Abstract

Improvements in imaging techniques have enabled the reconstruction of complex porous media which can be analyzed by computer simulations. The two most popular methods for numerical analysis of transport in porous media are direct numerical simulation (DNS) and pore network modeling (PNM). This work aims at assessing the suitability of these techniques to study dry and wet transport properties of porous transport layers for fuel cells and electrolyzers by comparing numerical predictions to experimental data for mercury intrusion, and transport properties.

Estimation of Relative Transport Properties in Porous Materials using Pore-Scale and Pore-Network Simulations

Justin Kinda

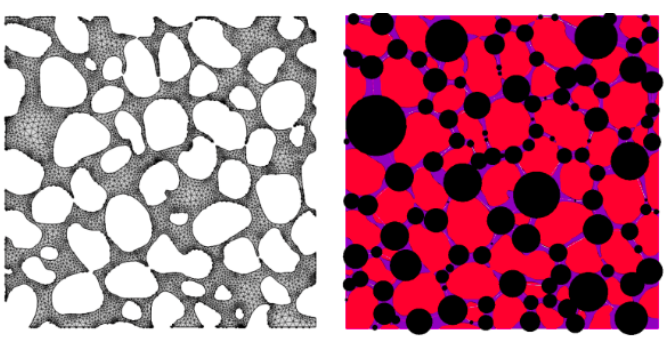
November 2022

1 Introduction

Porous materials play a vital role in many technologies and industries. Naturally occurring porous media such as rock and soil have been the subject of intense study in the context of oil recovery and aquifer management1. Engineered or manufactured porous materials have enabled breakthroughs in many fields such as concrete technology, electrodes, membranes, biomedical applications and more. Designing porous materials represents a classic optimization problem: the presence of the solid is necessary to provide some key function such as reactive surface area, but it also creates obstacles to flow and transport. A defining feature of all porous materials is that the pore structure can have an overriding impact on the transport processes.

Two materials with the same porosity will have dramatically different transport properties depending how the pores are spatially distributed, how they're connected, their shapes, size distribution, and so on.

Pore-scale models resolve the heterogeneity of the porous media by solving the governing equations of a given physical process. The two most commonly used pore-scale modeling approaches are pore network modeling (PNM) and direct numerical simulation (DNS). PNM treats the pores as nodes on a network and throats as the links, and geometric properties are assigned to each pore and throat based on the physical space they represent. One can either generate an artificial pore network or extract the network from imaging techniques, such as μCT. In PNM, transport inside the network is solved using a simplified 1D analytical solution of the relevant transport equations in each pore between pairs of connected pores. DNS, on the other hand, solves the transport equations on computational meshes generated from the tomography images instead of extracting idealized networks. In voxel-based mesh generation, a mesh can be generated by the direct conversion of the image voxels into mesh elements or the use of triangulation algorithms. 15 DNS does not rely on a simplified geometry, however, it needs more computational resources than PNM.



(a) FEM mesh (b) Equivalent PNM

Many studies have employed PNM to study the transport properties of porous media under dry and saturated conditions. The most common pore and throat shapes used in recent studies are spherical pores and cylindrical throats. It is possible and preferable to extract networks directly from μCT images. One of the most important structural metrics of a porous material is the connectivity of the pore space, or the so-called

pore network. This has bearing on the diffusive tortuosity, percolation of immiscible fluids, relative permeability, and so on. To this end, researchers are highly motivated to extract pore networks from tomographic images. Some studies employed watershed segmentation in order to extract a pore network from a 3D image of a microstructure. The watershed segmentation finds constrictions and separates the entire void phase into a set of local pore regions. These pore regions are then used to extract simplified spherical pores and cylindrical throats. [Gostick(2017)] validated a watershed-based network extraction algorithm and showed that the extraction provided acceptable predictions of the transport properties of fibrous media.

2 Mathematical modeling

2.1 Key concept

Transport models in OpenPNM treat the transport between two neighboring pores as 1D linear transport through three elements in series, i.e., half of each pore and the connecting throat. The conductance between two pores is determined by the geometric properties, i.e., radii, lengths, and shapes, of the elements and the fluid properties. A single value of conductance is determined for the conduit, and this determines the transport of a phase between the pores via analogy to Kirchhoff's law. At steady state and in the absence of reaction, the algebraic sum of transporting species entering and exiting each pore is zero, so a set of linear equations can be solved to find the quantity of interest in each pore, i.e., concentration, and the flow through each throat.

2.2 Image analysis

The stack of μCT radiographs was reconstructed using the back-projection algorithm implemented in the NRecon software (Bruker Inc.). Ring artifact reduction and beam hardening correction steps were performed to remove the artifacts in the CT scans. The initial size of each image stack was X × Y × Z voxels. In order to remove noise from the reconstructed stack, the bilateral filter implemented in the ImageJ software was applied. The filter smooths the images while preserving edges. The solid and void regions in the reconstructed stack were then segmented. the Otsu global thresholding algorithm implemented in ImageJ was applied.

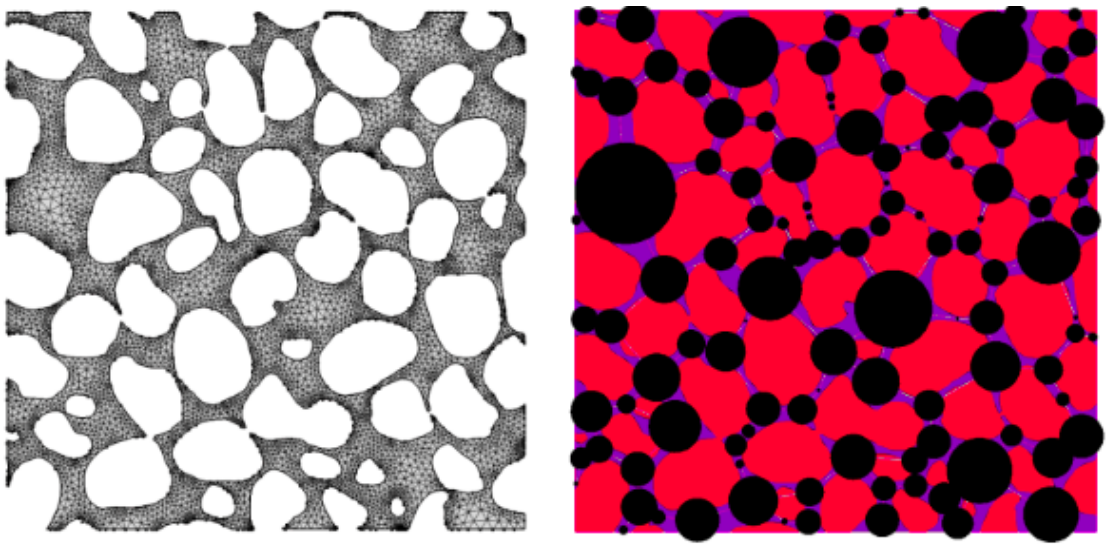
2.3 Network Extraction

The SNOW algorithm in PoreSpy https://porespy.org/ which uses a watershed segmentation algorithm to determine the pore and throat regions based on the distance from the solid region, was used to extract the pore network from the segmented μCT images. To determine markers for the watershed segmentation, the algorithm first applies a Gaussian filter with a sigma of 0.4 to remove spurious peaks from the distance transform. A maximum filter with a structuring element with a radius of 4 wasthen applied to determine the markers which were used as the centers of the pores. In this article, the same parameters are used for all materials as recommended by Gostick 21 to extract spherical pores and cylindrical throats from input images. More details about the algorithm and illustrations of the pore network extraction can be found in the reference. The network extraction is one of the most critical steps in the PNM simulation. According to Gostick, 21 selecting inappropriate parameters for the two filters can result in under- or over-segmentation of the domain. When the structural element is small, many spurious peaks appear making identification of throats challenging. On the other hand, when the structural element is too large, some maxima would be missed. The SNOW algorithm and watersheds, in general, are therefore dependent on image resolution to ensure small voids that are represented by enough pixels to be detected. This problem could be avoided by zooming the image by a factor 2, however, it was not applied in this work to keep the same input conditions.

3 Applications

[103]:
```
from IPython.display import Image
Image(filename='pnm.png')
```

[103]:



(a) FEM mesh (b) Equivalent PNM

4 Predicting absolute permeability

The example explains absolute permeabilty calculations on a cubic network. Note that permeability calcualtion for an extracted network from PoreSpy follows similar steps in assigning phase, algorithm and calculating permeability.

[107]:
```
import os
import numpy as np
import openpnm as op
import scipy.stats as spst
import matplotlib.pyplot as plt
op.visualization.set_mpl_style()
np.random.seed(10)
from matplotlib.pyplot import subplots
%matplotlib inline
np.set_printoptions(precision=5)

current_directory = os.getcwd()
try:
    import plotly
except ModuleNotFoundError:
    !pip install plotly
```

[108]:
```
ws = op.Workspace()
ws.settings.default_solver = "ScipySpsolve"
```

4.0.1 Create a random cubic network

[109]:
```
pn = op.network.Cubic(shape=[15, 15, 15], spacing=1e-6)
pn.add_model_collection(op.models.collections.geometry.spheres_and_cylinders)
pn.regenerate_models()
```

