[x+i][y]['fluid'] = 'gas'
46                             Matrix[x+i][y]['color'] = gas
47
48                     if 1 < y < h - 1:
49                         if Matrix[x][y+i]['type'] == 'body':
50                             Matrix[x][y+i]['fluid'] = 'gas'
51                             Matrix[x][y+i]['color'] = gas
52
53 def place_bridges():
54     #place bridges
55     for x in range(w):
56         for y in range(h):
57             if p_bridge[0] <= round(Matrix[x][y]['throat_size'],2) <= p_bridge[1]:
58                 Matrix[x][y]['fluid'] = 'bridge'
59                 Matrix[x][y]['color'] = bridge
60
61 def bfs_opposite_side(grid, start, end, fluid): #end is 'x-direction' or 'y-direction'
62     queue = collections.deque([[start]])
63     seen = set([start])
64     while queue:
65         path = queue.popleft()
66         x, y = path[-1]
67
68         for x2, y2 in ((x+1,y), (x-1,y), (x,y+1), (x,y-1)):
69
70             if (0 <= x2 < w) and (0 <= y2 < h) and ((x2, y2) not in seen):

```

```

71
72     fluid_condition = False
73
74     if (fluid == 'gas') and (Matrix[x2][y2]['fluid'] == 'gas'):
75         fluid_condition = True
76
77     if (fluid == 'water') and (Matrix[x2][y2]['fluid'] != 'gas'):
78         fluid_condition = True
79
80     if fluid_condition is True:
81         queue.append(path + [(x2, y2)])
82         seen.add((x2, y2))
83
84     if end == 'y-direction':
85         if x == start[0] and y == h-1:
86             #for (x , y) in seen:
87             #    Matrix[x][y]['color'] = gas_path
88             return path
89
90     if end == 'x-direction':
91         if x == w-1 and y == start[1]:
92             #for (x , y) in seen:
93             #    Matrix[x][y]['color'] = gas_path
94             return path
95
96 def bfs_remove_gas(grid, start):
97     queue = collections.deque([[start]])
98     seen = set([start])
99     while queue:
100         path = queue.popleft()
101         x, y = path[-1]
102         if (x in [0,w-1]) or (y in [0,h-1]):
103             return path
104
105         for x2, y2 in ((x+1,y), (x-1,y), (x,y+1), (x,y-1)):
106             if (0 <= x2 < w) and (0 <= y2 < h) and (Matrix[x2][y2]['fluid'] == 'gas') and (x2, y2)
107                 not in seen:

```

```

107         queue.append(path + [(x2, y2)])
108
109     for x,y in seen:
110
111         Matrix[x][y]['fluid'] = 'water'
112
113         Matrix[x][y]['color'] = water
114
115         checked.append((x,y))
116
117
118     def bfs_remove_bridge(grid, start):
119
120         queue = collections.deque([[start]])
121
122         seen = set([start])
123
124         while queue:
125
126             path = queue.popleft()
127
128             x, y = path[-1]
129
130             if (x in [0,w-1]) or (y in [0,h-1]):
131
132                 return path
133
134
135             for x2, y2 in ((x+1,y), (x-1,y), (x,y+1), (x,y-1)):
136
137                 if (0 <= x2 < w) and (0 <= y2 < h) and ((Matrix[x2][y2]['fluid'] == 'gas') or (Matrix[x2]
138 ])[y2]['fluid'] == 'bridge')) and (x2, y2) not in seen:
139
140                     queue.append(path + [(x2, y2)])
141
142                     seen.add((x2, y2))
143
144             for x,y in seen:
145
146                 Matrix[x][y]['fluid'] = 'water'
147
148                 Matrix[x][y]['color'] = water
149
150                 checked.append((x,y))

```

Code: The results shown in Fig. 20

```

1 import numpy as np
2 import matplotlib
3 import matplotlib.pyplot as plt
4 import matplotlib as mpl
5 import collections
6 import json
7 import copy
8 import os

```

```

9 import seaborn as sns
10
11 mpl.rcParams["figure.dpi"] = 375
12
13 sns.set_style("whitegrid")
14 sns.set_style(rc={"ytick.left": True})
15
16 mpl.rc("font", family="Times_New_Roman")
17 mpl.rcParams["mathtext.fontset"] = "custom"
18 mpl.rcParams["mathtext.rm"] = "Times_New_Roman"
19 mpl.rcParams["mathtext.it"] = "Times_New_Roman:italic"
20 mpl.rcParams["mathtext.bf"] = "Times_New_Roman:bold"
21
22 mpl.rcParams.update({"figure.autolayout": True})
23
24 font_size = 30
25
26 p_dict = {}
27 grid_range = range(4, 68, 4)
28 xtick_range = range(8, 72, 8)
29 p_water_medians = []
30 p_gas_medians = []
31
32 #colors
33 pillar = "silver"
34 pillar_path = "whitesmoke"
35 water = "royalblue"
36 water_path = "orange"      #orange
37 gas = "black"
38 gas_path = "lightgreen"   #lightgreen for paths
39 bridge = "yellow"
40 snap_off = "red"
41
42 for i in grid_range:#,20,22,24,26,28,30,32,34,36,38,40]
43     gridsize = i
44     gridsize_int = int(gridsize/2)  #amount of pillars
45

```

```

46     fign = 0
47
48     p_gas_list = []
49
50     p_water_list = []
51
52
53
54     #Initial Matrix
55
56     w, h = gridsize, gridsize #w = width
57
58     Matrix = [{ "throat_size":0, "type": "pillar", "fluid": "pillar", "color": pillar} for x in range
59
60         (w) ] for y in range(h) ]
61
62
63     #all bodies initially filled with water
64
65     for x in range(w) [::2]:
66
67         for y in range(h) [::2]:
68
69             Matrix[x][y] = { "throat_size":1, "type": "body", "fluid": "water", "color": water}
70
71
72     #setting throat size depending on p
73
74     for x in range(w+1) [::2]:
75
76         for y in range(h):
77
78             if (x - 1 - (y %2)) >= 0:
79
80                 Matrix[x - 1 - (y %2)][y] = { "throat_size":np.random.rand(), "type": "throat",
81
82                     "fluid": "water", "color": water}
83
84
85
86
87
88
89
90     for p in np.round(np.arange(0.1,1,0.01),2) [::-1]:
91
92         # setting gas throats
93
94         for x in range(w+1) [::2]:
95
96             for y in range(h):
97
98                 if (x - 1 - (y %2)) >= 0:
99
100                     if (Matrix[x - 1 - (y %2)][y][ "type"] == "throat") and (Matrix[x - 1 - (y
101
102                         %2)][y][ "throat_size"] > p):
103
104                         Matrix[x - 1 - (y %2)][y][ "fluid"] = "gas"
105
106                         Matrix[x - 1 - (y %2)][y][ "color"] = gas
107
108
109
110                     #setting porebodies adjacent to gas throats to gas

```

```

80         for x in range(0,w):
81             for y in range(0,h):
82                 if Matrix[x][y]["fluid"] == "gas":
83
84                     for i in [-1,1]:
85                         if 0 <= x <= w - 1:
86                             ix = i
87
88                             if x == 0:
89                                 ix = 1
90
91                             if x == w -1:
92                                 ix = -1
93
94                             if Matrix[x+ix][y]["type"] == "body":
95
96                                 Matrix[x+ix][y]["fluid"] = "gas"
97
98                                 Matrix[x+ix][y]["color"] = gas
99
100
101                     if y == h - 1:
102                         iy = -1
103
104                         if Matrix[x][y+iy]["type"] == "body":
105
106                             Matrix[x][y+iy]["fluid"] = "gas"
107
108                             Matrix[x][y+iy]["color"] = gas
109
110
111                     if 0 <= y <= h - 1:
112                         if y == 0:
113                             i = 1
114
115                         if y == h - 1:
116                             i = -1
117
118                         if Matrix[x][y+i]["type"] == "body":
119
120                             Matrix[x][y+i]["fluid"] = "gas"
121
122                             Matrix[x][y+i]["color"] = gas
123
124
125                     if y == 0: # make sure boundary has pore bodies adjacent
126                         horizontally

```

```

116             ix = i
117
118             if x == w - 1:
119
120                 ix = -1
121
122                 if Matrix[x+ix][y]["type"] == "body":
123
124                     Matrix[x+ix][y]["fluid"] = "gas"
125
126                     Matrix[x+ix][y]["color"] = gas
127
128
129             # make sure wrap around boundary condition is satisfied
130
131             if Matrix[w - 1][y]["fluid"] == "gas":
132
133                 Matrix[0][y]["type"] == "body"
134
135                 Matrix[0][y]["fluid"] = "gas"
136
137                 Matrix[0][y]["color"] = gas
138
139
140             if Matrix[x][h-1]["fluid"] == "gas":
141
142                 Matrix[x][0]["type"] == "body"
143
144                 Matrix[x][0]["fluid"] = "gas"
145
146                 Matrix[x][0]["color"] = gas
147
148
149
150
151             if not x_path_exists_check:
152

```

```

153     for y in range(0,h):
154
155         if (Matrix[0][y]["fluid"] == "gas"):
156
157             x_dir = bfs_opposite_side(Matrix, (0,y), "x-direction", "gas")
158
159
160             if x_dir is not None:
161
162                 x_path_coord_list_gas.append(y)
163
164                 x_path_exists_check = True
165
166                 x_dir_list_gas.append(x_dir)
167
168
169             if x_path_exists_check and y_path_exists_check:
170
171                 #remove gas
172
173                 checked = []
174
175                 for x in range(w):
176
177                     for y in range(h):
178
179                         bfs_remove_gas(Matrix, (x,y))
180
181
182                     #set pillars back
183
184
185                     for x in range(w)[1::2]:
186
187                         for y in range(h)[1::2]:
188
189                             Matrix[x][y] = {"throat_size":0, "type":"pillar", "fluid":"pillar", "color":
190
191                                 pillar}
192
193
194
195                     for y_dir in y_dir_list_gas:
196
197                         for (i,j) in y_dir:
198
199                             Matrix[i][j]["color"] = gas_path
200
201                     for x_dir in x_dir_list_gas:
202
203                         for (i,j) in x_dir:
204
205                             Matrix[i][j]["color"] = gas_path
206
207
208
209                     #create textfile for information and add p of the first gas flow paths
210
211                     if not os.path.exists(f"Network_images_final/Gridsize_{gridsize_int}/Seed_{seed}"):
212
213                         os.makedirs(f"Network_images_final/Gridsize_{gridsize_int}/Seed_{seed}")

```

```

189     with open(f"Network_images_final/Gridsize_{gridsize_int}/Seed_{seed}/x_paths_gas.txt
190             ", "a") as f:
191         for xp in x_path_coord_list_gas:
192             f.write(f"{xp}\n")
193
194     with open(f"Network_images_final/Gridsize_{gridsize_int}/Seed_{seed}/y_paths_gas.txt
195             ", "a") as f:
196         for yp in y_path_coord_list_gas:
197             f.write(f"{yp}\n")
198
199     #creating file of locations of corner shapes to be added in COMSOL
200
201     cornergrid = np.zeros((w,h))
202
203     for j in range(1,h,2):
204         for i in range(1,w,2):
205
206             #if (i < len(coolgrid)-2) and (j < len(coolgrid)-2):
207
208                 x_north = i
209
210                 x_east = i+1
211
212                 x_south = i
213
214                 x_west = i-1
215
216
217                 #wrap around boundary condition
218
219                 if i == w - 1:
220
221                     x_east = 0
222
223                     if j == h - 1:
224
225                         y_north = 0
226
227
228                     # check if porethroats filled with water
229
230                     N_throat = (Matrix[y_north][x_north]["fluid"] == "water")

```

```

224         E_throat = (Matrix[y_east][x_east]["fluid"] == "water")
225         S_throat = (Matrix[y_south][x_south]["fluid"] == "water")
226         W_throat = (Matrix[y_west][x_west]["fluid"] == "water")
227
228         # check if porebodies not filled with water
229         NE_pb = (Matrix[y_north][x_east]["fluid"] == "water")
230         SE_pb = (Matrix[y_south][x_east]["fluid"] == "water")
231         SW_pb = (Matrix[y_south][x_west]["fluid"] == "water")
232         NW_pb = (Matrix[y_north][x_west]["fluid"] == "water")
233
234         if W_throat and S_throat and not SW_pb :
235             cornergrid[i-1][j-1] = int(1)
236
237         if E_throat and S_throat and not SE_pb:
238             cornergrid[i][j-1] = 2
239
240         if E_throat and N_throat and not NE_pb:
241             cornergrid[i][j] = 3
242
243         if W_throat and N_throat and not NW_pb:
244             cornergrid[i-1][j] = 4
245
246         #print(cornergrid)
247         cornergrid = cornergrid.astype(int)
248         np.savetxt(f"Network_images_final/Gridsize_{gridsize_int}/Seed_{seed}/corners.txt",
249                     cornergrid, delimiter=" ", fmt="%s")
250         break
251
252
253     p_lowest= np.inf # first p to have bridges horizontally and vertically
254
255     x_path_coord_list_water = []
256     y_path_coord_list_water = []
257
258     first_water = 0
259     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)
260         :

```

```

259     if (int(p_i*100) > 100):
260         break
261
262     p_bridge = [p, p_i]
263
264     #place bridges
265     for x in range(w):
266         for y in range(h):
267             if (p_bridge[0] <= round(Matrix[x][y]['throat_size'],2) <= p_bridge[1]) and (
268                 Matrix[x][y]['fluid'] == 'gas'):
269                 if Matrix[x][y]['type'] == 'throat':
270                     Matrix[x][y]['fluid'] = 'bridge'
271                     Matrix[x][y]['color'] = bridge
272
273         n_x_paths = 0
274
275         n_y_paths = 0
276
277         #water paths
278         for x in range(0,w):
279             if (Matrix[x][0]["fluid"] != "gas"):
280                 y_dir2 = bfs_opposite_side(Matrix, (x,0),"y-direction","water")
281                 if y_dir2 is not None:
282
283                     y_dir_list_water.append(y_dir2)
284                     n_y_paths += 1
285
286             if x not in y_path_coord_list_water:
287                 y_path_coord_list_water.append(x)
288
289             for (i,j) in y_dir2:
290                 if Matrix[i][j]["fluid"] != "bridge":
291                     if Matrix[i][j]["fluid"] == "water":
292                         Matrix[i][j]["color"] = water_path
293
294

```