4.1 Geometry

[110]:
```
geo = op.models.collections.geometry.spheres_and_cylinders
```

[111]:
```
geo.keys()
```

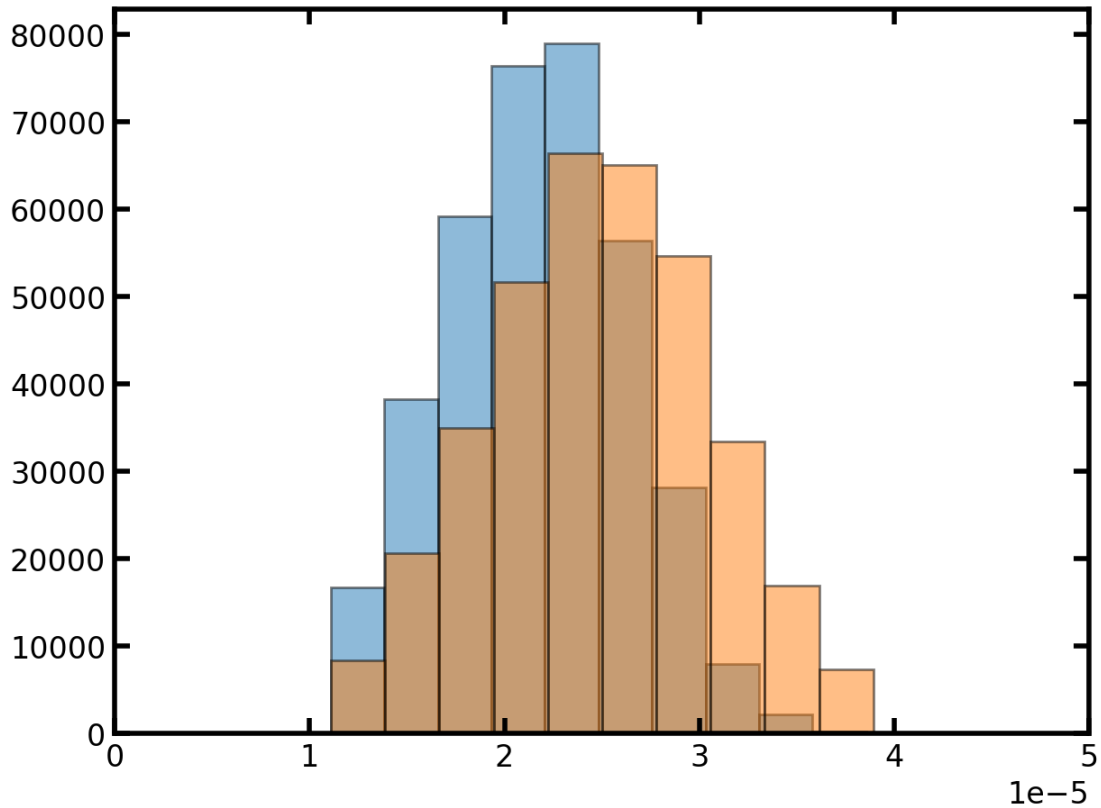
[111]:
```
dict_keys(['pore.seed', 'pore.max_size', 'pore.diameter', 'pore.volume',
'throat.max_size', 'throat.diameter', 'throat.length',
'throat.cross_sectional_area', 'throat.total_volume', 'throat.lens_volume',
'throat.volume', 'throat.diffusive_size_factors',
'throat.hydraulic_size_factors'])
```

[112]:
```
pn.add_model_collection(geo)
pn.regenerate_models()
#print(pn)
```

4.2 Adjusting geometric properties

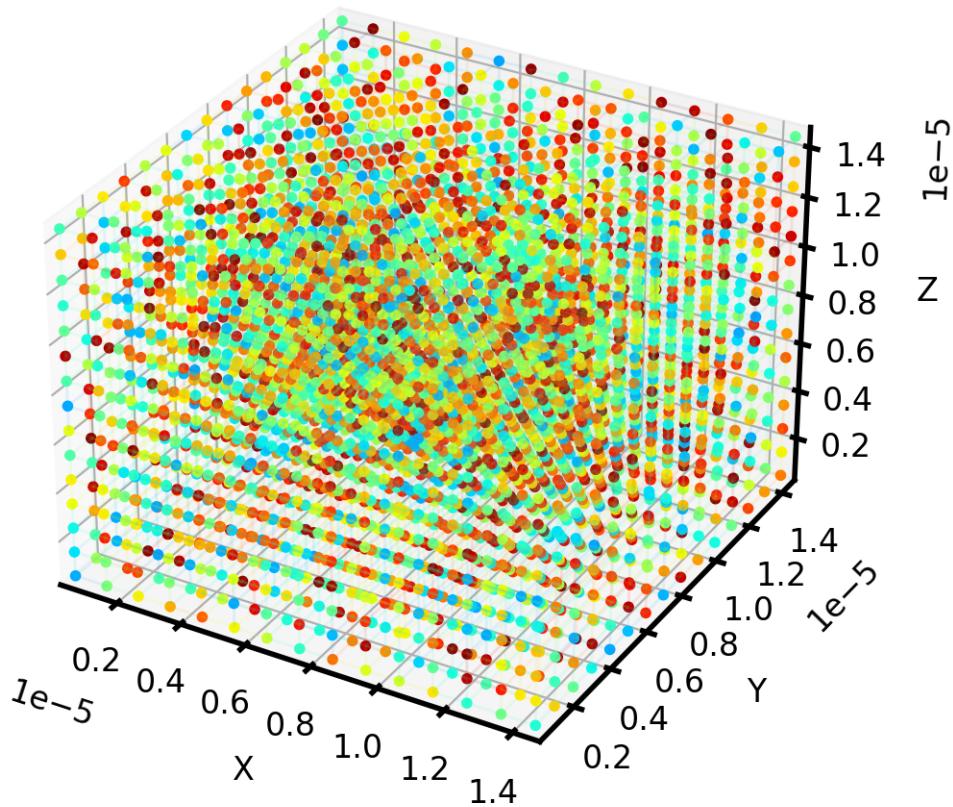
4.2.1 Adding pore and throat sizes from scipy.stats distributions

[113]:
```
import scipy.stats as spst

mean =25
lo, hi = 0.01, 0.99
pn['pore.seed'] = np.random.rand(pn.Np)*(hi - lo) + lo

psd = spst.norm.ppf(loc=mean, scale=6, q=pn['pore.seed'])
pn['pore.diameter'] = psd*1e-6

pn['throat.seed'] = np.amin(pn['pore.seed'][pn.conns], axis=1)

tsd = spst.norm.ppf(loc=mean, scale=6, q=pn['throat.seed'])
pn['throat.diameter'] = tsd*1e-6
plt.hist(pn['throat.diameter'], edgecolor='k', density=True, alpha=0.5)
plt.hist(pn['pore.diameter'], edgecolor='k', density=True, alpha=0.5)
plt.xlim([0, 50e-6]);

pn.regenerate_models()
```



4.2.2 Visualization

[114]:
```
import os
current_directory = os.getcwd()
try:
    import plotly
except ModuleNotFoundError:
    !pip install plotly

ax = op.visualization.plot_coordinates(pn, color_by=pn['pore.diameter'])
ax = op.visualization.plot_connections(pn, ax=ax, color_by=pn['throat.diameter'],␣
↪alpha=0.1)
ax = op.visualization.plot_notebook(pn, node_color=pn['pore.diameter'], node_scale=10)
ax.show()
```



4.2.3 Create phase object

It is assumed that a generic phase flowsthrough the porous medium. As absolute permeability is the porous medium property and not the fluid property, any other fluid with an assigned viscosity value can be used as the phase.

Permeability of the network is then found by applying Stokes Flow algorithm. To simulate the fluid flow algorithm, hydraulic conductance of the conduits must be defined. Here, the model collection is used to assign basic pore-scale models including `generic_hydraulic` to the phase.

[115]:
```
phase = op.phase.Phase(network=pn)
phase['pore.viscosity']=1.0
phase.add_model_collection(op.models.collections.physics.basic)
phase.regenerate_models()
```

4.2.4 Apply Stokes flow

To calculate permeability in x direction, a constant pressure boundary condition is applied on the left and right side of the network. Note that a similar procedure can be followed to find the permeability in y and z directions.

[100]:
```
inlet = pn.pores('left')
outlet = pn.pores('right')
```

```
flow = op.algorithms.StokesFlow(network=pn, phase=phase)
flow.set_value_BC(pores=inlet, values=1)
flow.set_value_BC(pores=outlet, values=0)
flow.run()
phase.update(flow.soln)
```

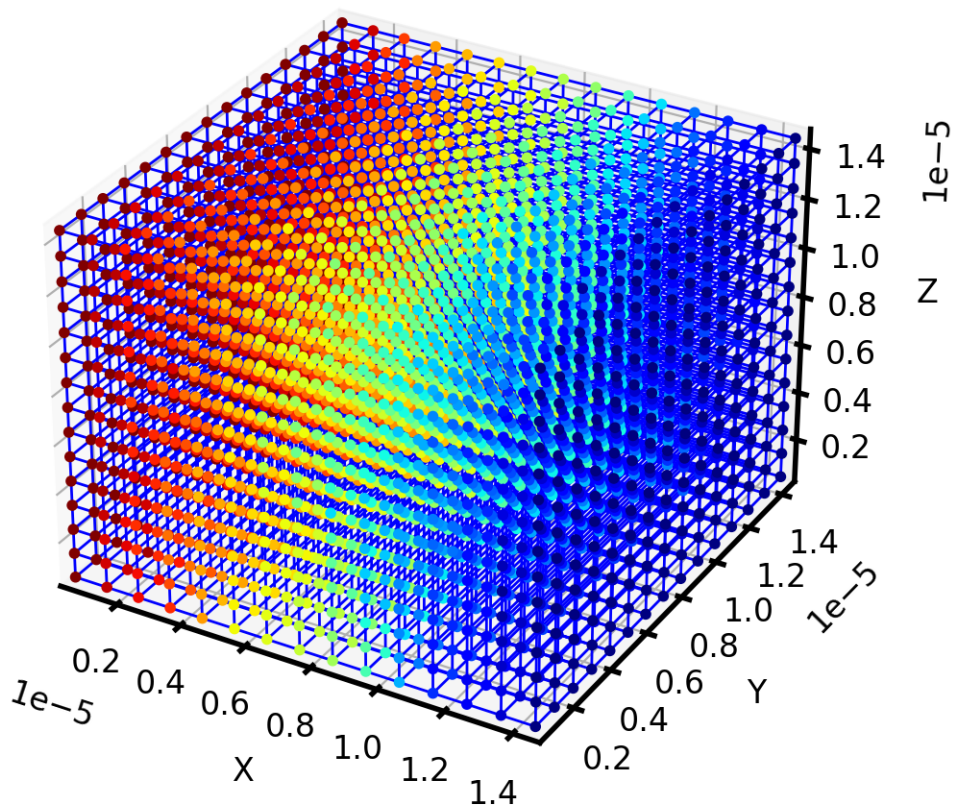
“‘{note} The Solution attribute

`flow.soln` is a `dict` with the `quantity` as the `key` (i.e. `'pore.pressure'`) and the solution as th

“‘

We can visualize the pressure in the network:

[101]:
```
ax = op.visualization.plot_connections(pn)
ax = op.visualization.plot_coordinates(pn, ax=ax, color_by=phase['pore.pressure'])
```



4.2.5 Calculate permeability

Calculate the permeability using Darcy's law:

$$K_{abs} = \frac{Q}{A}\frac{\mu L}{\Delta P}$$

where Q is the inlet flow rate, A is the inlet area, and L is the distance between inlet and outlet. As pressure difference and viscosity were assumed to be 1, we have a simplified equation.

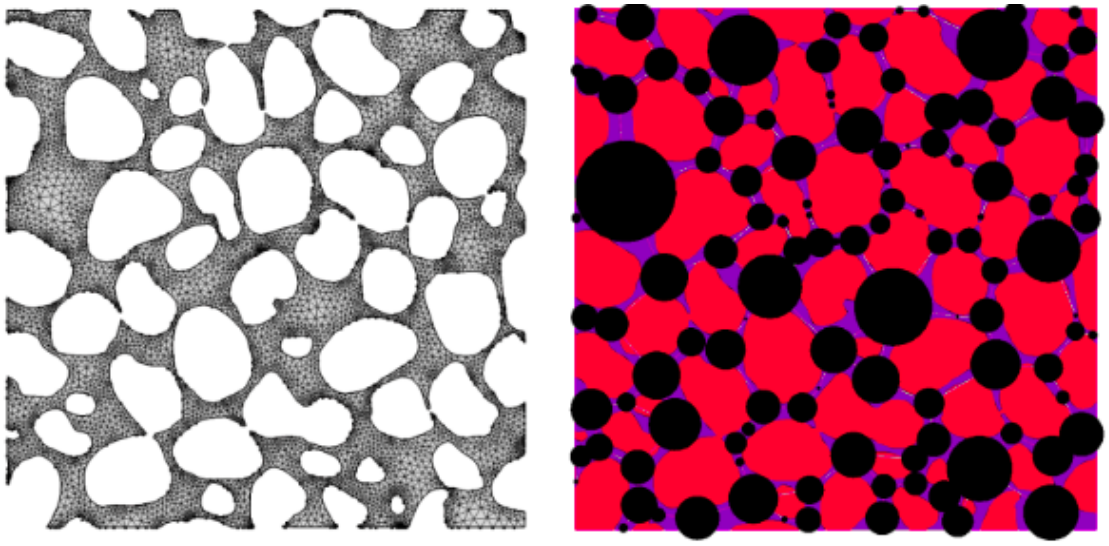
[102]:
```
# NBVAL_IGNORE_OUTPUT
Q = flow.rate(pores=inlet, mode='group')[0]
A = op.topotools.get_domain_area(pn, inlets=inlet, outlets=outlet)
L = op.topotools.get_domain_length(pn, inlets=inlet, outlets=outlet)
# K = Q * L * mu / (A * Delta_P) # mu and Delta_P were assumed to be 1.
K = Q * L / A
print(f'The value of K is: {K/0.98e-12*1000:.2f} mD')
```

```
The value of K is: 0.07 mD
```

[ ]:
```

```

[47]:
```
from IPython.display import Image
Image(filename='pnm.png')
```

[47]:



(a) FEM mesh (b) Equivalent PNM

5 Finding Effective Diffusivity and Tortuosity of a Network

In this example we show how to find the effective diffusivity and torutousity of a cubic network. A similar procedure can be used on more complicated extracted networks from Porespy. The effective diffusivity is defined as the diffusion coefficient multiplied by the porosity and divided by the tortuosity.

$$D_{AB} = D_{eff}\frac{\epsilon}{\tau}$$

The tortuosity is an attempt to correct for the windy path of matter moving through a porous substance. The classical definition is the actual length divided by the direct length. In reality, there are many actual pathways that matter could move through. This means that the actual length should more precisely be thought of as the average path length.

$$\tau = (\frac{L_{actual}}{L_{direct}})^2$$

[51]:
```
import openpnm as op
import matplotlib.pyplot as plt
import numpy as np
import copy

op.visualization.set_mpl_style()
np.random.seed(10)
%matplotlib inline
np.set_printoptions(precision=5)
```

5.1 Network

For this example we generate a `cubic` network.

[53]:
```
shape = [10, 10, 1]
spacing = 1e-5
net = op.network.Cubic(shape=shape, spacing=spacing)
```

5.2 Geometry

Next, we need to add geometry models to the generated network. OpenPNM has collections of geometry models ready to add to the network object. Choosing the right geometry models is important to represent the microstructure of different materials such as Toray090 carbon papers, sand stone, electrospun fibers, etc. For now, we will use a simple collection of geometry models for a spheres_and_cylinders geometry. These geometry models assume pores are spheres and throats are cylinders.

[54]:
```
geo = op.models.collections.geometry.spheres_and_cylinders
net.add_model_collection(geo, domain='all')
net.regenerate_models()
```

A collection of geometry models should now be added to the network object. Make sure to `regenerate_models` after adding any new models to the network. To view the models that have been added to the network object print the network as follows. Notice how `throat.diameter` and `pore.volume` as well as other shape factor models have been added to the network.

[55]:
```
print(net)
```

```
net : <openpnm.network.Cubic at 0x7fef5f555e50>
--------------------------------------------------------------------------------
  #  Properties                                                     Valid Values
--------------------------------------------------------------------------------
  2  pore.coords                                                       100 / 100
  3  throat.conns                                                      180 / 180
  4  pore.coordination_number                                          100 / 100
  5  pore.max_size                                                     100 / 100
  6  throat.spacing                                                    180 / 180
  7  pore.seed                                                         100 / 100
  8  pore.diameter                                                     100 / 100
  9  throat.max_size                                                   180 / 180
 10  throat.diameter                                                   180 / 180
 11  throat.cross_sectional_area                                       180 / 180
 12  throat.hydraulic_size_factors                                     180 / 180
 13  throat.diffusive_size_factors                                     180 / 180
 14  throat.lens_volume                                                180 / 180
 15  throat.length                                                     180 / 180
 16  throat.total_volume                                               180 / 180
 17  throat.volume                                                     180 / 180
 18  pore.volume                                                       100 / 100
--------------------------------------------------------------------------------
  #  Labels                                                     Assigned Locations
--------------------------------------------------------------------------------
  2  pore.surface                                                            36
  3  throat.surface                                                          36
```

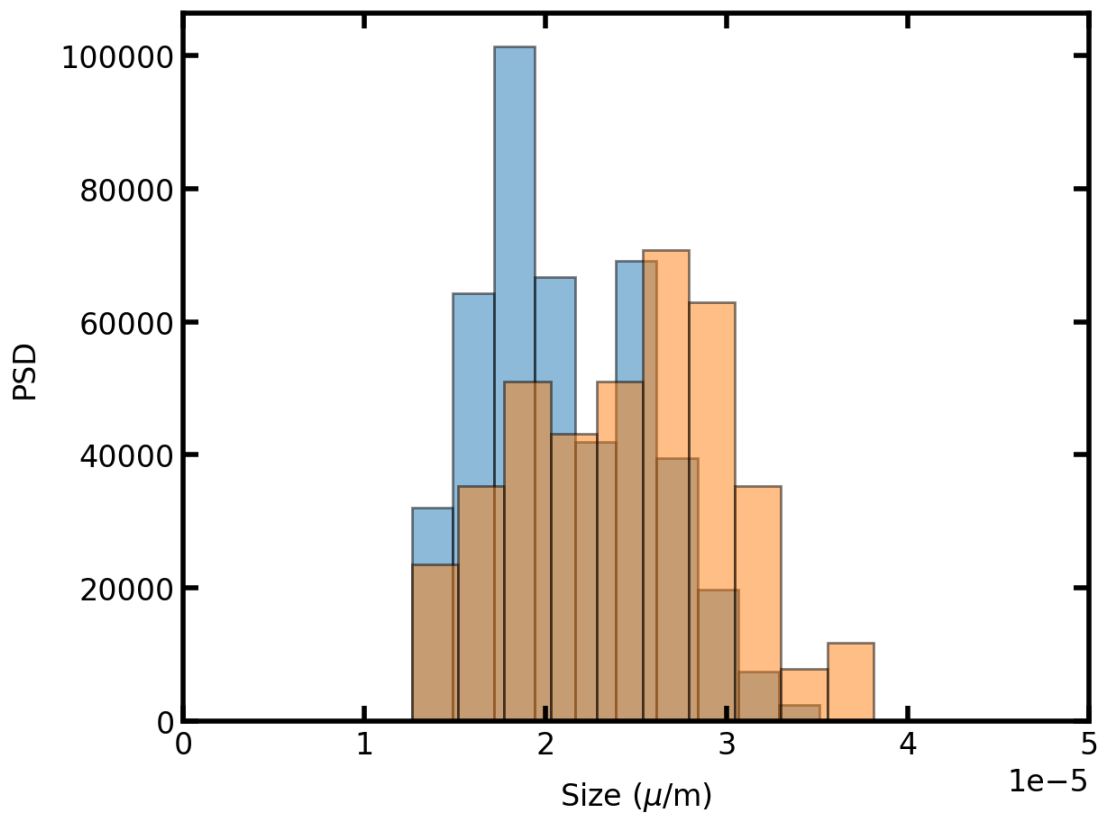
```
  4  pore.left                                                          10
  5  pore.right                                                         10
  6  pore.front                                                         10
  7  pore.back                                                          10
--------------------------------------------------------------------------
```