```

295     for y in range(0,h):
296
297         if (Matrix[0][y]["fluid"] != "gas"):
298
299             x_dir2 = bfs_opposite_side(Matrix, (0,y),"x-direction","water")
300
301             if x_dir2 is not None:
302
303                 x_dir_list_water.append(x_dir2)
304
305                 n_x_paths += 1
306
307
308             if y not in x_path_coord_list_water:
309
310                 x_path_coord_list_water.append(y)
311
312
313             for (i,j) in x_dir2:
314
315                 if Matrix[i][j]["fluid"] != "bridge":
316
317                     if Matrix[i][j]["fluid"] == "water":
318
319                         Matrix[i][j]["color"] = water_path
320
321
322             if (n_x_paths > 0) and (n_y_paths > 0):
323
324                 first_water += 1
325
326
327             if first_water == 1 or p_i == 1:
328
329                 if p_i < p_lowest:
330
331                     p_lowest = p_i #finding the lowest p value where gas and water both bridge
332
333                     horizontally and vertically
334
335
336             Output = np.array(copy.deepcopy(Matrix))
337
338             for jj in range(h):
339
340                 for ii in range(w):
341
342                     Output[ii,jj] = Output[ii,jj]["fluid"]
343
344
345             horizontals = Output[1::2,:,:2]
346
347             verticals = Output[:,::2, 1::2]
348
349             porebodies = Output[:,::2,:2]
350
351
352             p_values = [p_first_gas, p_lowest]
353
354             p_gas_list.append(p_first_gas)
355
356             p_water_list.append(p_lowest)

```

```

331     pnumb = int(round(10*(p_i - p_lowest)))
332
333     if not os.path.exists(f"Network_images_final/Gridsize_{gridsize_int}/Seed_{seed}
334         }/{pnumb}"):
335
336         os.makedirs(f"Network_images_final/Gridsize_{gridsize_int}/Seed_{seed}/{p
337             pnumb}")
338
339
340         np.savetxt(f"Network_images_final/Gridsize_{gridsize_int}/Seed_{seed}/{p
341             numb}/horizontals.txt", horizontals, delimiter="\t", fmt="%s")
342
343         np.savetxt(f"Network_images_final/Gridsize_{gridsize_int}/Seed_{seed}/{p
344             numb}/verticals.txt", verticals, delimiter="\t", fmt="%s")
345
346         np.savetxt(f"Network_images_final/Gridsize_{gridsize_int}/Seed_{seed}/{p
347             numb}/porebodies.txt", porebodies, delimiter="\t", fmt="%s")
348
349         #plot_grid(fign, f"Seed: {seed}, p: {p_i}, x_paths: {n_x_paths}, y_paths: {
350             n_y_paths}", f"P{pnumb}/network_bridge_iteration.png")
351
352         #fign += 1
353
354
355         with open(f"Network_images_final/Gridsize_{gridsize_int}/Seed_{seed}/info.txt",
356             "a") as f:
357
358             f.write(f"{p_i}\n")
359
360
361         with open(f"Network_images_final/Gridsize_{gridsize_int}/Seed_{seed}/{p
362             numb}/x_paths_water.txt", "a") as f:
363
364             for xp in x_path_coord_list_water:
365
366                 f.write(f"{xp}\t")
367
368
369         with open(f"Network_images_final/Gridsize_{gridsize_int}/Seed_{seed}/{p
370             numb}/y_paths_water.txt", "a") as f:
371
372             for yp in y_path_coord_list_water:
373
374                 f.write(f"{yp}\t")
375
376
377             break
378
379         print(gridsize)
380
381         fig1 = plt.figure(fign, figsize=(6, 6))
382
383         plt.scatter(gridsize*np.ones(len(p_water_list)), p_water_list, s=None, alpha=0.1, c="grey")

```

```

359 plt.scatter(gridsize, np.median(p_water_list), c="blue")
360 plt.title("First_WP_Flow.Paths", size=font_size)
361 plt.xlabel("Lattice_Size", fontsize=font_size)
362 plt.ylabel("$p_{network}$", fontsize=font_size)
363 plt.xticks(xtick_range, fontsize=font_size-5)
364 plt.yticks(fontsize=font_size)
365 plt.ylim([0,1])
366
367 fig2 = plt.figure(fign+1, figsize=(6, 6))
368 plt.scatter(gridsize*np.ones(len(p_gas_list)), p_gas_list, s=None, alpha=0.1, c="grey")
369 plt.scatter(gridsize, np.median(p_gas_list), c="green")
370 plt.title("First_NWP_Flow.Paths", size=font_size)
371 plt.xlabel("Lattice_Size", fontsize=font_size)
372 plt.ylabel("$p_{network}$", fontsize=font_size)
373 plt.xticks(xtick_range, fontsize=font_size-5)
374 plt.yticks(fontsize=font_size)
375 plt.ylim([0,1])
376 #p_water_list
377
378 p_water_medians.append(np.median(p_water_list))
379 p_gas_medians.append(np.median(p_gas_list))
380
381
382 fig3 = plt.figure(fign+2, figsize=(6, 6))
383 plt.scatter(grid_range, p_water_medians, c="blue")
384 plt.title("Median_Values")
385 plt.xlabel("Lattice_Size", fontsize=font_size)
386 plt.ylabel("$p_{network}$", fontsize=font_size)
387 plt.xticks(xtick_range, fontsize=font_size-5)
388 plt.yticks(fontsize=font_size)
389 plt.ylim([0,1])
390
391 plt.scatter(grid_range, p_gas_medians, c="green")
392 plt.xlabel("Lattice_Size", fontsize=font_size)
393 plt.ylabel("$p_{network}$", fontsize=font_size)
394 plt.xticks(xtick_range, fontsize=font_size-5)
395 plt.yticks(fontsize=font_size)

```

```

396 plt.ylim([0,1])
397
398 print(p_water_medians)
399 print(p_gas_medians)
400
401 #save figures
402 fig1.savefig("firstWP.png")
403 fig2.savefig("firstNWP.png")
404 fig3.savefig("firstmedians.png")

```

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

Fig. 22

```

1 import numpy as np
2 import matplotlib
3 import matplotlib.pyplot as plt
4 import collections
5 import json
6 import copy
7 import os
8
9 gridsize = 24
10 gridsize_int = int(gridsize/2) #amount of pillars
11
12 gridsizes = [gridsize]
13 ssizes = [750] #size of the squares in
14
15 s = ssizes[gridsizes.index(gridsize)]
16
17 #colors
18 pillar = 'silver'
19 pillar_path = 'whitesmoke'
20 water = 'royalblue'
21 water_path = 'royalblue'
22 gas = 'black'
23 gas_path = 'black'

```

```

24 bridge      = 'yellow'
25 snap_off    = 'red'
26
27 fign = 0
28
29 for seed in range(0,10):
30
31     np.random.seed(seed)
32
33     #Initial Matrix
34
35     w, h = gridsize, gridsize #w = width
36
37     Matrix = [[{'throat_size':0,'type':'pillar', 'fluid':'pillar','color':pillar} for x in range(w)]
38             for y in range(h)]
39
40     #all bodies initially filled with water
41
42     for x in range(w)[::2]:
43
44         for y in range(h)[::2]:
45
46             Matrix[x][y] = {'throat_size':1,'type':'body','fluid':'water', 'color':water}
47
48     #setting throat size depending on p
49
50     for x in range(w+1)[::2]:
51
52         for y in range(h):
53
54             if (x - 1 -(y %2)) >= 0:
55
56                 Matrix[x - 1 -(y %2)][y] = {'throat_size':np.random.rand(),'type':'throat','fluid':
57                                         ':water','color':water}
58
59
60     for p in np.round(np.arange(0.1,1,0.01),2)[::-1]:
61
62         # setting gas throats
63
64         for x in range(w+1)[::2]:
65
66             for y in range(h):
67
68                 if (x - 1 -(y %2)) >= 0:
69
70                     if (Matrix[x - 1 -(y %2)][y]['type'] == 'throat') and (Matrix[x - 1 -(y %2)][y]
71
72                         ]['throat_size'] > p):
73
74                         Matrix[x - 1 -(y %2)][y]['fluid'] = 'gas'
75
76                         Matrix[x - 1 -(y %2)][y]['color'] = gas
77
78
79             #setting porebodies adjacent to gas throats to gas

```

```

58     for x in range(0,w):
59         for y in range(0,h):
60             if Matrix[x][y]['fluid'] == 'gas':
61
62                 for i in [-1,1]:
63                     if 0 <= x <= w - 1:
64                         ix = i
65
66                     if x == 0:
67                         ix = 1
68
69                     if x == w - 1:
70                         ix = -1
71
72                     if Matrix[x+ix][y]['type'] == 'body':
73
74                         Matrix[x+ix][y]['fluid'] = 'gas'
75                         Matrix[x+ix][y]['color'] = gas
76
77
78                     if x == 0: # make sure boundary has pore bodies adjacent vertically
79                         iy = i
80
81
82                     if y == h - 1:
83                         iy = -1
84
85                     if Matrix[x][y+iy]['type'] == 'body':
86
87                         Matrix[x][y+iy]['fluid'] = 'gas'
88                         Matrix[x][y+iy]['color'] = gas
89
90
91
92                     if 0 <= y <= h - 1:
93                         if y == 0:
94                             i = 1
95
96                         if y == h - 1:
97                             i = -1
98
99                     if Matrix[x][y+i]['type'] == 'body':
100
101                         Matrix[x][y+i]['fluid'] = 'gas'
102                         Matrix[x][y+i]['color'] = gas
103
104
105                     if y == 0: # make sure boundary has pore bodies adjacent horizontally
106                         ix = i

```

```

95
96         if x == w - 1:
97             ix = -1
98             if Matrix[x+ix][y]['type'] == 'body':
99                 Matrix[x+ix][y]['fluid'] = 'gas'
100                Matrix[x+ix][y]['color'] = gas
101
102            # make sure wrap around boundary condition is satisfied
103            if Matrix[w - 1][y]['fluid'] == 'gas':
104                Matrix[0][y]['type'] == 'body'
105                Matrix[0][y]['fluid'] = 'gas'
106                Matrix[0][y]['color'] = gas
107            if Matrix[x][h-1]['fluid'] == 'gas':
108                Matrix[x][0]['type'] == 'body'
109                Matrix[x][0]['fluid'] = 'gas'
110                Matrix[x][0]['color'] = gas
111
112            x_path_exists_check, y_path_exists_check = False, False
113            x_path_coord_list_gas, y_path_coord_list_gas = [], []
114            x_dir_list_gas, y_dir_list_gas = [], [] #The whole gas backbone
115
116            if not y_path_exists_check:
117                for x in range(0,w):
118                    if (Matrix[x][0]['fluid'] == 'gas'):
119                        y_dir = bfs_opposite_side(Matrix, (x,0),'y-direction','gas')
120                        if y_dir is not None:
121                            y_path_coord_list_gas.append(x)
122                            y_path_exists_check = True
123                            y_dir_list_gas.append(y_dir)
124
125
126
127            if not x_path_exists_check:
128
129                for y in range(0,h):
130                    if (Matrix[0][y]['fluid'] == 'gas'):

```

```

132         x_dir = bfs_opposite_side(Matrix, (0,y),'x-direction','gas')
133
134     if x_dir is not None:
135
136         x_path_coord_list_gas.append(y)
137
138         x_path_exists_check = True
139
140         x_dir_list_gas.append(x_dir)
141
142
143     if x_path_exists_check and y_path_exists_check:
144
145         #remove gas
146
147         checked = []
148
149         for x in range(w):
150
151             for y in range(h):
152
153                 bfs_remove_gas(Matrix, (x,y))
154
155
156                 #set pillars back
157
158                 for x in range(w)[1::2]:
159
160                     for y in range(h)[1::2]:
161
162                         Matrix[x][y] = {'throat_size':0,'type':'pillar', 'fluid':'pillar','color':pillar}
163
164
165                 for y_dir in y_dir_list_gas:
166
167                     for (i,j) in y_dir:
168
169                         Matrix[i][j]['color'] = gas_path
170
171                     for x_dir in x_dir_list_gas:
172
173                         for (i,j) in x_dir:
174
175                             Matrix[i][j]['color'] = gas_path
176
177
178
179             #create textfile for information and add p of the first gas flow paths
180
181             if not os.path.exists(f'Network images final/Gridsize {gridsize_int}/Seed {seed}'):
182
183                 os.makedirs(f'Network images final/Gridsize {gridsize_int}/Seed {seed}')
184
185
186             with open(f'Network images final/Gridsize {gridsize_int}/Seed {seed}/x_paths_gas.txt', 'a') as f:
187
188                 for xp in x_path_coord_list_gas:

```

```

167         f.write(f'{xp} ')
168
169     with open(f'Network images final/Gridsize {gridsize_int}/Seed {seed}/y_paths_gas.txt', 'a') as f:
170         for yp in y_path_coord_list_gas:
171             f.write(f'{yp} ')
172             p_first_gas = p
173
174
175         plot_grid(fign, f'Seed: {seed}, p: {p_first_gas}, First gas flow paths',
176                   network_first_gas.png')
177         fign += 1
178
179     #creating file of locations of corner shapes to be added in COMSOL
180     cornergrid = np.zeros((w,h))
181     for j in range(1,h,2):
182         for i in range(1,w,2):
183             #if (i < len(coolgrid)-2) and (j < len(coolgrid)-2):
184
185                 x_north = i
186                 x_east = i+1
187                 x_south = i
188                 x_west = i-1
189
190                 y_north = j+1
191                 y_east = j
192                 y_south = j-1
193                 y_west = j
194
195             #wrap around boundary condition
196             if i == w - 1:
197                 x_east = 0
198                 if j == h - 1:
199                     y_north = 0
200
201             # check if porethroats filled with water

```

```

202         N_throat = (Matrix[y_north][x_north]['fluid'] == 'water')
203         E_throat = (Matrix[y_east][x_east]['fluid'] == 'water')
204         S_throat = (Matrix[y_south][x_south]['fluid'] == 'water')
205         W_throat = (Matrix[y_west][x_west]['fluid'] == 'water')
206
207         # check if porebodies not filled with water
208         NE_pb = (Matrix[y_north][x_east]['fluid'] == 'water')
209         SE_pb = (Matrix[y_south][x_east]['fluid'] == 'water')
210         SW_pb = (Matrix[y_south][x_west]['fluid'] == 'water')
211         NW_pb = (Matrix[y_north][x_west]['fluid'] == 'water')
212
213         if W_throat and S_throat and not SW_pb :
214             cornergrid[i-1][j-1] = int(1)
215
216         if E_throat and S_throat and not SE_pb:
217             cornergrid[i][j-1] = 2
218
219         if E_throat and N_throat and not NE_pb:
220             cornergrid[i][j] = 3
221
222         if W_throat and N_throat and not NW_pb:
223             cornergrid[i-1][j] = 4
224
225         #save(cornergrid)
226         cornergrid = cornergrid.astype(int)
227         np.savetxt(f'Network images final/Gridsize {gridsize_int}/Seed {seed}/corners.txt',
228                     cornergrid, delimiter=' ', fmt='%s')
229         break
230
231
232         p_lowest= np.inf # first p to have bridges horizontally and vertically
233
234         x_path_coord_list_water = []
235         y_path_coord_list_water = []
236
237         fign += 1
238
239         first_water = 0

```