5.3 Adjusting geometric properties

5.3.1 Adding pore and throat sizes from scipy.stats distributions

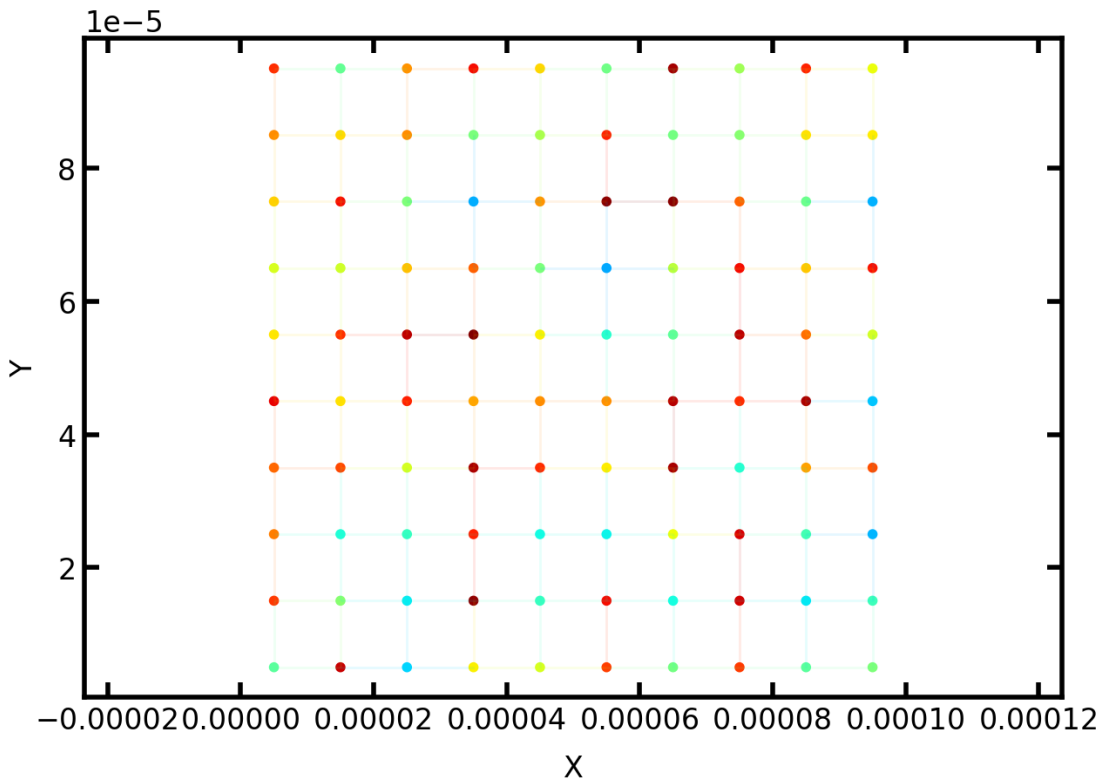
Setting the `lo` and `hi` values of the the seeds prevents values that lie far out in the tail of a di

[59]:
```
import scipy.stats as spst

mean =25
lo, hi = 0.01, 0.99
net['pore.seed'] = np.random.rand(net.Np)*(hi - lo) + lo

psd = spst.norm.ppf(loc=mean, scale=6, q=net['pore.seed'])
net['pore.diameter'] = psd*1e-6

net['throat.seed'] = np.amin(net['pore.seed'][net.conns], axis=1)

tsd = spst.norm.ppf(loc=mean, scale=6, q=net['throat.seed'])
net['throat.diameter'] = tsd*1e-6
plt.hist(net['throat.diameter'], edgecolor='k', density=True, alpha=0.5)
plt.hist(net['pore.diameter'], edgecolor='k', density=True, alpha=0.5)
plt.xlim([0, 50e-6]);
plt.xlabel('Size ($\mu$/m)')
plt.ylabel('PSD')
#plt.legend('Pores','Throats')
net.regenerate_models()
pn = copy.deepcopy(net)
```



5.3.2 Visualiztion

[60]:
```
import os
current_directory = os.getcwd()
try:
    import plotly
except ModuleNotFoundError:
    !pip install plotly

ax = op.visualization.plot_coordinates(pn, color_by=pn['pore.diameter'])
ax = op.visualization.plot_connections(pn, ax=ax, color_by=pn['throat.diameter'],␣
↪alpha=0.1)
ax = op.visualization.plot_notebook(pn, node_color=pn['pore.diameter'], node_scale=10)
ax.show()
```



5.4 Phase

Now we add a phase object to our simulation. A phase object contains thermophysical information about the working fluid in the simulation. For this simulation, we will use air as our working fluid. `OpenPNM` has other built-in phase objects to choose from.

[61]:
```
air = op.phase.Air(network=net)
```

5.5 Physics

Next we need to add physics models to the phase object. Here we add the `basic` collection of physics models. This includes a `diffusive_conductance` model which we will need for our fickian diffusion simulation. It is easy to remove any physics models you don't need such as `throat.entry_pressure`.

[62]:
```
phys = op.models.collections.physics.basic
#del phys['throat.entry_pressure']
air.add_model_collection(phys)
air.regenerate_models()
```

```
[12:54:05] WARNING  throat.entry_pressure was not run since the following property is␣
↪missing:       ]8;id=758993;file:///home/justin/miniconda3/envs/openpnm3/lib/python3.9/
↪site-packages/openpnm/core/_models.py\_models.py]8;;\:]8;id=56105;file:///home/justin/
↪miniconda3/envs/openpnm3/lib/python3.9/site-packages/openpnm/core/_models.py#480\480]8;
↪;\
                    'throat.surface_tension'
```

5.6 Performing Fickian Diffusion

Now that everything's set up, it's time to perform our Fickian diffusion simulation. For this purpose, we need to add the `FickianDiffusion` algorithm to our simulation. Here's how we do it:

[63]:
```
fd = op.algorithms.FickianDiffusion(network=net, phase=air)
```

Note that network and phase are required parameters for pretty much every algorithm we add, since we need to specify on which network and for which phase we want to run the algorithm.

[64]:
```
inlet = net.pores('left')
outlet = net.pores('right')
C_in, C_out = [10, 5]
fd.set_value_BC(pores=inlet, values=C_in)
fd.set_value_BC(pores=outlet, values=C_out)
```

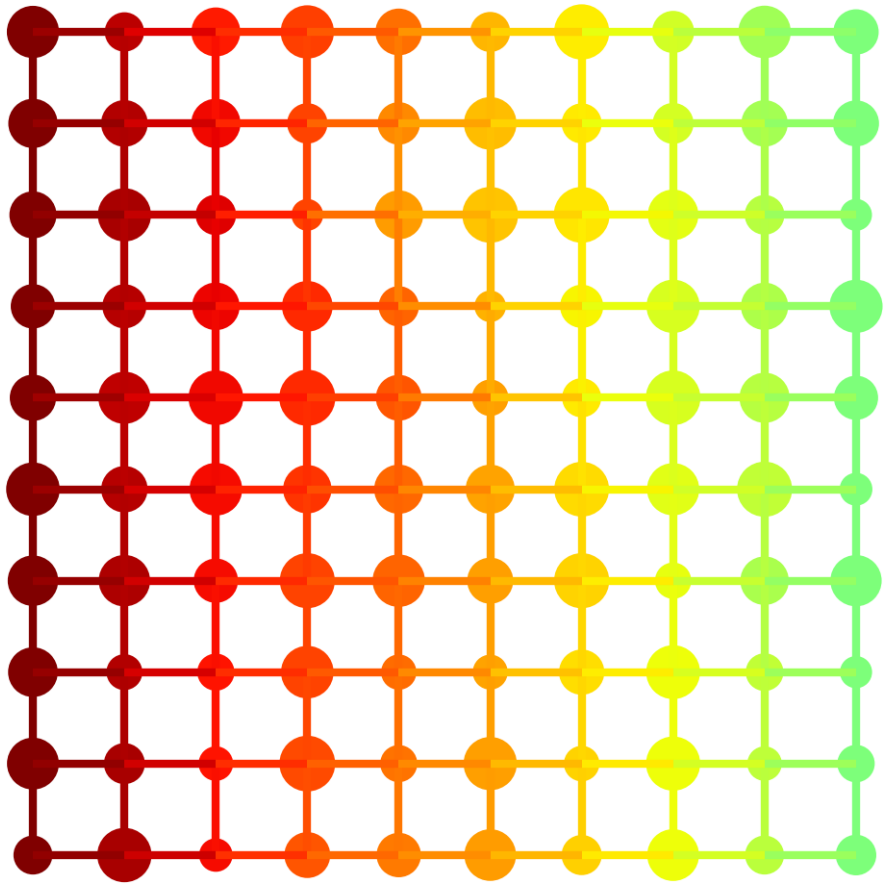
Now, it's time to run the algorithm. This is done by calling the run method attached to the algorithm object.

[65]:
```
fd.run();
```

5.7 Visualize the Results

Now that we know the quantity for which `FickianDiffusion` was solved, let's take a look at the results. We use the `plot_coordinates` and `plot_connections` available under visualization. Throat data is interpolated from pore data.

[66]:
```
pc = fd['pore.concentration']
tc = fd.interpolate_data(propname='throat.concentration')
d = net['pore.diameter']
fig, ax = plt.subplots(figsize=[5, 5])
op.visualization.plot_coordinates(network=net, color_by=pc, size_by=d, markersize=400,␣
 ↪ax=ax)
op.visualization.plot_connections(network=net, color_by=tc, linewidth=3, ax=ax)
_ = plt.axis('off')
```



5.8 Calculate the Effective Diffusivity

We can determine the effective diffusivity of the network by solving Fick's law as shown below.

$$D_{eff} = \frac{N_A L}{A \Delta C}$$

To determine the molar flow rate we can use the `rate` method attached to the algorithm object. The molar flow rate going through the boundary pores at the inlet is calculated. This gives the molar flow rate in units of moles per second.