```

238 pnumb = -1
239 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):
240
241     if (int(p_i*100) > 100):
242         break
243     if pnumb > 2:
244
245         break
246
247     p_bridge = [p, p_i]
248
249     #place bridges
250     for x in range(w):
251         for y in range(h):
252             if (p_bridge[0] <= round(Matrix[x][y]['throat_size'],2) <= p_bridge[1]) and (Matrix[x][y]['fluid'] == 'gas'):
253                 if Matrix[x][y]['type'] == 'throat':
254                     Matrix[x][y]['fluid'] = 'bridge'
255                     Matrix[x][y]['color'] = bridge
256
257             n_x_paths = 0
258             n_y_paths = 0
259
260             x_dir_list_water, y_dir_list_water = [], [] #The whole water backbone
261
262             #water paths
263             for x in range(0,w):
264                 if (Matrix[x][0]['fluid'] != 'gas'):
265                     y_dir2 = bfs_opposite_side(Matrix, (x,0),'y-direction','water')
266
267                     if y_dir2 is not None:
268
269                         y_dir_list_water.append(y_dir2)
270                         n_y_paths += 1
271
272                         if x not in y_path_coord_list_water:
273                             y_path_coord_list_water.append(x)
274
275                         for (i,j) in y_dir2:

```

```

274         if Matrix[i][j]['fluid'] != 'bridge':
275             #if Matrix[i][j]['fluid'] == None:
276                 #Matrix[i][j]['color'] = pillar_path
277             if Matrix[i][j]['fluid'] == 'water':
278                 Matrix[i][j]['color'] = water_path
279
280
281     for y in range(0,h):
282         if (Matrix[0][y]['fluid'] != 'gas'):
283             x_dir2 = bfs_opposite_side(Matrix, (0,y),'x-direction','water')
284
285         if x_dir2 is not None:
286             x_dir_list_water.append(x_dir2)
287             n_x_paths += 1
288
289         if y not in x_path_coord_list_water:
290             x_path_coord_list_water.append(y)
291
292         for (i,j) in x_dir2:
293             if Matrix[i][j]['fluid'] != 'bridge':
294                 #if Matrix[i][j]['fluid'] == None:
295                     #Matrix[i][j]['color'] = pillar_path
296                 if Matrix[i][j]['fluid'] == 'water':
297                     Matrix[i][j]['color'] = water_path
298
299             if (int(round((p_i*100 - p_first_gas*100)) % 10 == 0) or (int(round(p_i*100)) == 100)):
300
301                 pnumb += 1
302
303             if (n_x_paths > 0) and (n_y_paths > 0):
304
305                 Output = np.array(copy.deepcopy(Matrix))
306
307                 for jj in range(h):
308                     for ii in range(w):
309                         Output[ii,jj] = Output[ii,jj]['fluid']
310
311
312                 horizontals = Output[1::2,:,:2]
313
314                 verticals = Output[:,1::2,2]
315
316                 porebodies = Output[:,2::2,2]

```

```

311
312     if not os.path.exists(f'Network images final/Gridsize {gridsize_int}/Seed {seed}/P{pnumb}'):
313         os.makedirs(f'Network images final/Gridsize {gridsize_int}/Seed {seed}/P{pnumb}')
314
315
316     np.savetxt(f'Network images final/Gridsize {gridsize_int}/Seed {seed}/P{pnumb}/
317             horizontals.txt', horizontals, delimiter=' ', fmt='%s')
318     np.savetxt(f'Network images final/Gridsize {gridsize_int}/Seed {seed}/P{pnumb}/
319             verticals.txt', verticals, delimiter=' ', fmt='%s')
320     np.savetxt(f'Network images final/Gridsize {gridsize_int}/Seed {seed}/P{pnumb}/
321             porebodies.txt', porebodies, delimiter=' ', fmt='%s')
322     plot_grid(fign, f'Seed: {seed}, p: {p_i}, x_paths: {n_x_paths}, y_paths: {n_y_paths}
323             ', f'P{pnumb}/network_bridge_iteration.png')
324     fign += 1
325
326     with open(f'Network images final/Gridsize {gridsize_int}/Seed {seed}/info.txt', 'a') as f:
327         f.write(f'{p_i}\n')
328
329     with open(f'Network images final/Gridsize {gridsize_int}/Seed {seed}/P{pnumb}/
330             x_paths_water.txt', 'a') as f:
331         for xp in x_path_coord_list_water:
332             f.write(f'{xp} ')
333
334     with open(f'Network images final/Gridsize {gridsize_int}/Seed {seed}/P{pnumb}/
335             y_paths_water.txt', 'a') as f:
336         for yp in y_path_coord_list_water:
337             f.write(f'{yp} ')
338
339     #break

```

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

```
1 package builder;
2
3 import com.comsol.api.*;
4 import com.comsol.model.*;
5 import com.comsol.model.physics.*;
6 import com.comsol.model.application.*;
7
8 public class partsbuilder extends ApplicationMethod {
9
10    public void execute() {
11
12        model.param().set("H", "2");
13        model.param().descr("H", "Height");
14        model.param().set("R", "1");
15        model.param().descr("R", "Radius");
16        model.param().set("D", "2");
17        model.param().descr("D", "Displacement");
18
19        model.geom().create("part0", "Part", 3);
20        model.geom("part0").label("Prep_Part");
21        model.geom("part0").lengthUnit("\u00b5m");
22        model.geom("part0").create("wp1", "WorkPlane");
23        model.geom("part0").feature("wp1").set("unite", true);
24        model.geom("part0").feature("wp1").geom().create("r1", "Rectangle");
25        model.geom("part0").feature("wp1").geom().feature("r1").set("size", new int[]{2, 1});
26        model.geom("part0").feature("wp1").geom().feature("r1").set("size", new double[]{2, 1.95});
27        model.geom("part0").feature("wp1").geom().feature("r1").set("pos", new double[]{0, 0.025});
28        model.geom("part0").create("copy1", "Copy");
29        model.geom("part0").feature("copy1").set("displz", 2);
30        model.geom("part0").feature("copy1").selection("input").set("wp1");
31        model.geom("part0").create("wp2", "WorkPlane");
32        model.geom("part0").feature("wp2").set("unite", true);
33        model.geom("part0").feature("wp2").set("quickz", "11/40");
34        model.geom("part0").feature("wp2").geom().create("r1", "Rectangle");
```

```

35 model.geom("part0").feature("wp2").geom().feature("r1").set("size", new int[]{2, 1});
36 model.geom("part0").feature("wp2").geom().feature("r1").set("type", "curve");
37 model.geom("part0").feature("wp2").geom().feature("r1").set("pos", new String[]{"0", "1/2"});
38 model.geom("part0").feature("wp2").geom().create("qb1", "QuadraticBezier");
39 model.geom("part0").feature("wp2").geom().feature("qb1").setIndex("p", "1/2", 1, 0);
40 model.geom("part0").feature("wp2").geom().feature("qb1").setIndex("p", 1, 0, 1);
41 model.geom("part0").feature("wp2").geom().feature("qb1").setIndex("p", "3/5", 1, 1);
42 model.geom("part0").feature("wp2").geom().feature("qb1").setIndex("p", 2, 0, 2);
43 model.geom("part0").feature("wp2").geom().feature("qb1").setIndex("p", "1/2", 1, 2);
44 model.geom("part0").feature("wp2").geom().feature("qb1").set("w", new int[]{1, 1, 1});
45 model.geom("part0").feature("wp2").geom().create("mir1", "Mirror");
46 model.geom("part0").feature("wp2").geom().feature("mir1").selection("input").set("qb1");
47 model.geom("part0").feature("wp2").geom().feature("mir1").set("pos", new int[]{0, 1});
48 model.geom("part0").feature("wp2").geom().feature("mir1").set("axis", new int[]{0, 0});
49 model.geom("part0").feature("wp2").geom().feature("mir1").set("axis", new int[]{0, 1});
50 model.geom("part0").feature("wp2").geom().feature("mir1").set("keep", true);
51 model.geom("part0").feature("wp2").geom().feature().create("dell1", "Delete");
52 model.geom("part0").feature("wp2").geom().feature("dell1").selection("input").init(1);
53 model.geom("part0").feature("wp2").geom().feature("dell1").selection("input").set("r1", 1, 3);
54 model.geom("part0").feature("wp2").geom().create("csol1", "ConvertToSolid");
55 model.geom("part0").feature("wp2").geom().feature("csol1").selection("input").set("dell1", "mir1"
    , "qb1");

56 model.geom("part0").create("copy2", "Copy");
57 model.geom("part0").feature("copy2").selection("input").set("wp2");
58 model.geom("part0").feature("copy2").set("displz", "58/40");
59 model.geom("part0").create("wp3", "WorkPlane");
60 model.geom("part0").feature("wp3").set("unite", true);
61 model.geom("part0").feature("wp3").set("quickz", 1);
62 model.geom("part0").feature("wp3").geom().create("r1", "Rectangle");
63 model.geom("part0").feature("wp3").geom().feature("r1").set("size", new int[]{2, 1});
64 model.geom("part0").feature("wp3").geom().feature("r1").set("pos", new String[]{"0", "1/2"});
65 model.geom("part0").feature("wp3").geom().create("qb1", "QuadraticBezier");
66 model.geom("part0").feature("wp3").geom().feature("qb1").setIndex("p", "1/2", 1, 0);
67 model.geom("part0").feature("wp3").geom().feature("qb1").setIndex("p", 1, 0, 1);
68 model.geom("part0").feature("wp3").geom().feature("qb1").setIndex("p", "9/8", 1, 1);
69 model.geom("part0").feature("wp3").geom().feature("qb1").setIndex("p", 2, 0, 2);
70 model.geom("part0").feature("wp3").geom().feature("qb1").setIndex("p", "1/2", 1, 2);

```

```

71 model.geom("part0").feature("wp3").geom().feature("qb1").set("w", new int[]{1, 1, 1});
72 model.geom("part0").feature("wp3").geom().create("mir1", "Mirror");
73 model.geom("part0").feature("wp3").geom().feature("mir1").set("pos", new int[]{0, 1});
74 model.geom("part0").feature("wp3").geom().feature("mir1").set("axis", new int[]{0, 0});
75 model.geom("part0").feature("wp3").geom().feature("mir1").set("axis", new int[]{0, 1});
76 model.geom("part0").feature("wp3").geom().feature("mir1").selection("input").set("qb1");
77 model.geom("part0").feature("wp3").geom().feature().create("dell1", "Delete");
78 model.geom("part0").feature("wp3").geom().feature("dell1").selection("input").init();
79 model.geom("part0").feature("wp3").geom().feature("mir1").set("keep", true);
80 model.geom("part0").feature("wp3").geom().feature("dell1").selection("input").init(1);
81 model.geom("part0").feature("wp3").geom().feature("dell1").selection("input").set("r1", 1, 3);
82 model.geom("part0").feature("wp3").geom().create("csol1", "ConvertToSolid");
83 model.geom("part0").feature("wp3").geom().feature("csol1").selection("input").set("dell1", "mir1"
    , "qb1");
84 model.geom("part0").create("loft1", "Loft");
85 model.geom("part0").feature("loft1").selection("profile").set("copy1", "copy2", "wp1", "wp2", "
    wp3");
86
87 model.geom().create("part1", "Part", 3);
88 model.geom("part1").lengthUnit("\u00b5m");
89 model.geom("part1").create("pil", "PartInstance");
90 model.geom("part1").feature("pil").set("selkeepnoncontr", false);
91 model.geom("part1").feature("pil").set("part", "part0");
92 model.geom("part1").create("wp1", "WorkPlane");
93 model.geom("part1").feature("wp1").set("unite", true);
94 model.geom("part1").feature("wp1").set("quickplane", "yz");
95 model.geom("part1").feature("wp1").geom().create("cro1", "CrossSection");
96 model.geom("part1").feature("wp1").geom().create("ls1", "LineSegment");
97 model.geom("part1").feature("wp1").geom().feature("ls1").set("specify1", "coord");
98 model.geom("part1").feature("wp1").geom().feature("ls1").set("coord1", new int[]{1, 0});
99 model.geom("part1").feature("wp1").geom().feature("ls1").set("specify2", "coord");
100 model.geom("part1").feature("wp1").geom().feature("ls1").set("coord2", new int[]{1, 2});
101 model.geom("part1").feature("wp1").geom().create("unil", "Union");
102 model.geom("part1").feature("wp1").geom().feature("unil").selection("input").set("cro1");
103 model.geom("part1").feature("wp1").geom().feature("unil").selection("input").set("cro1", "ls1");
104 model.geom("part1").feature("wp1").geom().feature().create("dell1", "Delete");
105 model.geom("part1").feature("wp1").geom().feature("dell1").selection("input").init(1);

```

```

106 model.geom("part1").feature("wp1").geom().feature("dell").selection("input").set("unil", 5, 6,
107   7);
108
109 model.geom("part1").feature().create("rev1", "Revolve");
110
111 model.geom("part1").feature("rev1").set("workplane", "wp1");
112
113 model.geom("part1").feature("rev1").selection("input").set("wp1");
114
115 model.geom("part1").feature("rev1").set("angtype", "specang");
116
117 model.geom("part1").feature("rev1").set("pos", new int[]{1, 0});
118
119 model.geom("part1").feature().create("dell", "Delete");
120
121 model.geom("part1").feature("dell").selection("input").init();
122
123 model.geom("part1").feature("dell").selection("input").set("pil");
124
125 model.geom("part1").create("mov1", "Move");
126
127 model.geom("part1").feature("mov1").set("disply", -1);
128
129 model.geom("part1").feature("mov1").selection("input").set("rev1");
130
131
132
133
134
135
136
137
138
139
140
141

```

```

142 model.geom("part2").feature("dif1").selection("input2").set("copy1", "copy2", "cyl1", "mov1");
143 model.geom("part2").create("mov2", "Move");
144 model.geom("part2").feature("mov2").selection("input").set("dif1");
145 model.geom("part2").feature("mov2").set("disply", -1);
146 model.geom("part2").create("blk1", "Block");
147 model.geom("part2").feature("blk1").set("size", new int[]{2, 2, 1});
148 model.geom("part2").feature("blk1").set("pos", new int[]{0, -1, 1});
149 model.geom("part2").create("dif2", "Difference");
150 model.geom("part2").feature("dif2").selection("input").set("mov2");
151 model.geom("part2").feature("dif2").selection("input2").set("blk1");
152
153 model.geom().create("part3", "Part", 3);
154 model.geom("part3").lengthUnit("\u000b5m");
155 model.geom("part3").create("pil", "PartInstance");
156 model.geom("part3").feature("pil").set("selkeepnoncontr", false);
157 model.geom("part3").feature("pil").set("part", "part2");
158 model.geom("part3").create("wp1", "WorkPlane");
159 model.geom("part3").feature("wp1").set("unite", true);
160 model.geom("part3").feature("wp1").geom().create("e1", "Ellipse");
161 model.geom("part3").feature("wp1").geom().run("e1");
162 model.geom("part3").create("wp2", "WorkPlane");
163 model.geom("part3").feature("wp2").set("unite", true);
164 model.geom("part3").feature("wp2").geom().create("e1", "Ellipse");
165 model.geom("part3").feature("wp2").geom().feature("e1").set("semiaxes", new String[]{"6/10", "1"});
166
167 model.geom("part3").feature("wp2").geom().feature("e1").set("semiaxes", new String[]{"6/10", "35/100"});
168 model.geom("part3").feature("wp2").set("quickz", 1);
169 model.geom("part3").create("copy1", "Copy");
170 model.geom("part3").feature("copy1").selection("input").set("wp1");
171 model.geom("part3").feature("copy1").set("displz", 2);
172 model.geom("part3").create("loft1", "Loft");
173 model.geom("part3").feature("loft1").selection("profile").set("wp1");
174 model.geom("part3").feature("loft1").selection("profile").set("wp1", "wp2");
175 model.geom("part3").create("rtl", "RigidTransform");
176 model.geom("part3").feature("rtl").selection("input").set("loft1");

```

```

177 model.geom("part3").feature("rt1").set("axistype", "x");
178 model.geom("part3").feature("rt1").set("rot", 90);
179 model.geom("part3").feature("rt1").set("displ", new int[]{1, 1, 1});
180 model.geom("part3").create("dif1", "Difference");
181 model.geom("part3").feature("dif1").selection("input").set("pil");
182 model.geom("part3").feature("dif1").selection("input2").set("rt1");
183
184 model.geom().create("part4", "Part", 3);
185 model.geom("part4").label("Part_4");
186 model.geom("part4").lengthUnit("\u00b5m");
187 model.geom("part4").create("pil", "PartInstance");
188 model.geom("part4").feature("pil").set("selkeepnoncontr", false);
189 model.geom("part4").feature("pil").set("part", "part1");
190 model.geom("part4").create("cyl1", "Cylinder");
191 model.geom("part4").feature("cyl1").set("r", "1/2");
192 model.geom("part4").feature("cyl1").set("h", 2);
193 model.geom("part4").create("blk1", "Block");
194 model.geom("part4").feature("blk1").set("size", new String[]{"2", "-1", "1"});
195 model.geom("part4").feature("blk1").set("size", new int[]{2, 2, 1});
196 model.geom("part4").feature("blk1").set("pos", new int[]{-1, -1, 1});
197 model.geom("part4").create("dif1", "Difference");
198 model.geom("part4").feature("dif1").selection("input").set("pil");
199 model.geom("part4").feature("dif1").selection("input2").set("blk1", "cyl1");
200
201 model.geom().create("part5", "Part", 3);
202 model.geom("part5").lengthUnit("\u00b5m");
203 model.geom("part5").create("pil", "PartInstance");
204 model.geom("part5").feature("pil").set("selkeepnoncontr", false);
205 model.geom("part5").feature("pil").set("part", "part1");
206 model.geom("part5").create("cyl1", "Cylinder");
207 model.geom("part5").feature("cyl1").set("r", "1/2");
208 model.geom("part5").feature("cyl1").set("h", 2);
209 model.geom("part5").create("arr1", "Array");
210 model.geom("part5").feature("arr1").selection("input").set("pil");
211 model.geom("part5").feature("arr1").selection("input").set("cyl1", "pil");
212 model.geom("part5").feature("arr1").set("fullsize", new int[]{2, 2, 1});
213 model.geom("part5").feature("arr1").set("displ", new int[]{2, 2, 0});

```

```

214 model.geom("part5").create("pi2", "PartInstance");
215 model.geom("part5").feature("pi2").set("selkeepnoncontr", false);
216 model.geom("part5").feature("pi2").set("part", "part2");
217 model.geom("part5").create("pi3", "PartInstance");
218 model.geom("part5").feature("pi3").set("selkeepnoncontr", false);
219 model.geom("part5").feature("pi3").set("part", "part2");
220 model.geom("part5").feature("pi3").set("displ", new int[]{0, 2, 0});
221 model.geom("part5").create("pi4", "PartInstance");
222 model.geom("part5").feature("pi4").set("selkeepnoncontr", false);
223 model.geom("part5").feature("pi4").set("part", "part2");
224 model.geom("part5").feature("pi4").set("displ", new int[]{1, 0, 0});
225 model.geom("part5").feature("pi4").set("rot", 90);
226 model.geom("part5").feature("pi4").set("displ", new int[]{0, 0, 0});
227 model.geom("part5").create("pi5", "PartInstance");
228 model.geom("part5").feature("pi5").set("selkeepnoncontr", false);
229 model.geom("part5").feature("pi5").set("part", "part2");
230 model.geom("part5").feature("pi5").set("rot", 90);
231 model.geom("part5").feature("pi5").set("displ", new int[]{2, 0, 0});
232 model.geom("part5").create("unil", "Union");
233 model.geom("part5").feature("unil").selection("input").set("arr1", "pi2", "pi3", "pi4", "pi5");
234 model.geom("part5").create("blk1", "Block");
235 model.geom("part5").feature("blk1").set("size", new int[]{2, 2, 1});
236 model.geom("part5").create("dif1", "Difference");
237 model.geom("part5").feature("dif1").selection("input").set("blk1");
238 model.geom("part5").feature("dif1").selection("input2").set("unil");
239
240 model.geom().create("part6", "Part", 3);
241 model.geom("part6").lengthUnit("\u00b5m");
242 model.geom("part6").create("pi1", "PartInstance");
243 model.geom("part6").feature("pi1").set("selkeepnoncontr", false);
244 model.geom("part6").feature("pi1").set("part", "part1");
245 model.geom("part6").create("pi2", "PartInstance");
246 model.geom("part6").feature("pi2").set("selkeepnoncontr", false);
247 model.geom("part6").feature("pi2").set("part", "part2");
248 model.geom("part6").create("pi3", "PartInstance");
249 model.geom("part6").feature("pi3").set("selkeepnoncontr", false);
250 model.geom("part6").feature("pi3").set("part", "part2");

```

```

251 model.geom("part6").feature("pi3").set("rot", 90);
252 model.geom("part6").create("mir1", "Mirror");
253 model.geom("part6").feature("mir1").selection("input").set("pi2", "pi3");
254 model.geom("part6").feature("mir1").set("axis", new int[]{0, 0, 1});
255 model.geom("part6").feature("mir1").set("pos", new int[]{0, 0, 1});
256 model.geom("part6").feature("mir1").set("keep", true);
257 model.geom("part6").create("cyl1", "Cylinder");
258 model.geom("part6").feature("cyl1").set("r", "1/2");
259 model.geom("part6").feature("cyl1").set("h", 2);
260 model.geom("part6").feature("cyl1").set("pos", new int[]{0, 0, 0});
261 model.geom("part6").create("wp1", "WorkPlane");
262 model.geom("part6").feature("wp1").set("unite", true);
263 model.geom("part6").feature("wp1").set("quickz", 1);
264 model.geom("part6").feature("wp1").geom().create("qb1", "QuadraticBezier");
265 model.geom("part6").feature("wp1").geom().feature("qb1").setIndex("p", 0.15, 0, 0);
266 model.geom("part6").feature("wp1").geom().feature("qb1").setIndex("p", 1.95, 1, 0);
267 model.geom("part6").feature("wp1").geom().feature("qb1").setIndex("p", 0.5, 0, 1);
268 model.geom("part6").feature("wp1").geom().feature("qb1").setIndex("p", 1.5, 1, 1);
269 model.geom("part6").feature("wp1").geom().feature("qb1").setIndex("p", 0.95, 0, 2);
270 model.geom("part6").feature("wp1").geom().feature("qb1").setIndex("p", 1.15, 1, 2);
271 model.geom("part6").feature("wp1").geom().feature("qb1").set("w", new String[]{"1", "1/10", "1"});
272
273 model.geom("part6").feature("wp1").geom().feature("qb1").setIndex("p", 0.95, 1, 0);
274 model.geom("part6").feature("wp1").geom().feature("qb1").setIndex("p", 0.5, 1, 1);
275 model.geom("part6").feature("wp1").geom().feature("qb1").setIndex("p", 0.15, 1, 2);
276 model.geom("part6").feature("wp1").geom().create("pol1", "Polygon");
277 model.geom("part6").feature("wp1").geom().feature("pol1").set("source", "table");
278 model.geom("part6").feature("wp1").geom().feature("pol1").set("type", "open");
279 model.geom("part6").feature("wp1").geom().feature("pol1").setIndex("table", 0.15, 0, 0);
280 model.geom("part6").feature("wp1").geom().feature("pol1").setIndex("table", 1.95, 0, 1);
281 model.geom("part6").feature("wp1").geom().feature("pol1").setIndex("table", 0, 1, 0);
282 model.geom("part6").feature("wp1").geom().feature("pol1").setIndex("table", 1.95, 1, 1);
283 model.geom("part6").feature("wp1").geom().feature("pol1").setIndex("table", 0, 2, 0);
284 model.geom("part6").feature("wp1").geom().feature("pol1").setIndex("table", 1, 2, 1);
285 model.geom("part6").feature("wp1").geom().feature("pol1").setIndex("table", 0.95, 3, 0);
286 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);

```

```

287 model.geom("part6").feature("wp1").geom().feature("pol1").setIndex("table", 1.15, 4, 1);
288 model.geom("part6").feature("wp1").geom().feature("pol1").setIndex("table", 0.15, 0, 0);
289 model.geom("part6").feature("wp1").geom().feature("pol1").setIndex("table", 1.95, 0, 1);
290 model.geom("part6").feature("wp1").geom().feature("pol1").setIndex("table", 0, 1, 0);
291 model.geom("part6").feature("wp1").geom().feature("pol1").setIndex("table", 1.95, 1, 1);
292 model.geom("part6").feature("wp1").geom().feature("pol1").setIndex("table", 0, 2, 0);
293 model.geom("part6").feature("wp1").geom().feature("pol1").setIndex("table", 1, 2, 1);
294 model.geom("part6").feature("wp1").geom().feature("pol1").setIndex("table", 0.95, 3, 0);
295 model.geom("part6").feature("wp1").geom().feature("pol1").setIndex("table", 1, 3, 1);
296 model.geom("part6").feature("wp1").geom().feature("pol1").setIndex("table", 0.95, 4, 0);
297 model.geom("part6").feature("wp1").geom().feature("pol1").setIndex("table", 1.15, 4, 1);
298 model.geom("part6").feature("wp1").geom().feature("pol1").setIndex("table", 0.95, 0, 1);
299 model.geom("part6").feature("wp1").geom().feature("pol1").setIndex("table", 0.95, 1, 1);
300 model.geom("part6").feature("wp1").geom().feature("pol1").setIndex("table", 0, 2, 1);
301 model.geom("part6").feature("wp1").geom().feature("pol1").setIndex("table", 0, 3, 1);
302 model.geom("part6").feature("wp1").geom().feature("pol1").setIndex("table", 0.15, 4, 1);
303 model.geom("part6").feature("wp1").geom().create("csol1", "ConvertToSolid");
304 model.geom("part6").feature("wp1").geom().feature("csol1").selection("input").set("qb1");
305 model.geom("part6").feature("wp1").geom().feature("csol1").selection("input").set("pol1", "qb1")
;
306 model.geom("part6").create("wp2", "WorkPlane");
307 model.geom("part6").feature("wp2").set("unite", true);
308 model.geom("part6").feature("wp2").set("quickz", 1/2);
309 model.geom("part6").feature("wp2").geom().create("qb1", "QuadraticBezier");
310 model.geom("part6").feature("wp2").geom().feature("qb1").setIndex("p", 0.25, 0, 0);
311 model.geom("part6").feature("wp2").geom().feature("qb1").setIndex("p", 0.95, 1, 0);
312 model.geom("part6").feature("wp2").geom().feature("qb1").setIndex("p", 0.5, 0, 1);
313 model.geom("part6").feature("wp2").geom().feature("qb1").setIndex("p", 1.5, 1, 1);
314 model.geom("part6").feature("wp2").geom().feature("qb1").setIndex("p", 0.95, 0, 2);
315 model.geom("part6").feature("wp2").geom().feature("qb1").setIndex("p", 0.25, 1, 2);
316 model.geom("part6").feature("wp2").geom().feature("qb1").setIndex("p", 0.5, 1, 1);
317 model.geom("part6").feature("wp2").geom().feature("qb1").set("w", new double[]{1, 0.1, 1});
318 model.geom("part6").feature("wp2").geom().create("pol1", "Polygon");
319 model.geom("part6").feature("wp2").geom().feature("pol1").set("source", "table");
320 model.geom("part6").feature("wp2").geom().feature("pol1").set("type", "open");
321 model.geom("part6").feature("wp2").geom().feature("pol1").setIndex("table", 0.25, 0, 0);
322 model.geom("part6").feature("wp2").geom().feature("pol1").setIndex("table", 0.95, 0, 1);

```