[67]:
```
rate_inlet = fd.rate(pores=inlet)[0]
print(f'Molar flow rate: {rate_inlet:.5e} mol/s')
```

```
Molar flow rate: 4.83276e-11 mol/s
```

Now let's calculate the effective diffusivity

[68]:
```
A = (shape[1] * shape[2])*(spacing**2)
L = shape[0]*spacing
D_eff = rate_inlet * L / (A * (C_in - C_out))
print("{0:.6E}".format(D_eff))
```

```
9.665517E-07
```

5.9 Calculate the Tortuosity

Tortuosity is most easily determined as a fitting factor to transport data. We can use the effective diffusivity measured from the fickian diffusion simulation and the known diffusivity of air to solve for the tortuosity.

$$\tau = \varepsilon \frac{D_{AB}}{D_{eff}}$$

But first the porosity of the network must be calculated. Here is how we can calculate the porosity. For more information on how to measure the porosity please refer to the porosity notebook.

[69]:
```
V_p = net['pore.volume'].sum()
V_t = net['throat.volume'].sum()
V_bulk = np.prod(shape)*(spacing**3)
e = (V_p + V_t) / V_bulk
print('The porosity is: ', "{0:.6E}".format(e))
```

```
The porosity is:  9.677553E-02
```

[35]:
```
D_AB = air['pore.diffusivity'][0]
tau = e * D_AB / D_eff
print('The tortuosity is:', "{0:.6E}".format(tau))
```

```
The tortuosity is: 2.222980E+00
```

[ ]:
```

```

6 Obtaining Relative Diffusivity Curves

This is an example of calculating relative diffusivity in OpenPNM. The term **relative** indicates the existence of another phase in the medium. The concept of relative property (permeability) in multiphase transport was applied in relative permeability example for a flow problem. In this example the transport mechanism is defined as diffusion.

[3]:
```
import numpy as np
import openpnm as op
import numpy as np
import matplotlib.pyplot as plt
op.visualization.set_mpl_style()
np.random.seed(10)
%matplotlib inline
np.set_printoptions(precision=5)
ws = op.Workspace()
ws.settings.default_solver = "ScipySpsolve"
```

6.0.1 Create network and phases

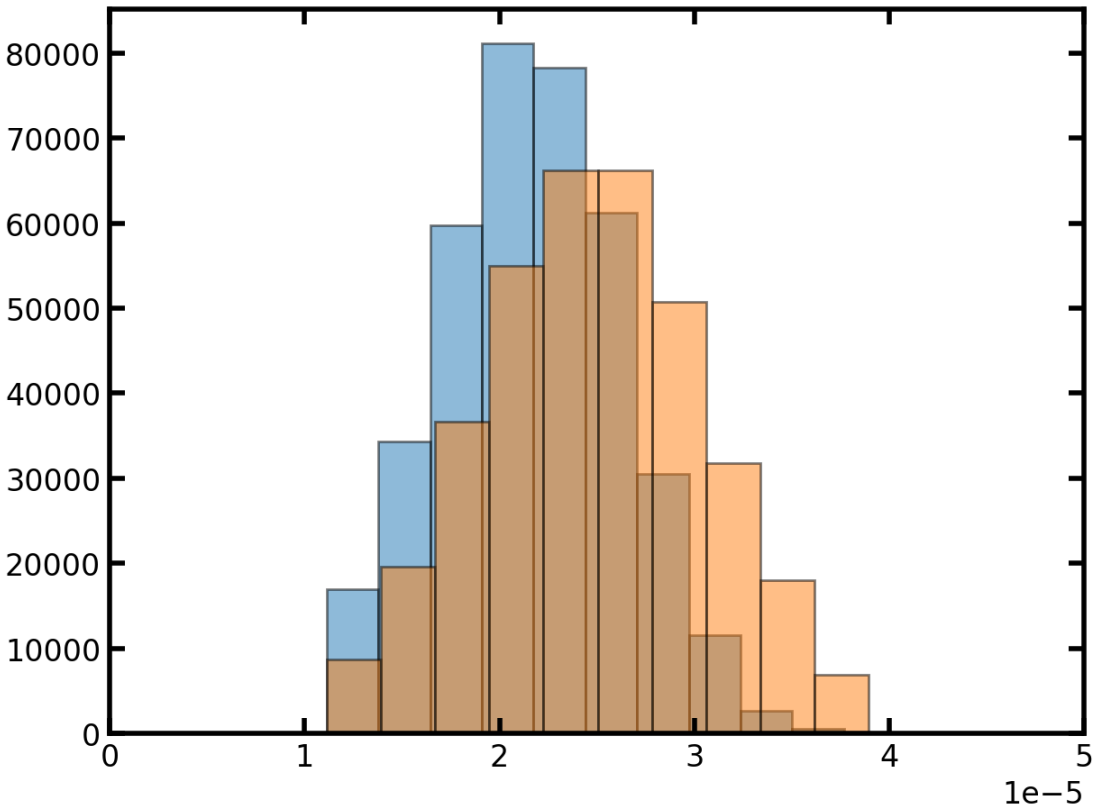
First, we create a network and assign phases and properties in a similar way that we used to do for the other examples. Note that `op.models.collections.phase` is added to assign fluid properties such as diffusivity, surface tension, etc, and `op.models.collections.physics` is added to assign pore-scale models such as entry pressure, conduit conductance. Surface tension info will be used in Invasion percolation in the next step. Here, we assumed a user-defined value for air surface tension and contact angle, as they are not included in `collections.phase.air`. Similarly, a user-defined value for water diffusivity was assumed, as it is not included in `collections.phase.water`.

[4]:
```
pn = op.network.Cubic(shape=[15, 15, 15], spacing=1e-6)
pn.add_model_collection(op.models.collections.geometry.spheres_and_cylinders)
pn.regenerate_models()
```

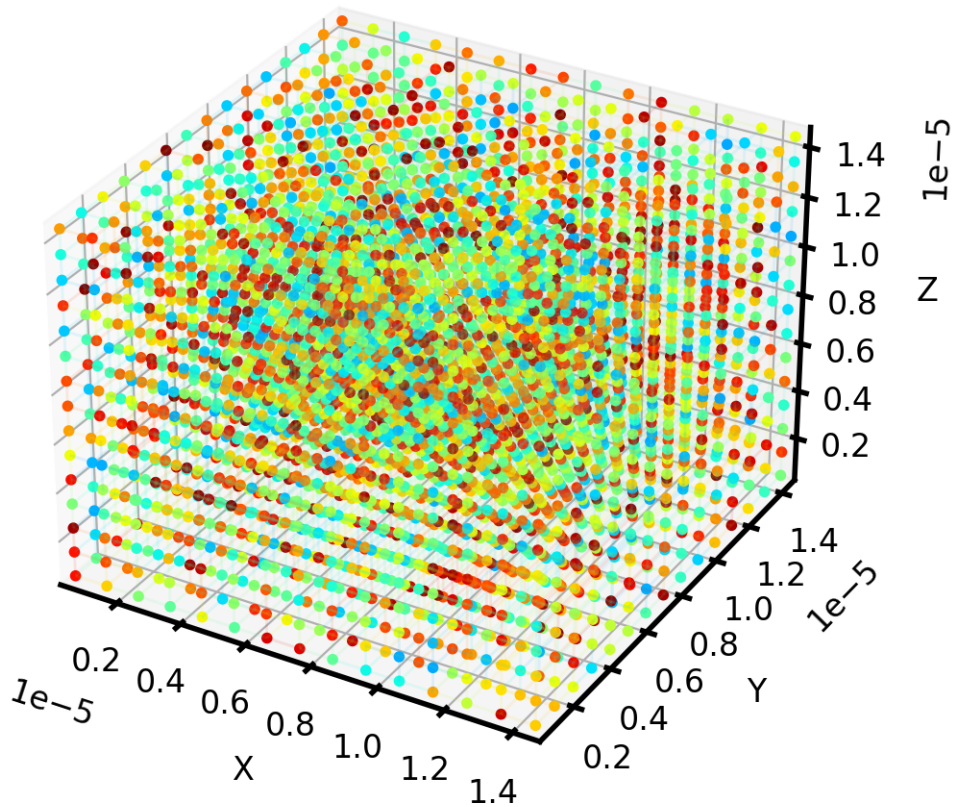
6.1 Adjusting geometric properties

6.1.1 Adding pore and throat sizes from `scipy.stats` distributions

[6]:
```
import scipy.stats as spst

mean =25
lo, hi = 0.01, 0.99
pn['pore.seed'] = np.random.rand(pn.Np)*(hi - lo) + lo

psd = spst.norm.ppf(loc=mean, scale=6, q=pn['pore.seed'])
pn['pore.diameter'] = psd*1e-6

pn['throat.seed'] = np.amin(pn['pore.seed'][pn.conns], axis=1)

tsd = spst.norm.ppf(loc=mean, scale=6, q=pn['throat.seed'])
pn['throat.diameter'] = tsd*1e-6
plt.hist(pn['throat.diameter'], edgecolor='k', density=True, alpha=0.5)
plt.hist(pn['pore.diameter'], edgecolor='k', density=True, alpha=0.5)
plt.xlim([0, 50e-6]);

pn.regenerate_models()
```



[16]:
```
import os
current_directory = os.getcwd()
try:
    import plotly
except ModuleNotFoundError:
    !pip install plotly

ax = op.visualization.plot_coordinates(pn, color_by=pn['pore.diameter'])
ax = op.visualization.plot_connections(pn, ax=ax, color_by=pn['throat.diameter'],␣
↪alpha=0.1)
ax = op.visualization.plot_notebook(pn, node_color=pn['pore.diameter'], node_scale=10)
ax.show()
```



6.2 Phases set up

[7]:
```
air = op.phase.Air(network=pn,name='air')
air['pore.surface_tension'] = 0.072
air['pore.contact_angle'] = 180.0
air.add_model_collection(op.models.collections.phase.air)
air.add_model_collection(op.models.collections.physics.basic)
air.regenerate_models()
water = op.phase.Water(network=pn,name='water')
water['pore.diffusivity'] = 1e-9
water.add_model_collection(op.models.collections.phase.water)
water.add_model_collection(op.models.collections.physics.basic)
water.regenerate_models()
```

6.2.1 Apply ordinary percolation

To simulate a partially saturated material we first invade some non-wetting phase. This will be accomplished using the `InvasonPercolation` or `OrdinaryPercolation` algorithm. In both algorithms the invading phase invades the network based on the capillary pressure of the throats in the network. These two algorithms have difference in their percolation algorithm and results. As Invasion percolation was implemented in the relative permeability example, here we choose ordinary percolation to demonstrate how we can use its results. It is assumed that the invading phase is water:

[8]:
```
drn = op.algorithms.Drainage(network=pn, phase=water)
drn.set_inlet_BC(pn.pores('left'))
drn.run()
```

```
Performing drainage simulation:   0%|          | 0/25 [00:00<?, ?it/s]
```

In ordinary percolation, occupancies and saturation can be extracted from the results for a capillary pressure point. For example:

[9]:
```
water['pore.occupancy'] = drn['pore.invasion_pressure'] < 10000
water['throat.occupancy'] = drn['throat.invasion_pressure'] < 10000
air['pore.occupancy'] = 1 - water['pore.occupancy']
air['throat.occupancy'] = 1 - water['throat.occupancy']
```

6.2.2 Saturation and rate functions

To update calculate the saturation based on occupancy from each pressure point, we created a costum function:

[10]:
```
def sat_update(network, nwp, wp):
    r"""
        Calculates the saturation of each phase using occupancy information from␣
 ↪ordinary percolation.
        Parameters
        ----------
        network: network
        nwp : phase
            non-wetting phase
        wp : phase
            wetting phase
    """
    pore_mask = nwp["pore.occupancy"] == 1
    throat_mask = nwp["throat.occupancy"] == 1
    sat_p = np.sum(network['pore.volume'][pore_mask])
    sat_t = np.sum(network['throat.volume'][throat_mask])
    sat1 = sat_p + sat_t
    bulk = network['pore.volume'].sum() + network['throat.volume'].sum()
    sat = sat1/bulk
    return sat
```

Similar to the relative permeability example, the geometrical parameters in relative diffusivity can cancell out. Therefore, the remaining fraction includes diffusion rates as follows:

$D_r = \frac{D_{eff-multiphase}}{D_{eff-singlephase}} = \frac{Rate_{multiphase}}{Rate_{singlephase}}$

Where $Rate_{multiphase}$ is the inlet diffusion rate of the phase (water/air) for a Fickian diffusion algorithm with `multiphase conduit_conductance` assigned as conduit diffusive conductance model to account for the existence of the other phase. Whereas $Rate_{singlephase}$ is the inlet diffusion rate of the phase (water/air) for a Fickian diffusion algorithm with `generic_diffusive_conductance` assigned as diffusive conductance model for a single phase flow.

Note that $D_{eff-singlephase}$ is a property of the porous material, not the fluid. However, here to cancel out diffusivity in the D_r fraction, the same phase must be used to calculate $Rate_{singlephase}$. Therefore, we only need to find the rate values for calcuating the relative permeability. We define a function `Rate_calc`, which simulates a Fickian diffusion given the diffusive conductance keyword and returns the rate diffusion at inlet pores.

[11]:
```
def Rate_calc(network, phase, inlet, outlet, conductance):
        phase.regenerate_models()
        Fd = op.algorithms.FickianDiffusion(network=network, phase=phase)
        Fd.settings._update({'conductance' : conductance})
        Fd.set_value_BC(pores=inlet, values=1)
        Fd.set_value_BC(pores=outlet, values=0)
        Fd.run()
        val = np.abs(Fd.rate(pores=inlet, mode='group'))
        return val
```

In this example we are focusing on finding the relative diffusivities in x direction. A similar procedure can be applied on y and z directions. Let's define inlet and outlet pores:

[12]:
```
diff_in = pn.pores('left')
diff_out = pn.pores('right')
```

6.2.3 Define multiphase conductance

Similar to the relative permeability example, multiphase conductance models are assigned to account for the presence of the other phase:

[13]:
```
model_mp_cond = op.models.physics.multiphase.conduit_conductance
air.add_model(model=model_mp_cond, propname='throat.conduit_diffusive_conductance',
              throat_conductance='throat.diffusive_conductance', mode='medium',␣
 ↪regen_mode='deferred')
water.add_model(model=model_mp_cond, propname='throat.conduit_diffusive_conductance',
              throat_conductance='throat.diffusive_conductance', mode='medium',␣
 ↪regen_mode='deferred')
```

6.2.4 Calculate relative diffusivities

Now that we have defined the required models and functions, we can find the relative diffusivity of phases over a range of saturation. The saturation points are found at a range of invasion pressure:

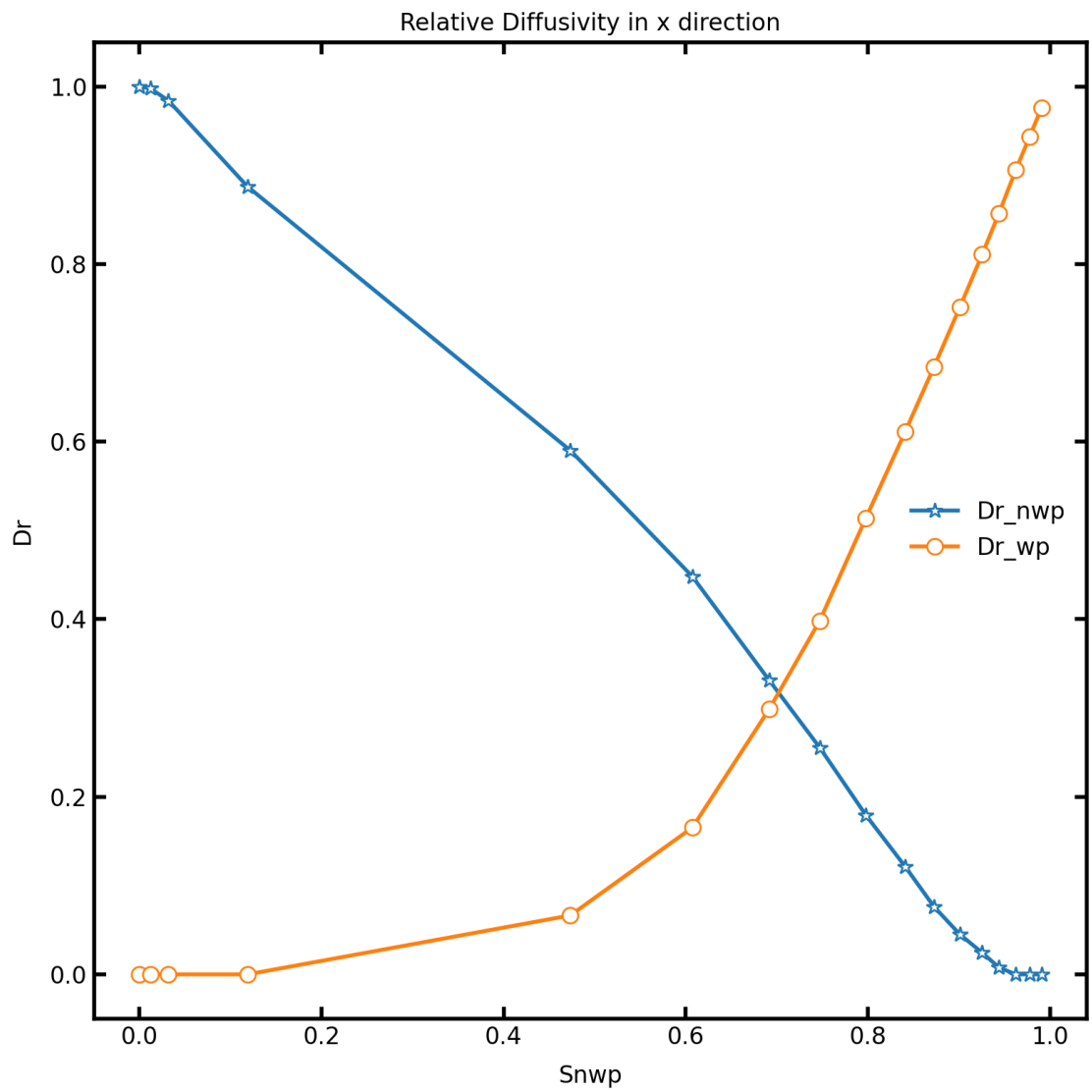
[14]:
```
Snwparr = []
reldiff_nwp = []
reldiff_wp = []
for Pc in np.unique(drn['pore.invasion_pressure']):
    water['pore.occupancy'] = drn['pore.invasion_pressure'] < Pc
    water['throat.occupancy'] = drn['throat.invasion_pressure'] < Pc
    air['pore.occupancy'] = 1 - water['pore.occupancy']
    air['throat.occupancy'] = 1 - water['throat.occupancy']
    air.regenerate_models()
    water.regenerate_models()
    sat_point = sat_update(pn, water, air)
    Snwparr.append(sat_point)
    Rate_single_nwp = Rate_calc(pn, air, diff_in, diff_out, conductance = 'throat.
 ↪diffusive_conductance')
    Rate_single_wp = Rate_calc(pn, water, diff_in, diff_out, conductance = 'throat.
 ↪diffusive_conductance')
    Rate_mult_nwp = Rate_calc(pn, air, diff_in, diff_out, conductance = 'throat.
 ↪conduit_diffusive_conductance')
    Rate_mult_wp = Rate_calc(pn, water, diff_in, diff_out, conductance = 'throat.
 ↪conduit_diffusive_conductance')
```

```
    reldiff_nwp.append(Rate_mult_nwp/Rate_single_nwp)
    reldiff_wp.append(Rate_mult_wp/Rate_single_wp)
```

Plotting the curves:

[15]:
```
plt.figure(figsize=[8,8])
plt.plot(Snwparr, reldiff_nwp, '*-', label='Dr_nwp')
plt.plot(Snwparr, reldiff_wp, 'o-', label='Dr_wp')
plt.xlabel('Snwp')
plt.ylabel('Dr')
plt.title('Relative Diffusivity in x direction')
plt.legend()
```