```

323 model.geom("part6").feature("wp2").geom().feature("pol1").setIndex("table", 0, 1, 0);
324 model.geom("part6").feature("wp2").geom().feature("pol1").setIndex("table", 0.95, 1, 1);
325 model.geom("part6").feature("wp2").geom().feature("pol1").setIndex("table", 0, 2, 0);
326 model.geom("part6").feature("wp2").geom().feature("pol1").setIndex("table", 1, 2, 1);
327 model.geom("part6").feature("wp2").geom().feature("pol1").setIndex("table", 0.95, 3, 0);
328 model.geom("part6").feature("wp2").geom().feature("pol1").setIndex("table", 0, 2, 1);
329 model.geom("part6").feature("wp2").geom().feature("pol1").setIndex("table", 0, 3, 1);
330 model.geom("part6").feature("wp2").geom().feature("pol1").setIndex("table", 0.95, 4, 0);
331 model.geom("part6").feature("wp2").geom().feature("pol1").setIndex("table", 0.25, 4, 1);
332 model.geom("part6").feature("wp2").geom().create("csol1", "ConvertToSolid");
333 model.geom("part6").feature("wp2").geom().feature("csol1").selection("input").set("pol1");
334 model.geom("part6").feature("wp2").geom().feature("csol1").selection("input").set("pol1", "qb1")
335 ;
336 model.geom("part6").create("copy1", "Copy");
337 model.geom("part6").feature("copy1").set("displz", 1);
338 model.geom("part6").feature("copy1").selection("input").set("wp2");
339 model.geom("part6").create("wp3", "WorkPlane");
340 model.geom("part6").feature("wp3").set("unite", true);
341 model.geom("part6").feature("wp3").geom().create("pol1", "Polygon");
342 model.geom("part6").feature("wp3").geom().feature("pol1").set("source", "table");
343 model.geom("part6").feature("wp3").geom().feature("pol1").setIndex("table", 0.75, 0, 0);
344 model.geom("part6").feature("wp3").geom().feature("pol1").setIndex("table", 0.9, 0, 1);
345 model.geom("part6").feature("wp3").geom().feature("pol1").setIndex("table", 0, 1, 0);
346 model.geom("part6").feature("wp3").geom().feature("pol1").setIndex("table", 0.9, 1, 1);
347 model.geom("part6").feature("wp3").geom().feature("pol1").setIndex("table", 0, 2, 0);
348 model.geom("part6").feature("wp3").geom().feature("pol1").setIndex("table", 0, 2, 1);
349 model.geom("part6").feature("wp3").geom().feature("pol1").setIndex("table", 0.9, 3, 0);
350 model.geom("part6").feature("wp3").geom().feature("pol1").setIndex("table", 0, 3, 1);
351 model.geom("part6").feature("wp3").geom().feature("pol1").setIndex("table", 0.9, 4, 0);
352 model.geom("part6").feature("wp3").geom().feature("pol1").setIndex("table", 0.75, 4, 1);
353 model.geom("part6").feature("wp3").geom().feature("pol1").setIndex("table", 0.75, 5, 0);
354 model.geom("part6").feature("wp3").geom().feature("pol1").setIndex("table", 0.9, 5, 1);
355 model.geom("part6").create("copy2", "Copy");
356 model.geom("part6").feature("copy2").selection("input").set("wp3");
357 model.geom("part6").feature("copy2").set("displz", 2);
358 model.geom("part6").create("loft1", "Loft");
359 model.geom("part6").feature("loft1").selection("profile").set("copy1", "copy2", "wp1", "wp2", " "

```

```

        wp3");

359 model.geom("part6").create("dif1", "Difference");

360 model.geom("part6").feature("dif1").selection("input").set("loft1");

361 model.geom("part6").feature("dif1").selection("input2").set("cyl1", "mir1", "pil1", "pi2", "pi3")

;

362 model.geom("part6").create("mov1", "Move");

363 model.geom("part6").feature("mov1").set("disply", 1);

364 model.geom("part6").feature("mov1").selection("input").set("dif1");

365 model.geom("part6").create("rt1", "RigidTransform");

366 model.geom("part6").feature("rt1").selection("input").set("mov1");

367 model.geom("part6").feature("rt1").set("displ", new int[]{1, 2, 0});

368 model.geom("part6").feature("rt1").set("rot", 180);

369 model.geom("part6").create("blk1", "Block");

370 model.geom("part6").feature("blk1").set("pos", new int[]{0, 0, 1});

371 model.geom("part6").create("dif2", "Difference");

372 model.geom("part6").feature("dif2").selection("input").set("rt1");

373 model.geom("part6").feature("dif2").selection("input2").set("blk1");

374 }

375 }

```

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

```

1 package builder;

2

3 import com.comsol.api.*;
4 import com.comsol.model.*;
5 import com.comsol.model.physics.*;
6 import com.comsol.model.application.*;
7 import java.util.Random;
8 import javax.swing.JOptionPane;
9 import java.util.ArrayList;
10 import java.io.*;
11 import java.util.List;
12 import java.nio.file.*;
13 import javax.swing.*;

```

```

14
15
16
17 public class Build extends ApplicationMethod {
18
19     public String[][] loadFileToArray(int gridsize, String filename) {
20
21         // JOptionPane.showMessageDialog(null, ""+System.getProperty("java.io.tmpdir"));
22
23         try {
24
25             File file = new File("C:\\\\Users\\\\eobbens\\\\Master_Thesis_Grids\\\\Gridsize_" + gridsize + "\\\\"
26
27                         + filename);
28
29             BufferedReader br = new BufferedReader(new FileReader(file));
30
31             String st;
32             int lineCount = 0;
33
34             while ((st = br.readLine()) != null) {
35
36                 lineCount++;
37
38             }
39
40             br.close();
41
42             String[][] fileArray = new String[lineCount][];
43
44             lineCount = 0;
45
46             while ((st = br.readLine()) != null) {
47
48                 fileArray[lineCount++] = st.split("\\_");
49
50             }
51
52             br.close();
53
54             return fileArray;
55
56         } catch (Exception e) {
57
58             e.printStackTrace();
59
60             // JOptionPane.showMessageDialog(null);
61
62         }
63
64         return null;
65
66     }

```

```

50
51
52 public void execute() {
53
54     //Deletes all the settings of the previous iteration
55     Boolean deletenodes = new Boolean("true");
56
57     for (int gridsize = 16; gridsize < 17; gridsize++) {
58         for (int seedn = 7; seedn < 10; seedn++) {
59             for (int pn = 0; pn < 5; pn++) {
60                 for (int directionn = 0; directionn < 1; directionn++) { //0 = x-direction, 1 = y-
61                     direction
62
63                     for (int fluidn = 0; fluidn < 1; fluidn++) { // 0 = water, 1 = gas
64
65                     String[][] horizontals = loadFileToArray(gridsize, "Seed\u208e"+seedn+"\\"+pn+"\\"
66                                         +horizontals+".txt");
67
68                     String[][] porebodies = loadFileToArray(gridsize, "Seed\u208e"+seedn+"\\"+pn+"\\"
69                                         +porebodies+".txt");
70
71                     String[][] verticals = loadFileToArray(gridsize, "Seed\u208e"+seedn+"\\"+pn+"\\\"+verticals
72                                         .txt");
73
74                     String[][] corners = loadFileToArray(gridsize, "Seed\u208e"+seedn+"\\"+corners.txt");
75
76                     if (horizontals == null || porebodies == null || verticals == null || corners ==
77                         null) {
78
79                         continue;
80
81                     }
82
83
84                     int count = 1;
85
86
87                     model.component().create("comp1", true);
88
89                     model.component("comp1").geom().create("geom1", 3);
90
91                     model.component("comp1").geom("geom1").lengthUnit("\u00b5m"); //set lengthscale to
92
93                         microns
94
95                     model.component("comp1").mesh().create("mesh1");
96
97                     model.component("comp1").physics().create("spf", "CreepingFlow", "geom1"); //add

```

```

    creeping flow physics

81

82

83     //whole network selection list

84     ArrayList<String> selectionlist = new ArrayList<String>();

85

86     // create first pillar

87     model.component("comp1").geom("geom1").create("pi"+count, "PartInstance");

88     model.component("comp1").geom("geom1").feature("pi"+count).set("part", "part4");

89     count++;

90     // create array of pillars

91     model.component("comp1").geom("geom1").create("arr1", "Array");

92     model.component("comp1").geom("geom1").feature("arr1").selection("input").set("pil")

93     ;

94     model.component("comp1").geom("geom1").feature("arr1").set("fullsize", new int[]{

95         gridsize, gridsize, 1});

96

97     model.component("comp1").geom("geom1").feature("arr1").set("displ", new String[]{"D"

98         , "D", "0"});

99     selectionlist.add("arr1");

100

101     // Arrange the geometry parts according to the interface text files created with the

102         network arrangement code.

103     // "verticals.txt": WP pore throat or bridge or NWP pore throat with a top-bottom

104         orientation.

105     // "horizontals.txt": WP pore throat or bridge or NWP pore throat with a left-right

106         orientation.

107     // "porebodies.txt": WP or NWP pore body.

108     // "corners.txt": locations and orientation to add a WP part geometry where two WP

109         filled pore throats meet at a 90 degree angle.

110

111     try {

112         for (int i = 0; i < gridsize; i++) {

113             for (int j = 0; j < gridsize; j++) {

114                 //String name = null;

115                 System.out.println("i:"+i+", j:"+j);

116                 if (verticals[i][j].equals("water")) {

117                     model.component("comp1").geom("geom1").create("pi"+count, "PartInstance");

```

```

110     model.component("comp1").geom("geom1").feature("pi"+count).set("part", "part2");
111
112     model.component("comp1").geom("geom1").feature("pi"+count).set("displ", new
113         String[]{i+"*D-2", "D*"+j, "0"});
114
115     model.component("comp1").geom("geom1").feature("pi"+count).set("rot", 0);
116     selectionlist.add("pi"+count);
117
118
119
120     if (i == 0) {
121         model.component("comp1").geom("geom1").create("copy"+count, "Copy");
122         model.component("comp1").geom("geom1").feature("copy"+count).selection("input")
123             .set("pi"+count);
124         model.component("comp1").geom("geom1").feature("copy"+count).set("displx",
125             gridsize+"*D");
126         selectionlist.add("copy"+count);
127     }
128 }
129
130
131     if (verticals[i][j].equals("bridge")) {
132         model.component("comp1").geom("geom1").create("pi"+count, "PartInstance");
133         model.component("comp1").geom("geom1").feature("pi"+count).set("part", "part3");
134         model.component("comp1").geom("geom1").feature("pi"+count).set("displ", new
135             String[]{i+"*D-2", "D*"+j, "0"});
136         model.component("comp1").geom("geom1").feature("pi"+count).set("rot", 0);
137         selectionlist.add("pi"+count);
138     }
139
140     count++;

```

```

139
140 }
141 } catch (Exception e) {
142 JOptionPane.showMessageDialog(null, ""+e);
143 }
144 // adding the
145 for (int i = 0; i < gridsize; i++) {
146 for (int j = 0; j < gridsize; j++) {
147
148 if (horizontals[i][j].equals("water")) {
149 model.component("comp1").geom("geom1").create("pi"+count, "PartInstance");
150 model.component("comp1").geom("geom1").feature("pi"+count).set("part", "part2"
151 );
152 model.component("comp1").geom("geom1").feature("pi"+count).set("displ", new
153 String[] {"D*"+i, j+"*D-2", "0"});
154 model.component("comp1").geom("geom1").feature("pi"+count).set("rot", 90);
155 selectionlist.add("pi"+count);
156
157 if (j == 0) {
158 model.component("comp1").geom("geom1").create("copy"+count, "Copy");
159 model.component("comp1").geom("geom1").feature("copy"+count).selection("
160 input").set("pi"+count);
161 model.component("comp1").geom("geom1").feature("copy"+count).set("display",
162 gridsize+"*D");
163 selectionlist.add("copy"+count);
164 }
165 }
166 if (horizontals[i][j].equals("bridge")) {
167 model.component("comp1").geom("geom1").create("pi"+count, "PartInstance");
168 model.component("comp1").geom("geom1").feature("pi"+count).set("part", "part3"
169 );
170 model.component("comp1").geom("geom1").feature("pi"+count).set("displ", new
171 String[] {"D*"+i, j+"*D-2", "0"});
172 model.component("comp1").geom("geom1").feature("pi"+count).set("rot", 90);
173 selectionlist.add("pi"+count);
174
175 if (j == 0) {

```

```

170     model.component("comp1").geom("geom1").create("copy"+count, "Copy");
171
172     model.component("comp1").geom("geom1").feature("copy"+count).selection("
173         input").set("pi"+count);
174
175     model.component("comp1").geom("geom1").feature("copy"+count).set("display",
176         gridsize+"*D");
177
178     selectionlist.add("copy"+count);
179
180 }
181
182 count++;
183
184 }
185
186 // adding the water filled pore bodies
187
188 for (int i = 0; i < gridsize; i++) {
189
190     for (int j = 0; j < gridsize; j++) {
191
192         if (porebodies[i][j].equals("water")) {
193
194             model.component("comp1").geom("geom1").create("pi"+count, "PartInstance");
195
196             model.component("comp1").geom("geom1").feature("pi"+count).set("part", "part5"
197
198             );
199
200             model.component("comp1").geom("geom1").feature("pi"+count).set("displ", new
201
202                 String[] {"D*"+(i-1), "D*"+(j-1), "0"});
203
204             selectionlist.add("pi"+count);
205
206
207             if (i == 0 && j != 0) {
208
209                 model.component("comp1").geom("geom1").create("copy"+count, "Copy");
210
211                 model.component("comp1").geom("geom1").feature("copy"+count).selection("
212                     input").set("pi"+count);
213
214                 model.component("comp1").geom("geom1").feature("copy"+count).set("displx",
215
216                     gridsize+"*D");
217
218                 selectionlist.add("copy"+count);
219
220             }
221
222
223             if (i != 0 && j == 0) {
224
225                 model.component("comp1").geom("geom1").create("copy"+count, "Copy");
226
227                 model.component("comp1").geom("geom1").feature("copy"+count).selection("
228                     input").set("pi"+count);

```

```

200     model.component("comp1").geom("geom1").feature("copy"+count).set("disply",
201         gridsize+"*D");
202
203     selectionlist.add("copy"+count);
204 }
205
206
207     if (i == 0 && j == 0) {
208
209         model.component("comp1").geom("geom1").create("copy"+count, "Copy");
210
211         model.component("comp1").geom("geom1").feature("copy"+count).selection("
212             input").set("pi"+count);
213
214         model.component("comp1").geom("geom1").feature("copy"+count).set("displx",
215             gridsize+"*D");
216
217         selectionlist.add("copy"+count);
218
219     }
220
221     count++;
222
223 }
224
225
226     for (int i = 0; i < 2*gridsize; i++) {
227
228         for (int j = 0; j < 2*gridsize; j++) {
229
230             if (corners[i][j].equals("1")) {
231
232                 model.component("comp1").geom("geom1").create("pi"+count, "PartInstance");
233
234                 model.component("comp1").geom("geom1").feature("pi"+count).set("part", "part6"
235                     );
236
237                 model.component("comp1").geom("geom1").feature("pi"+count).set("displ", new

```

```

    String[] {"D*(j/2-0.5), "D*(i/2-0.5), "0"});

231    selectionlist.add("pi"+count);

232

233    count++;

234 }

235 if (corners[i][j].equals("2")) {

236     model.component("comp1").geom("geom1").create("pi"+count, "PartInstance");

237     model.component("comp1").geom("geom1").feature("pi"+count).set("part", "part6"

238     );

239     model.component("comp1").geom("geom1").feature("pi"+count).set("displ", new

240         String[] {"D*(j/2-0.5), "D*(i/2+0.5), "0"});

241         model.component("comp1").geom("geom1").feature("pi"+count).set("rot", -90);

242         selectionlist.add("pi"+count);

243

244         count++;

245     }

246     if (corners[i][j].equals("3")) {

247         model.component("comp1").geom("geom1").create("pi"+count, "PartInstance");

248         model.component("comp1").geom("geom1").feature("pi"+count).set("part", "part6"

249         );

250         model.component("comp1").geom("geom1").feature("pi"+count).set("displ", new