[15]: <matplotlib.legend.Legend at 0x7fc8d46e5bb0>

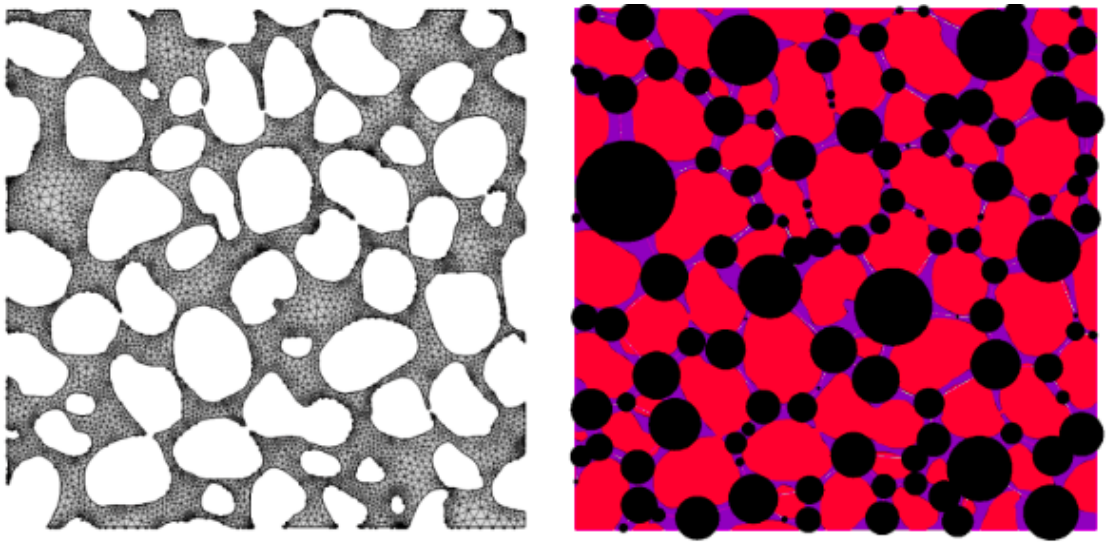


[ ]:

```

```

[26]:
```
from IPython.display import Image
Image(filename='pnm.png')
```

[26]:



(a) FEM mesh (b) Equivalent PNM

7 Predicting relative permeability

This is an example of calculating relative permeability in OpenPNM.

[14]:
```
import numpy as np
import openpnm as op
import matplotlib.pyplot as plt
op.visualization.set_mpl_style()
np.random.seed(10)
%matplotlib inline
np.set_printoptions(precision=5)
ws = op.Workspace()
ws.settings.default_solver = "ScipySpsolve"
```

7.0.1 Create network and phases

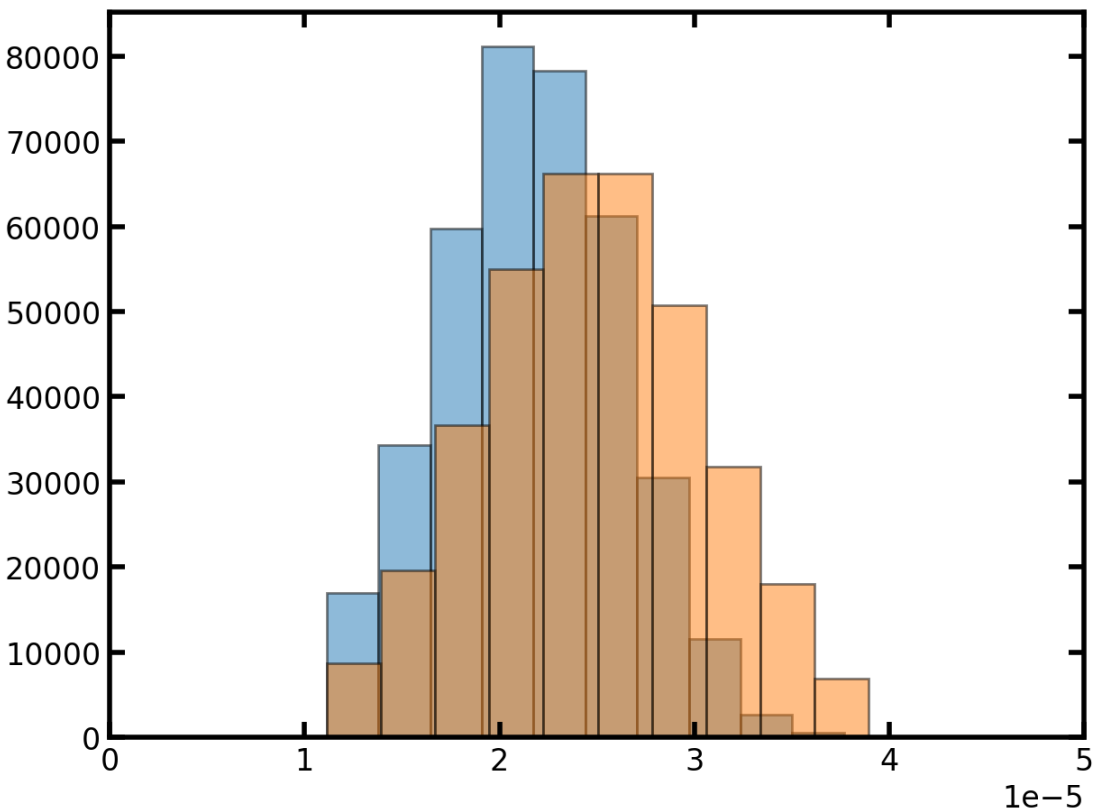
First, we create a network and assign phases and properties in a similar way that we used to do for the other examples. Note that `op.models.collections.phase` is added to assign fluid properties such as viscosity, surface tension, etc, and `op.models.collections.physics` is added to assign pore-scale models such as entry pressure, conduit conductance. Surface tension info will be used in Invasion percolation in the next step. Here, we assumed a user-defined value for air surface tension and contact angle, as they are not included in `collections.phase.air`.

[15]:
```
pn = op.network.Cubic(shape=[15, 15, 15], spacing=1e-6)
pn.add_model_collection(op.models.collections.geometry.spheres_and_cylinders)
pn.regenerate_models()
```

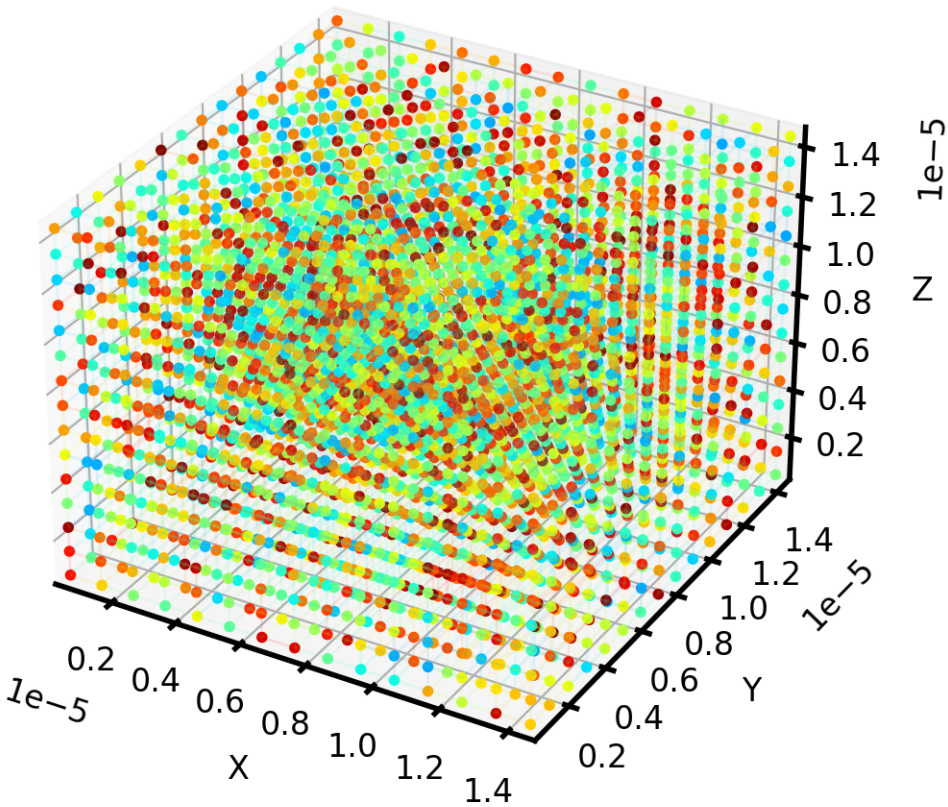
7.1 Adjusting geometric properties

7.1.1 Adding pore and throat sizes from scipy.stats distributions

[16]:
```
import scipy.stats as spst

mean =25
lo, hi = 0.01, 0.99
pn['pore.seed'] = np.random.rand(pn.Np)*(hi - lo) + lo

psd = spst.norm.ppf(loc=mean, scale=6, q=pn['pore.seed'])
pn['pore.diameter'] = psd*1e-6

pn['throat.seed'] = np.amin(pn['pore.seed'][pn.conns], axis=1)

tsd = spst.norm.ppf(loc=mean, scale=6, q=pn['throat.seed'])
pn['throat.diameter'] = tsd*1e-6
plt.hist(pn['throat.diameter'], edgecolor='k', density=True, alpha=0.5)
plt.hist(pn['pore.diameter'], edgecolor='k', density=True, alpha=0.5)
plt.xlim([0, 50e-6]);

pn.regenerate_models()
```



[27]:
```
import os
current_directory = os.getcwd()
try:
    import plotly
except ModuleNotFoundError:
    !pip install plotly

ax = op.visualization.plot_coordinates(pn, color_by=pn['pore.diameter'])
ax = op.visualization.plot_connections(pn, ax=ax, color_by=pn['throat.diameter'],␣
↪alpha=0.1)
ax = op.visualization.plot_notebook(pn, node_color=pn['pore.diameter'], node_scale=10)
ax.show()
```



7.1.2 Phase set up

[17]:
```
air = op.phase.Air(network=pn,name='air')
air['pore.surface_tension'] = 0.072
air['pore.contact_angle'] = 180.0
air.add_model_collection(op.models.collections.phase.air)
air.add_model_collection(op.models.collections.physics.basic)
air.regenerate_models()
```

```
water = op.phase.Water(network=pn,name='water')
water.add_model_collection(op.models.collections.phase.water)
water.add_model_collection(op.models.collections.physics.basic)
water.regenerate_models()
```

```
[12:28:37] WARNING  throat.diffusive_conductance was not run since the following␣
↪property is         ]8;id=239712;file:///home/justin/miniconda3/envs/openpnm3/lib/
↪python3.9/site-packages/openpnm/core/_models.py\_models.py]8;;\:]8;id=694099;file:///
↪home/justin/miniconda3/envs/openpnm3/lib/python3.9/site-packages/openpnm/core/_models.
↪py#480\480]8;;\
                    missing: 'throat.diffusivity'
```

7.1.3 Apply invasion percolation

The next step is to apply an invasion percolation algorithm to obtain the invasion sequence. Asumming a drainage process, the air(invading/non-wetting phase) will be invading the medium. The IP algorithm can be impplemented through a user-defined inlet face. Here, we use the left surface pores in the x direction. By updating the air phase, the invasion sequence can then be found using the phase occupancy which is a property of the phase.