251             String[] {"D*(j/2+0.5), "D*(i/2+0.5), "0"});

252             model.component("comp1").geom("geom1").feature("pi"+count).set("rot", -180);

253             selectionlist.add("pi"+count);

254

255             count++;

256     }

257     if (corners[i][j].equals("4")) {

258         model.component("comp1").geom("geom1").create("pi"+count, "PartInstance");

259         model.component("comp1").geom("geom1").feature("pi"+count).set("part", "part6"

260         );

261         model.component("comp1").geom("geom1").feature("pi"+count).set("displ", new

262             String[] {"D*(j/2+0.5), "D*(i/2-0.5), "0"});

263             model.component("comp1").geom("geom1").feature("pi"+count).set("rot", -270);

264             selectionlist.add("pi"+count);

265

266             count++;

```

```

261     }
262
263
264     }
265
266     }
267
268     //adding side cutting blocks
269
270     model.component("compl1").geom("geom1").create("blk1", "Block");
271
272     model.component("compl1").geom("geom1").feature("blk1").set("pos", new String[]{"-D",
273         "-D/2", "0"});
274
275     model.component("compl1").geom("geom1").feature("blk1").set("size", new String[]{"1",
276         gridsize+"*D", "1"});
277
278     model.component("compl1").geom("geom1").create("blk2", "Block");
279
280     model.component("compl1").geom("geom1").feature("blk2").set("pos", new String[]{"-D",
281         "-D", "0"});
282
283     model.component("compl1").geom("geom1").feature("blk2").set("size", new String[]{"1",
284         gridsize+1+"*D", "1", "1"});
285
286     model.component("compl1").geom("geom1").create("blk3", "Block");
287
288     model.component("compl1").geom("geom1").feature("blk3").set("pos", new String[]{(
289         gridsize+"*D-D/2", "-D/2", "0"});
290
291     model.component("compl1").geom("geom1").feature("blk3").set("size", new String[]{"1",
292         gridsize+"*D", "1"});
293
294     model.component("compl1").geom("geom1").create("blk4", "Block");
295
296     model.component("compl1").geom("geom1").feature("blk4").set("pos", new String[]{"-D",
297         gridsize+"*D-D/2", "0"});
298
299     model.component("compl1").geom("geom1").feature("blk4").set("size", new String[]{(
300         gridsize+1+"*D", "1", "1"});
301
302     model.component("compl1").geom("geom1").create("dif1", "Difference");
303
304     model.component("compl1").geom("geom1").feature("dif1").selection("input").set(
305         selectionlist.toArray(new String[0]));
306
307     model.component("compl1").geom("geom1").feature("dif1").selection("input2").set("blk1",
308         "blk2", "blk3", "blk4");

```

```

288     model.component("compl").geom("geom1").feature("dif1").set("intbnd", false);

289

290     int boxselcount = 0;

291

292     //invert geometry for gas flow

293     if (fluidn == 1) {

294         model.component("compl").geom("geom1").create("cyl1", "Cylinder");

295         model.component("compl").geom("geom1").feature("cyl1").set("r", "1/2");

296         model.component("compl").geom("geom1").create("arr2", "Array");

297         model.component("compl").geom("geom1").feature("arr2").set("fullsize", new int[]{

298             gridsize, gridsize, 1});

299         model.component("compl").geom("geom1").feature("arr2").set("displ", new int[]{2,

300             2, 0});

301         model.component("compl").geom("geom1").feature("arr2").selection("input").set("

302             cyl1");

303

304

305         ArrayList<String> selectionlist2 = new ArrayList<String>();

306         selectionlist2.add("dif1");

307         selectionlist2.add("arr2");

308

309         model.component("compl").geom("geom1").create("boxsel"+boxselcount, "BoxSelection"

310             );

311         boxselcount++;

312

313         model.component("compl").geom("geom1").create("blk5", "Block");

314         model.component("compl").geom("geom1").feature("blk5").set("pos", new int[]{-1,

315             -1, 0});

316         model.component("compl").geom("geom1").feature("blk5").set("size", new int[]{2*


317             gridsize, 2*gridsize, 1});

318         model.component("compl").geom("geom1").create("dif2", "Difference");

319         model.component("compl").geom("geom1").feature("dif2").selection("input").set("

320             blk5");

321         model.component("compl").geom("geom1").feature("dif2").selection("input2").set(


322             selectionlist2.toArray(new String[0]));

323

324

325         //move part of the network to the otherside to complete gas backbone

326         model.component("compl").geom("geom1").create("arr3", "Array");

```

```

317     model.component("comp1").geom("geom1").feature("arr3").selection("input").set("
318         dif2");
319     model.component("comp1").geom("geom1").feature("arr3").set("fullsize", new int
320         []{2, 2, 1});
321     model.component("comp1").geom("geom1").feature("arr3").set("displ", new int[]{-2*
322         gridsize, -2*gridsize, 0});
323
324     model.component("comp1").geom("geom1").create("blk6", "Block");
325     model.component("comp1").geom("geom1").feature("blk6").set("pos", new int[]{-2,
326         -2, 0});
327     model.component("comp1").geom("geom1").feature("blk6").set("size", new int[]{2*
328         gridsize, 2*gridsize, 1});
329
330     model.component("comp1").geom("geom1").create("unil", "Union");
331     model.component("comp1").geom("geom1").feature("unil").selection("input").set("
332         arr3");
333
334     // remove the edge that is created by the union
335     model.component("comp1").geom("geom1").create("int1", "Intersection");
336     model.component("comp1").geom("geom1").feature("int1").selection("input").set("
337         blk6");
338     model.component("comp1").geom("geom1").feature("int1").selection("input").set("
339         blk6", "unil");
340
341     model.component("comp1").geom("geom1").feature("int1").set("intbnd", false);
342
343
344     model.component("comp1").geom("geom1").create("boxsel"+boxselcount, "BoxSelection"
345         );
346
347     model.component("comp1").geom("geom1").feature("boxsel"+boxselcount).set("
348         entitydim", 1);
349
350     model.component("comp1").geom("geom1").feature("boxsel"+boxselcount).set("xmin",
351         -1.01);
352
353     model.component("comp1").geom("geom1").feature("boxsel"+boxselcount).set("xmax",
354         -0.99);
355
356     model.component("comp1").geom("geom1").feature("boxsel"+boxselcount).set("
357         condition", "inside");
358
359
360     model.component("comp1").geom("geom1").create("igel1", "IgnoreEdges");

```

```

341     model.component("comp1").geom("geom1").feature("ige1").selection("input").named("
342         boxsel"+boxselcount);
343
344     model.component("comp1").geom("geom1").create("boxsel"+boxselcount, "BoxSelection"
345         );
346     model.component("comp1").geom("geom1").feature("boxsel"+boxselcount).set("
347         entitydim", 1);
348     model.component("comp1").geom("geom1").feature("boxsel"+boxselcount).set("ymin",
349         -1.01);
350     model.component("comp1").geom("geom1").feature("boxsel"+boxselcount).set("ymax",
351         -0.99);
352     model.component("comp1").geom("geom1").feature("boxsel"+boxselcount).set("
353         condition", "inside");
354
355     model.component("comp1").geom("geom1").create("ige2", "IgnoreEdges");
356     model.component("comp1").geom("geom1").feature("ige2").selection("input").named("
357         boxsel"+boxselcount);
358     boxselcount++;
359
360     if (porebodies[0][0].equals("water")) {
361         model.component("comp1").geom("geom1").create("blk7", "Block");
362         model.component("comp1").geom("geom1").feature("blk7").set("pos", new int[]{-2,
363             0, 0});
364         model.component("comp1").geom("geom1").feature("blk7").set("pos", new int[]{-2,
365             -2, 0});
366
367         model.component("comp1").geom("geom1").create("dif3", "Difference");
368         model.component("comp1").geom("geom1").feature("dif3").selection("input").set("
369             int1");
370         model.component("comp1").geom("geom1").feature("dif3").selection("input2").set("
371             blk7");
372     }
373 }
374
375
376
377     model.component("comp1").geom("geom1").selection().create("csel1", "

```

```

            CumulativeSelection"); //create cumulative selection for the upper boundary

367 model.component("compl").geom("geom1").selection("csel1").label("Cumulative_
Selection_1");

368

369 model.component("compl").geom("geom1").selection().create("csel2", "
CumulativeSelection"); //create cumulative selection for the outlets if left-
right flow direction

370 model.component("compl").geom("geom1").selection("csel2").label("Cumulative_
Selection_2");

371

372 model.component("compl").geom("geom1").selection().create("csel3", "
CumulativeSelection"); //create cumulative selection for the outlets if top-
bottom flow direction

373 model.component("compl").geom("geom1").selection("csel3").label("Cumulative_
Selection_3");

374

375

376 //upper boundary

377 model.component("compl").geom("geom1").create("boxsel"+boxselcount, "BoxSelection");
model.component("compl").geom("geom1").feature("boxsel"+boxselcount).set("zmin",
0.9);

379 model.component("compl").geom("geom1").feature("boxsel"+boxselcount).set("entitydim",
2);

380 model.component("compl").geom("geom1").feature("boxsel"+boxselcount).set("condition",
"inside");

381

382 model.component("compl").geom("geom1").feature("boxsel"+boxselcount).set("
contributeto", "cse1"); //contribute to cumulative selection used for inlets

383

384 boxselcount++;

385

386 if (fluiddn == 0) {

387

388 //merging faces and removing edges

389

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

```

```

391     model.component("comp1").geom("geom1").feature("boxsel"+boxselcount).set("entitydim", 2);

392     model.component("comp1").geom("geom1").feature("boxsel"+boxselcount).set("zmax",
393         0.01);

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

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

399     boxselcount++;

400

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

402     model.component("comp1").geom("geom1").feature("boxsel"+boxselcount).set("zmin",
403         0.99);

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

406     model.component("comp1").geom("geom1").feature("boxsel"+boxselcount).set("entitydim", 2);

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

410     boxselcount++;

411

412

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

415

416

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

```

```

        );

418    model.component("comp1").geom("geom1").feature("boxsel"+boxselcount).set(
        "entitydim", 1);

419    model.component("comp1").geom("geom1").feature("boxsel"+boxselcount).set("xmin",
        -0.9);

420    model.component("comp1").geom("geom1").feature("boxsel"+boxselcount).set("xmax",
        2*gridsize-1.1);

421    model.component("comp1").geom("geom1").feature("boxsel"+boxselcount).set("ymin",
        -0.9);

422    model.component("comp1").geom("geom1").feature("boxsel"+boxselcount).set("ymax",
        2*gridsize-1.1);

423    model.component("comp1").geom("geom1").feature("boxsel"+boxselcount).set("zmin",
        0.3);

424    model.component("comp1").geom("geom1").feature("boxsel"+boxselcount).set("zmax",
        0.9);

425    model.component("comp1").geom("geom1").feature("boxsel"+boxselcount).set(
        "contributeto", "csel4");

426

427    boxselcount++;

428

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

430    model.component("comp1").geom("geom1").feature("boxsel"+boxselcount).set(
        "entitydim", 1);

431    model.component("comp1").geom("geom1").feature("boxsel"+boxselcount).set("xmin",
        -0.9);

432    model.component("comp1").geom("geom1").feature("boxsel"+boxselcount).set("xmax",
        2*gridsize-1.1);

433    model.component("comp1").geom("geom1").feature("boxsel"+boxselcount).set("ymin",
        -0.9);

434    model.component("comp1").geom("geom1").feature("boxsel"+boxselcount).set("ymax",
        2*gridsize-1.1);

435    model.component("comp1").geom("geom1").feature("boxsel"+boxselcount).set("zmin",
        0.01);

436    model.component("comp1").geom("geom1").feature("boxsel"+boxselcount).set("zmax",
        0.2);

437    model.component("comp1").geom("geom1").feature("boxsel"+boxselcount).set(

```

```

        contributeTo", "csel4");

438

439     boxselcount++;

440

441     model.component("comp1").geom("geom1").create("igel1", "IgnoreEdges");

442     model.component("comp1").geom("geom1").feature("igel1").selection("input").named("csel4");

443

444     //selection for the no slip boundary condition

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

446         );

447     model.component("comp1").geom("geom1").feature("boxsel"+boxselcount).set("entitydim", 2);

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

449     model.component("comp1").geom("geom1").feature("boxsel"+boxselcount).set("zmax", 0.3);

450

451     //setting Slip BC on all boundaries

452     model.component("comp1").physics("spf").feature("wallbc1").set("BoundaryCondition", "Slip");

453

454     //setting No-Slip on bottom plate and pillars which overrides the slip BC.

455     model.component("comp1").physics("spf").create("wallbc2", "WallBC", 2);

456     model.component("comp1").physics("spf").feature("wallbc2").selection().named("geom1_boxsel"+boxselcount);

457     model.component("comp1").physics("spf").feature("wallbc2").set("BoundaryCondition", "NoSlip");

458

459     boxselcount++;

460

461     if (fluidn == 1) {

462         //merging faces and removing edges

463

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

```

```

465     model.component("comp1").geom("geom1").feature("boxsel"+boxselcount).set("
466         entitydim", 1);
467
468     model.component("comp1").geom("geom1").feature("boxsel"+boxselcount).set("xmin",
469         -1.999);
470
471     model.component("comp1").geom("geom1").feature("boxsel"+boxselcount).set("xmax",
472         2*gridsize-2.001);
473
474     model.component("comp1").geom("geom1").feature("boxsel"+boxselcount).set("ymin",
475         -1.999);
476
477     model.component("comp1").geom("geom1").feature("boxsel"+boxselcount).set("ymax",
478         2*gridsize-2.001);
479
480     model.component("comp1").geom("geom1").feature("boxsel"+boxselcount).set("zmax",
481         0.5);
482
483
484     model.component("comp1").geom("geom1").feature("boxsel"+boxselcount).set("
485         condition", "intersects");
486
487
488     // merge all bottom horizontal faces
489
490     model.component("comp1").geom("geom1").create("ige3", "IgnoreEdges");
491
492     model.component("comp1").geom("geom1").feature("ige3").selection("input").named("
493         boxsel"+boxselcount);
494
495
496     boxselcount++;
497
498
499     }
500
501
502     //ignore vertices to aid mesh creation
503
504     model.component("comp1").geom("geom1").create("boxsel"+boxselcount, "BoxSelection");
505
506     model.component("comp1").geom("geom1").feature("boxsel"+boxselcount).set("entitydim"
507         , 0);
508
509
510     model.component("comp1").geom("geom1").create("igv1", "IgnoreVertices");
511
512     model.component("comp1").geom("geom1").feature("igv1").selection("input").named("
513         boxsel"+boxselcount);
514
515
516     boxselcount++;
517
518
519     }
520
521
522     boxselcount++;
523
524
525     }
526
527
528     boxselcount++;
529
530
531     }
532
533
534     boxselcount++;
535
536
537     }
538
539
540     boxselcount++;
541
542
543     }
544
545
546     boxselcount++;
547
548
549     }
550
551
552     boxselcount++;
553
554
555     }
556
557
558     boxselcount++;
559
560
561     }
562
563
564     boxselcount++;
565
566
567     }
568
569
570     boxselcount++;
571
572
573     }
574
575
576     boxselcount++;
577
578
579     }
580
581
582     boxselcount++;
583
584
585     }
586
587
588     boxselcount++;
589
590
591     }
592
593
594     boxselcount++;
595
596
597     }
598
599
599     boxselcount++;
600
601
602     }
603
604
605     boxselcount++;
606
607
608     }
609
610
611     boxselcount++;
612
613
614     }
615
616
617     boxselcount++;
618
619
619     }
620
621
622     boxselcount++;
623
624
625     }
626
627
628     boxselcount++;
629
630
631     }
632
633
634     boxselcount++;
635
636
637     }
638
639
639     boxselcount++;
640
641
642     }
643
644
645     boxselcount++;
646
647
648     }
649
650
651     boxselcount++;
652
653
654     }
655
656
657     boxselcount++;
658
659
659     }
660
661
662     boxselcount++;
663
664
665     }
666
667
668     boxselcount++;
669
670
671     }
672
673
674     boxselcount++;
675
676
677     }
678
679
679     boxselcount++;
680
681
682     }
683
684
685     boxselcount++;
686
687
688     }
689
690
691     boxselcount++;
692
693
694     }
695
696
697     boxselcount++;
698
699
699     }
700
701
702     boxselcount++;
703
704
705     }
706
707
708     boxselcount++;
709
710
711     }
712
713
714     boxselcount++;
715
716
717     }
718
719
719     boxselcount++;
720
721
722     }
723
724
725     boxselcount++;
726
727
728     }
729
730
731     boxselcount++;
732
733
734     }
735
736
737     boxselcount++;
738
739
739     }
740
741
742     boxselcount++;
743
744
745     }
746
747
748     boxselcount++;
749
750
751     }
752
753
754     boxselcount++;
755
756
757     }
758
759
759     boxselcount++;
760
761
762     }
763
764
765     boxselcount++;
766
767
768     }
769
770
771     boxselcount++;
772
773
774     }
775
776
777     boxselcount++;
778
779
779     }
780
781
782     boxselcount++;
783
784
785     }
786
787
788     boxselcount++;
789
790
791     }
792
793
794     boxselcount++;
795
796
797     }
798
799
799     boxselcount++;
800
801
802     }
803
804
805     boxselcount++;
806
807
808     }
809
810
811     boxselcount++;
812
813
814     }
815
816
817     boxselcount++;
818
819
819     }
820
821
822     boxselcount++;
823
824
825     }
826
827
828     boxselcount++;
829
830
831     }
832
833
834     boxselcount++;
835
836
837     }
838
839
839     boxselcount++;
840
841
842     }
843
844
845     boxselcount++;
846
847
848     }
849
850
851     boxselcount++;
852
853
854     }
855
856
857     boxselcount++;
858
859
859     }
860
861
862     boxselcount++;
863
864
865     }
866
867
868     boxselcount++;
869
870
871     }
872
873
874     boxselcount++;
875
876
877     }
878
879
879     boxselcount++;
880
881
882     }
883
884
885     boxselcount++;
886
887
888     }
889
890
891     }

```



```

520     model.component("comp1").geom("geom1").feature("boxsel"+boxselcount).set("
521         contributeto", "csel2"); //contribute to cumulative selection used for inlets
522
523     boxsellist1.add(boxselcount);
524
525     boxselcount++;
526
527     //outlet side selections
528
529     model.component("comp1").geom("geom1").create("boxsel"+boxselcount, "BoxSelection"
530         );
531
532     model.component("comp1").geom("geom1").feature("boxsel"+boxselcount).set("
533         entitydim", 2);
534
535     model.component("comp1").geom("geom1").feature("boxsel"+boxselcount).set(
536         direction_indicator1+"min", gridsize-1+*D+0.99+fluidoffset);
537
538     model.component("comp1").geom("geom1").feature("boxsel"+boxselcount).set(
539         direction_indicator2+"min", 2*boundn-1.1+fluidoffset);
540
541     model.component("comp1").geom("geom1").feature("boxsel"+boxselcount).set(
542         direction_indicator2+"max", 2*boundn+1.1+fluidoffset);
543
544     model.component("comp1").geom("geom1").feature("boxsel"+boxselcount).set("
545         condition", "inside");
546
547     model.component("comp1").geom("geom1").feature("boxsel"+boxselcount).set("
548         contributeto", "csel3"); //contribute to cumulative selection used for inlets
549
550     boxsellist2.add(boxselcount);
551
552     boxselcount++;
553
554 }
555
556 //JFrame jFrame = new JFrame();
557
558 // JOptionPane.showMessageDialog(jFrame, surfaceIntegrals);
559
560
561 // ADD Material properties: water & CO2
562
563 model.component("comp1").material().create("mat1", "Common");
564
565
566 if (fluidn == 0) {
567
568     model.component("comp1").material("mat1").label("H2O");
569
570     model.component("comp1").material("mat1").propertyGroup("def").set("density", new

```

```

        String[]{"1000"});  

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

550}  

551  

552 if (fluidn == 1) {  

553    model.component("comp1").material("mat1").label("H2O");  

554    model.component("comp1").material("mat1").propertyGroup("def").set("density", new  

555        String[]{"1000"});  

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

557    //If NWP is CO2  

558    //model.component("comp1").material("mat1").label("CO2");  

559    //model.component("comp1").material("mat1").propertyGroup("def").set("density",  

560        new String[]{"1.84104"});  

561    //model.component("comp1").material("mat1").propertyGroup("def").set("dynamicviscosity", new String[]{"0.0000146885"});  

562}  

563  

564 ArrayList<Integer> selectn = new ArrayList<Integer>();  

565 selectn.add(1);  

566 selectn.add(2);  

567 selectn.add(3);  

568 selectn.add(4);  

569  

570 String path_filename = "";  

571  

572 if (fluidn == 0) {  

573    if (directionn == 0) {  

574        path_filename = "\\\\"+pn+"\\x_paths_water";  

575    }  

576    if (directionn == 1) {  

577        path_filename = "\\\\"+pn+"\\y_paths_water";  

578    }  

579    if (fluidn == 1) {  

580        if (directionn == 0) {  


```

```

580         path_filename = "x_paths_gas";
581     }
582     if (directionn == 1) {
583         path_filename = "y_paths_gas";
584     }
585 }
586
587 String[][] connected_indexes = loadFileToArray(gridsize, "Seed\u209c"+seedn+"\\"+
588         path_filename+".txt");
589
590 // set symmetry on top surface
591 model.component("comp1").physics("spf").create("sym1", "Symmetry", 2);
592 model.component("comp1").physics("spf").feature("sym1").selection().named(
593         "geom1_csell_bnd");
594
595 int counter = 1;
596
597 for (Integer boxsel_n = 0; boxsel_n < boxesellist1.size(); boxsel_n++) {
598     for (int i = 0; i < connected_indexes[0].length; i++) {
599
600         int path_index = Integer.parseInt(connected_indexes[0][i])/2;
601
602         if (boxsel_n == path_index) {
603
604             //set inlets
605             //JFrame jFrame = new JFrame();
606             // JOptionPane.showMessageDialog(jFrame, boxsel_n.toString()+connected_indexes
607             [0][i]);
608
609             model.component("comp1").physics("spf").create("inl"+counter, "InletBoundary",
610                 2);
611             model.component("comp1").physics("spf").feature("inl"+counter).set(
612                 "BoundaryCondition", "Pressure");
613             model.component("comp1").physics("spf").feature("inl"+counter).set("p0",
614                 gridsize);
615             model.component("comp1").physics("spf").feature("inl"+counter).selection();

```

```

611
612         //set outlets
613
614         model.component("compl").physics("spf").create("out"+counter, "OutletBoundary"
615
616         , 2);
617
618         model.component("compl").physics("spf").feature("out"+counter).selection().
619
620         named("geom1_boxsel"+boxsellist1.get(boxsel_n));
621
622         counter++;
623
624         break;
625
626     }
627
628 }
629
630
631
632         //set mesh
633
634
635         model.component("compl").mesh("mesh1").automatic(false);
636
637
638         model.component("compl").mesh("mesh1").feature("size").set("hauto", 9);

```

```

639
640     model.component("compl").mesh("mesh1").create("size1", "Size");
641
642     model.component("compl").mesh("mesh1").feature("size1").set("hauto", 9);
643
644     model.component("compl").mesh("mesh1").feature("size1").selection().named("geom1_boxsel"+boxselcount);
645
646
647     model.component("compl").mesh("mesh1").feature("size1").set("table", "cfld");
648
649
650     model.component("compl").mesh("mesh1").create("ftet1", "FreeTet");
651
652
653     //Solve
654
655     model.study().create("std1");
656
657     model.study("std1").create("stat", "Stationary");
658
659     model.study("std1").feature("stat").activate("spf", true);
660
661
662     model.sol().create("sol1");
663
664     model.sol("sol1").study("std1");
665
666     model.study("std1").feature("stat").set("notlistsolnum", 1);
667
668     model.study("std1").feature("stat").set("notsolnum", "1");
669
670     model.study("std1").feature("stat").set("listsolnum", 1);
671
672     model.study("std1").feature("stat").set("solnum", "1");
673
674     model.sol("sol1").create("st1", "StudyStep");
675
676     model.sol("sol1").feature("st1").set("study", "std1");
677
678     model.sol("sol1").feature("st1").set("studystep", "stat");
679
680     model.sol("sol1").create("v1", "Variables");
681
682     model.sol("sol1").feature("v1").set("control", "stat");
683
684     model.sol("sol1").create("s1", "Stationary");
685
686     model.sol("sol1").feature("s1").set("stol", 0.2);
687
688     model.sol("sol1").feature("s1").feature("aDef").set("cachepattern", true);
689
690     model.sol("sol1").feature("s1").create("fc1", "FullyCoupled");
691
692     model.sol("sol1").feature("s1").feature("fc1").set("dtech", "auto");
693
694     model.sol("sol1").feature("s1").feature("fc1").set("initstep", 0.01);
695
696     model.sol("sol1").feature("s1").feature("fc1").set("minstep", 1.0E-4);
697
698     model.sol("sol1").feature("s1").feature("fc1").set("maxiter", 100);
699
700     model.sol("sol1").feature("s1").create("i1", "Iterative");
701
702     model.sol("sol1").feature("s1").feature("i1").set("linsolver", "gmres");
703
704     model.sol("sol1").feature("s1").feature("i1").set("prefuntype", "left");
705
706     model.sol("sol1").feature("s1").feature("i1").set("itrestart", 50);

```

```

675     model.sol("sol1").feature("s1").feature("il").set("rhob", 20);

676     model.sol("sol1").feature("s1").feature("il").set("maxlimit", 1000);

677     model.sol("sol1").feature("s1").feature("il").set("nlinnormuse", "on");

678     model.sol("sol1").feature("s1").feature("il").label("AMG,_fluid_flow_variables_(spf)

679     ");

680     model.sol("sol1").feature("s1").feature("il").create("mg1", "Multigrid");

681     model.sol("sol1").feature("s1").feature("il").feature("mg1").set("prefun", "saamg");

682     model.sol("sol1").feature("s1").feature("il").feature("mg1").set("mgcycle", "v");

683     model.sol("sol1").feature("s1").feature("il").feature("mg1").set("maxcoarsedof",

684     80000);

685     model.sol("sol1").feature("s1").feature("il").feature("mg1").set("strconn", 0.02);

686     model.sol("sol1").feature("s1").feature("il").feature("mg1").set("nullspace", "constant");

687     model.sol("sol1").feature("s1").feature("il").feature("mg1").set("usesmooth", false);

688     ;

689     model.sol("sol1").feature("s1").feature("il").feature("mg1").set("saamgcompwise", true);

690     model.sol("sol1").feature("s1").feature("il").feature("mg1").set("loweramg", true);

691     model.sol("sol1").feature("s1").feature("il").feature("mg1").feature("pr").create("val", "Vanka");

692     model.sol("sol1").feature("s1").feature("il").feature("mg1").feature("pr").feature("val")

693     .set("linesweeptype", "ssor");

694     model.sol("sol1").feature("s1").feature("il").feature("mg1").feature("pr").feature("val")

695     .set("iter", 0);

696     model.sol("sol1").feature("s1").feature("il").feature("mg1").feature("pr").feature("val")

697     .set("vankarelax", 0.8);

698     model.sol("sol1").feature("s1").feature("il").feature("mg1").feature("pr").feature("val")

699     .set("vankavars", new String[]{"compl_p"});

700     model.sol("sol1").feature("s1").feature("il").feature("mg1").feature("pr").feature("val")

701     .set("seconditer", 1);

702     model.sol("sol1").feature("s1").feature("il").feature("mg1").feature("pr").feature("val")

703     .set("relax", 0.5);

704     model.sol("sol1").feature("s1").feature("il").feature("mg1").feature("pr").create("val", "Vanka");

705     model.sol("sol1").feature("s1").feature("il").feature("mg1").feature("pr").feature("val")

```

```

        val")
699     .set("linesweeptype", "ssor");
700
model.sol("sol1").feature("s1").feature("i1").feature("mg1").feature("po").feature("
    val").set("iter", 1);

model.sol("sol1").feature("s1").feature("i1").feature("mg1").feature("po").feature("
    val").set("vankarelax", 0.8);

model.sol("sol1").feature("s1").feature("i1").feature("mg1").feature("po").feature("
    val")

703     .set("vankavars", new String[]{"compl_p"});
704
model.sol("sol1").feature("s1").feature("i1").feature("mg1").feature("po").feature("
    val").set("seconditer", 2);

model.sol("sol1").feature("s1").feature("i1").feature("mg1").feature("po").feature("
    val").set("relax", 0.5);

706 model.sol("sol1").feature("s1").feature("i1").feature("mg1").feature("cs").create("
    d1", "Direct");

707
model.sol("sol1").feature("s1").feature("i1").feature("mg1").feature("cs").feature("
    d1")

708     .set("linsolver", "pardiso");
709
model.sol("sol1").feature("s1").feature("i1").feature("mg1").feature("cs").feature("
    d1")

710     .set("pivotperturb", 1.0E-13);

711 model.sol("sol1").feature("s1").create("d1", "Direct");

712 model.sol("sol1").feature("s1").feature("d1").set("linsolver", "pardiso");
713
model.sol("sol1").feature("s1").feature("d1").set("pivotperturb", 1.0E-13);
714
model.sol("sol1").feature("s1").feature("d1").label("Direct,_fluid_flow_variables_(
    spf");

715 model.sol("sol1").feature("fc1").set("linsolver", "i1");
716
model.sol("sol1").feature("s1").feature("fc1").set("dtech", "auto");
717
model.sol("sol1").feature("s1").feature("fc1").set("initstep", 0.01);
718
model.sol("sol1").feature("s1").feature("fc1").set("minstep", 1.0E-4);
719
model.sol("sol1").feature("s1").feature("fc1").set("maxiter", 100);
720
model.sol("sol1").feature("s1").feature().remove("fcDef");
721
model.sol("sol1").attach("std1");

722
723 //create velocity plot
724
model.result().dataset("dset1").set("geom", "geom1");
725
model.result().create("pg1", "PlotGroup3D");

```

```

726     model.result("pg1").label("Velocity_(spf)");
727
728     model.result("pg1").set("frametype", "spatial");
729
730     model.result("pg1").feature().create("slc1", "Slice");
731
732     model.result("pg1").feature("slc1").label("Slice");
733
734     model.result("pg1").feature("slc1").set("smooth", "internal");
735
736     model.result("pg1").feature("slc1").set("data", "parent");
737
738
739
740     //create pressure plot
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762

```

- 726 model.result("pg1").label("Velocity_(spf)");
- 727
- 728 model.result("pg1").set("frametype", "spatial");
- 729
- 730 model.result("pg1").feature().create("slc1", "Slice");
- 731 model.result("pg1").feature("slc1").label("Slice");
- 732 model.result("pg1").feature("slc1").set("smooth", "internal");
- 733
- 734 model.result("pg1").feature("slc1").set("data", "parent");
- 735
- 736 //create pressure plot
- 737
- 738 model.result().create("pg2", "PlotGroup3D");
- 739
- 740 model.result("pg2").label("Pressure_(spf)");
- 741 model.result("pg2").set("frametype", "spatial");
- 742
- 743
- 744 model.result("pg2").feature().create("con1", "Contour");
- 745
- 746 model.result("pg2").feature("con1").label("Pressure");
- 747
- 748 model.result("pg2").feature("con1").set("expr", "p");
- 749
- 750 model.result("pg2").feature("con1").set("number", 20);
- 751 model.result("pg2").feature("con1").set("levelrounding", true);
- 752
- 753 model.result("pg2").feature("con1").set("data", "dset1");
- 754
- 755 model.result("pg2").feature("con1").set("contourtype", "filled");
- 756
- 757
- 758 model.sol("sol1").runAll();
- 759
- 760
- 761
- 762 //create horizontal velocity plane

- 750 model.result("pg1").feature("slc1").set("quickplane", "xy");
- 751
- 752 model.result("pg1").feature("slc1").set("quickznumber", 1);
- 753 model.result("pg1").feature("slc1").set("interactive", true);
- 754
- 755 String sliceHeight = "-3.8E-7";
- 756
- 757 if (fluidn == 1) {
- 758 sliceHeight = "5.0E-7";
- 759 }
- 760 model.result("pg1").feature("slc1").set("shift", sliceHeight);
- 761
- 762 //create surface integral east side

```

763
764     model.result().numerical().create("int1", "IntSurface");
765     model.result().numerical("int1").set("intvolume", true);
766
767     if (fluidn == 0) {
768         model.result().numerical("int1").selection().named("geom1_csel2_bnd");
769     }
770
771     if (fluidn == 1) {
772         model.result().numerical("int1").selection().named("geom1_csel3_bnd");
773     }
774
775     model.result("pg1").run();
776
777     //save result to text file
778     model.result().numerical("int1").setIndex("expr", "spf.U", 0);
779     model.result().table().create("tbl1", "Table");
780     model.result().table("tbl1").comments("Surface_Integration_2");
781     model.result().numerical("int1").set("table", "tbl1");
782     model.result().numerical("int1"). setResult();
783     model.result().table("tbl1")
784         .save("C:\\\\Users\\\\eobbens\\\\Master_Thesis_Grids\\\\Gridsize_" + gridsize + "\\Seed_"+
785             seedn + "\\P" + pn + "\\table_" + directionn + "_" + fluidn + ".txt");
786
787     //export first image
788     model.result().export().create("img1", "pg1", "Image");
789     model.result().export("img1").set("size", "manualweb");
790     model.result().export("img1").set("zoomextents", true);
791     model.result().export("img1").set("options3d", true);
792     model.result().export("img1").set("title3d", false);
793     model.result().export("img1").set("grid", false);
794     model.result().export("img1").set("axisorientation", false);
795     model.result().export("img1").set("logo3d", false);
796     model.result().export("img1").set("lockview", "on");
797     model.result().export("img1").set("view", "view2");
798

```

```

799     model.result().export("img1")
800         .set("pngfilename", "C:\\\\Users\\\\eobbens\\\\Master_Thesis_Grids\\\\Gridsize_" + gridsize +
801             "\\\\Seed_" + seedn + "\\P" + pn + "\\Velocity_Profile_" + directionn + "_" + fluidn + ".png");
802
803
804     //export second image
805
806     model.result().export().create("img2", "pg2", "Image");
807
808     model.result().export("img2").set("size", "manualweb");
809
810     model.result().export("img2").set("zoomextents", true);
811
812     model.result().export("img2").set("options3d", true);
813
814     model.result().export("img2").set("title3d", false);
815
816     model.result().export("img2").set("grid", false);
817
818     model.result().export("img2").set("axisorientation", false);
819
820     model.result().export("img2").set("logo3d", false);
821
822     model.result().export("img2").set("lockview", "on");
823
824     model.result().export("img2").set("view", "view2");
825
826
827     model.result().export("img2")
828         .set("pngfilename", "C:\\\\Users\\\\eobbens\\\\Master_Thesis_Grids\\\\Gridsize_" + gridsize +
829             "\\\\Seed_" + seedn + "\\P" + pn + "\\Pressure_Profile_" + directionn + "_" + fluidn + ".png");
830
831
832     model.result().run();
833
834
835
836     if (deletenodes == true) {
837
838         model.component().remove("comp1");
839
840         model.study().remove("std1");
841
842         model.result().table().remove("tbl1");
843
844         model.result().export().remove("img1");
845
846         model.result().remove("pg1");
847
848         model.result().numerical().remove("int1");
849
850         model.result().remove("pg2");
851
852         model.result().remove("pg1");
853
854         model.result().export().remove("img2");
855
856         model.result().dataset().remove("surf1");
857
858         model.result().dataset().remove("dset1");
859
860     }

```

```

834
835         }
836
837         catch (Exception e) {
838             JOptionPane.showMessageDialog(null, ""+e);
839
840             model.component().remove("comp1");
841
842             model.study().remove("std1");
843
844             model.result().table().remove("tbl1");
845
846             model.result().export().remove("img1");
847
848             model.result().remove("pg1");
849
850             model.result().numerical().remove("int1");
851
852             model.result().remove("pg2");
853
854             model.result().remove("pg1");
855
856             model.result().export().remove("img2");
857
858             model.result().dataset().remove("surf1");
859
860             model.result().dataset().remove("dset1");
861
862         }
863     }
864
865     continue;
866
867 }
868
869 } //directionn
870
871
872 } //fluidn
873
874
875 } //pn
876
877 } //seed
878
879 }
880
881 }
882 }
```

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

```

1 import numpy as np
2 import matplotlib as mpl
3 import matplotlib.pyplot as plt
4 import seaborn as sns
```

```

5 import os
6 import pandas as pd
7 from scipy.stats import hmean
8
9 #figure resolution
10 mpl.rcParams['figure.dpi']= 375
11
12 sns.set_style("whitegrid")
13 sns.set_style(rc={'ytick.left': True})
14
15 mpl.rc('font',family='Times_New_Roman')
16 mpl.rcParams['mathtext.fontset'] = 'custom'
17 mpl.rcParams['mathtext.rm'] = 'Times_New_Roman'
18 mpl.rcParams['mathtext.it'] = 'Times_New_Roman:italic'
19 mpl.rcParams['mathtext.bf'] = 'Times_New_Roman:bold'
20
21 mpl.rcParams.update({'figure.autolayout': True})
22
23 font_size = 30
24
25 directory = 'Network_images_brdif'
26 all_folders = [folder[0] for folder in os.walk(directory)]
27
28 df_water = pd.DataFrame()
29 df_gas = pd.DataFrame()
30 grid_n = 0
31 seed_n = 0
32
33 p_n = 0
34
35 info_dict = {}
36
37 for folder in all_folders:
38     info_path = folder + '\\info.txt'
39     if os.path.exists(info_path):
40         with open(info_path,'r') as fin:
41             grid_n = int(folder[30])

```

```

42         seed_n = int(folder[37])
43
44         lines = fin.readlines()
45
46     for i in range(2):
47
48         for j in range(2):
49
50             fpath = folder + f'\\table_{i}_{j}.txt'
51
52             if os.path.exists(fpath):
53
54                 with open(fpath , 'r') as fin:
55
56                     lines = fin.readlines()
57
58                     flow_measurement = float(lines[5][:-1])
59
60
61                     grid_n = int(fpath[30])
62
63                     seed_n = int(fpath[37])
64
65                     p_n = int(fpath[40])
66
67                     #
68
69                     if i ==0:
70
71                         direction = 'hor'
72
73                     if i == 1:
74
75                         direction = 'ver'
76
77                     #fluid = 'water'
78
79                     if j == 0:
80
81                         df_water = df_water.append({{'Grid':grid_n,
82
83                                         'Seed':seed_n,
84
85                                         'P_n':p_n,
86
87                                         'Direction':direction,
88
89                                         'Q_water':flow_measurement}},ignore_index=True)
90
91
92                     if j == 1:
93
94                         #fluid = 'gas'
95
96                         Q = 'Q_gas'
97
98                         df_gas = df_gas.append({{'Grid':grid_n,
99
100                                         'Seed':seed_n,
101
102                                         'P_n':p_n,
103
104                                         'Direction':direction,
105
106                                         'Q_gas':Q}})
```

```

79                                     'Q_gas':flow_measurement}),ignore_index=True)
80
81 df_water = df_water.reindex(columns= ['Grid','Seed','P_n', 'Direction','Q_water'])
82
83 df_gas = df_gas.reindex(columns= ['Grid','Seed','P_n', 'Direction','Q_gas'])
84
85 df = pd.concat([df_water, df_gas], axis=1).drop_duplicates()
86
87 df = df.loc[:,~df.columns.duplicated()]
88
89 #flow rate through a microfluidic device of gridsize 1 filled completely with one fluid (water or
90 #co2)
91 fullwater = 2.1473E-16
92 fullgas = 1.4755E-14
93 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".
94 #df['Q_gas_full'] = df['Grid'] * fullgas
95 df['Q_water_full'] = df['Grid'] * fullwater
96
97 df['gas_rel_perm'] = df['Q_gas']/df['Q_water_full']
98 df['water_rel_perm'] = df['Q_water']/df['Q_water_full']
99 df['rel_perm_frac'] = df['water_rel_perm']/df['gas_rel_perm']
100
101 df['P_n'] = df['P_n']/10
102
103 fign= 0
104
105 figsize = (6,6)
106
107 tick_list = [0.1,0.2,0.3]
108
109 legend_fontsize = 24
110 plt.rcParams['legend.title_fontsize'] = legend_fontsize
111 ### WETTING PHASE
112 plt.figure(fign,figsize=figsize)
113 sns.scatterplot(data=df,x='P_n',y='water_rel_perm',color='grey',alpha=0.5)

```

```

114
115 plt.figure(fign,figsize=figsize)
116 sns.regplot(data=df,x='P_n',y='water_rel_perm',x_estimator=np.mean, ci=None,fit_reg=False,color='
    green',label='Arithmetic')
117
118 fig = plt.figure(fign,figsize=figsize)
119 ax = sns.regplot(data=df,x='P_n',y='water_rel_perm',x_estimator=hmean, ci=None,fit_reg=False,color='
    red', label='Harmonic')
120 plt.title('$k_{rel,WP}$',size=font_size)
121 plt.xlabel('$\Delta_p_{network}$',fontsize=font_size)
122 plt.xticks(size=font_size)
123 plt.yticks(size=font_size)
124 plt.yscale('log')
125
126 plt.ylim([10e-5,10e0])
127
128 ax.set_xticks(tick_list)
129 #ax.set_yticks(np.geomspace(10e-4, 10e-2 ,31))
130
131 #plt.legend(title = "Average")
132 fig.savefig('gz12WP.png')
133 fign += 1
134
135 ##### NON WETTING PHASE
136 plt.figure(fign,figsize=figsize)
137 sns.scatterplot(data=df,x='P_n',y='gas_rel_perm',color='grey',alpha=0.5)
138
139 plt.figure(fign,figsize=figsize)
140 sns.regplot(data=df,x='P_n',y='gas_rel_perm',x_estimator=np.mean, ci=None,fit_reg=False,color='green
    ',label='Arithmetic')
141
142 fig = plt.figure(fign,figsize=figsize)
143 ax = sns.regplot(data=df,x='P_n',y='gas_rel_perm',x_estimator=hmean, ci=None,fit_reg=False,color='
    red', label='Harmonic')
144 plt.title('$k_{rel,NWP}$',size=font_size)
145 plt.xlabel('$\Delta_p_{network}$',fontsize=font_size)
146 plt.xticks(size=font_size)

```

```

147 plt.yticks(size=font_size)
148 plt.yscale('log')
149 plt.ylim([10e-5,10e0])
150 ax.set_xticks(tick_list)
151 #plt.legend(title = "Average",loc=4)
152 fig.savefig('gz12NWP.png')
153 fign += 1
154
155 ### Fraction of rel perms
156 plt.figure(fign,figsize=figsize)
157 sns.scatterplot(data=df,x='P_n',y='rel_perm_frac',color='grey',alpha=0.5)
158
159 plt.figure(fign,figsize=figsize)
160 sns.regplot(data=df,x='P_n',y='rel_perm_frac',x_estimator=np.mean, ci=None,fit_reg=False,color='green',label='Arithmetic')
161
162 fig = plt.figure(fign,figsize=figsize)
163 ax = sns.regplot(data=df,x='P_n',y='rel_perm_frac',x_estimator=hmean, ci=None,fit_reg=False,color='red', label='Harmonic')
164 plt.title('$k_{rel,WP}/k_{rel,NWP}$',size=font_size)
165 plt.xlabel('$\Delta_p_{network}$',fontsize=font_size)
166 plt.xticks(size=font_size)
167 plt.yticks(size=font_size)
168 plt.yscale('log')
169 plt.ylim([10e-5,10e0])
170 ax.set_xticks(tick_list)
171 plt.legend(title = "Average",loc=4,fontsize = legend_fontsize)
172 fig.savefig('gz12frac.png')
173 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>.