[18]:
```
ip = op.algorithms.InvasionPercolation(network=pn, phase=air)
Finlets_init = pn.pores('left')
Finlets=([Finlets_init[x] for x in range(0, len(Finlets_init), 2)])
ip.set_inlet_BC(pores=Finlets)
ip.run()
```

7.1.4 Occupancy and permeability functions

To update the occupancy of phases at each saturation point, we created a costum function:

[19]:
```
def sat_occ_update(network, nwp, wp, ip, i):
    r"""
        Calculates the saturation of each phase using the invasion
        sequence from either invasion percolation.
        Parameters
        ----------
        network: network
        nwp : phase
            non-wetting phase
        wp : phase
            wetting phase
        ip : IP
            invasion percolation (ran before calling this function)
        i: int
            The invasion_sequence limit for masking pores/throats that
            have already been invaded within this limit range. The
            saturation is found by adding the volume of pores and thorats
            that meet this sequence limit divided by the bulk volume.
    """
    pore_mask = ip['pore.invasion_sequence'] < i
    throat_mask = ip['throat.invasion_sequence'] < i
    sat_p = np.sum(network['pore.volume'][pore_mask])
    sat_t = np.sum(network['throat.volume'][throat_mask])
```

```
    sat1 = sat_p + sat_t
    bulk = network['pore.volume'].sum() + network['throat.volume'].sum()
    sat = sat1/bulk
    nwp['pore.occupancy'] = pore_mask
    nwp['throat.occupancy'] = throat_mask
    wp['throat.occupancy'] = 1-throat_mask
    wp['pore.occupancy'] = 1-pore_mask
    return sat
```

As relative permeability is the ratio of effective over absolute permeability, given the same boundary condition and viscosity, the geometrical parameters in K_{rphase} will cancel out. The remaining fraction includes flow rates as follows:

$K_{rphase} = \frac{K_{ephase}}{K_{abs}} = \frac{Q_{e-phase}}{Q_{abs-phase}}$

Where Q_{ephase} is the inlet flow rate of the phase (water/air) for a Stokes flow algorithm with `multiphase conduit_conductance` assigned as conduit hydraulic conductance model to account for the existence of the other phase. Whereas Q_{abs} is the inlet flow rate of the phase (water/air) for a Stokes flow algorithm with `generic_hydraulic_conductance` assigned as hydraulic conductance model for a single phase flow.

Note that K_{abs} is a property of the porous material, not the fluid. However, here to cancel out viscosity in the K_{rphase} fraction, the same phase must be used to calculate $Q_{abs-phase}$. Therefore, naming `abs-phase` was used here to indicate the flow rate from a single phase flow algorithm for the same phase that relative permeability is being calculated. Therefore, we only need to find the rate values for calcuating the relative permeability.

We define a function `Rate_calc`, which simulates a stokes flow given the hydraulic conductance keyword and returns the rate of inlet fluid flow.

[20]:
```
def Rate_calc(network, phase, inlet, outlet, conductance):
    phase.regenerate_models()
    St_p = op.algorithms.StokesFlow(network=network, phase=phase)
    St_p.settings._update({'conductance' : conductance})
    St_p.set_value_BC(pores=inlet, values=1)
    St_p.set_value_BC(pores=outlet, values=0)
    St_p.run()
    val = np.abs(St_p.rate(pores=inlet, mode='group'))
    return val
```

In this example we are focusing on finding the relativep permeabilities in x direction. A similar procedure can be applied on y and z directions. Let's define inlet and outlet pores:

[21]:
```
flow_in = pn.pores('left')
flow_out = pn.pores('right')
```

7.1.5 Define multiphase conductance

Assigning multiphase conductance models to the phases. The multiphase conduit conductance accounts for the presence of the other phase in effective permeability calculations. For more details, plesase see the model's code in OpenPNM package.

[22]:
```
model_mp_cond = op.models.physics.multiphase.conduit_conductance
air.add_model(model=model_mp_cond, propname='throat.conduit_hydraulic_conductance',
              throat_conductance='throat.hydraulic_conductance', mode='medium',␣
 ↪regen_mode='deferred')
water.add_model(model=model_mp_cond, propname='throat.conduit_hydraulic_conductance',
```

```
              throat_conductance='throat.hydraulic_conductance', mode='medium',␣
↪regen_mode='deferred')
```

7.1.6 Calculate relative permeabilities

Now that we have defined the required models and functions, we can find the relative permeability of phases over a range of saturation. We selected 10 saturation points. A higher value can be used to obtain a better approximation of the curve.

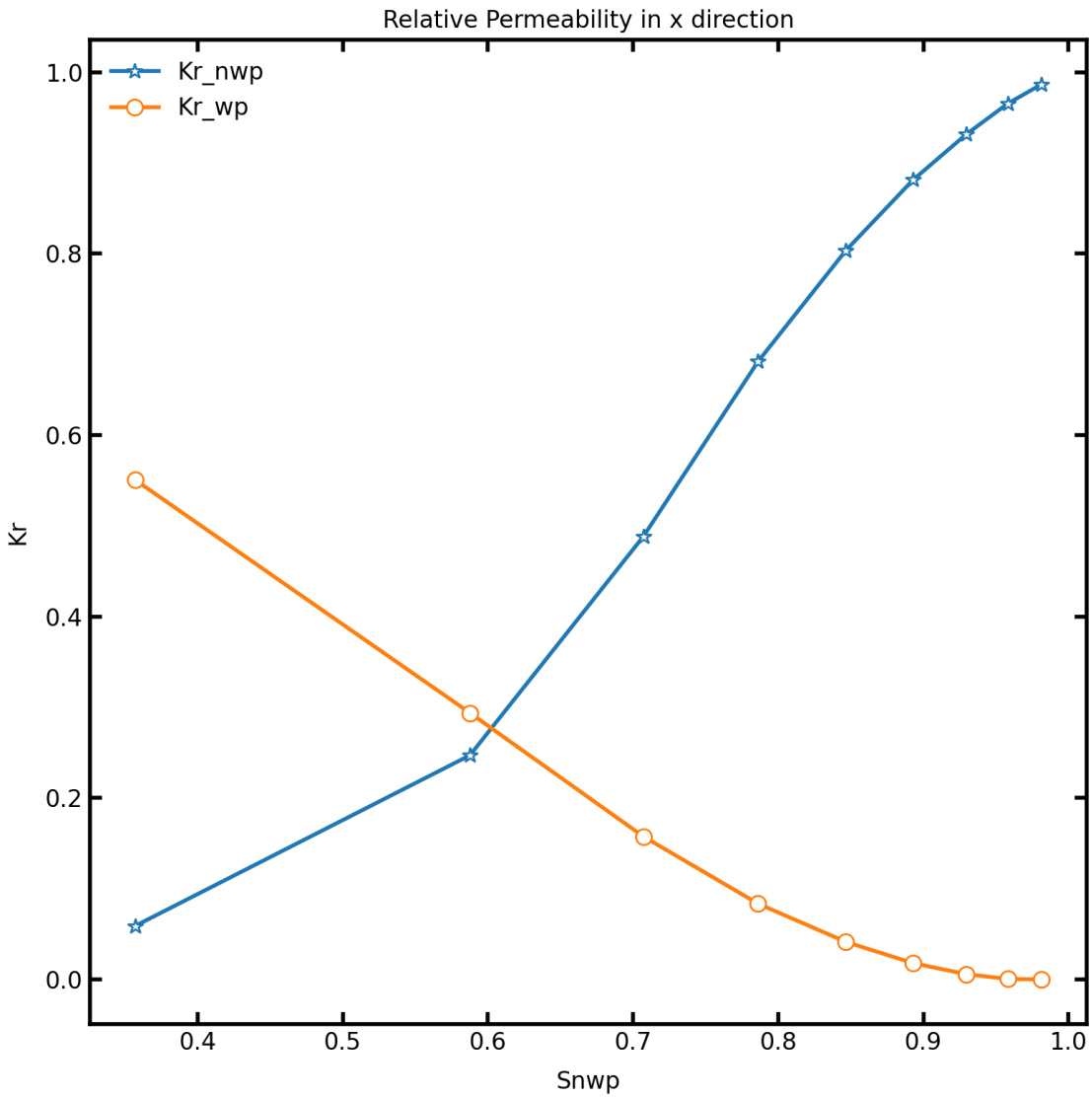
[23]:
```
Snwp_num=10
flow_in = pn.pores('left')
flow_out = pn.pores('right')
max_seq = np.max([np.max(ip['pore.invasion_sequence']),
          np.max(ip['throat.invasion_sequence'])])
start = max_seq//Snwp_num
stop = max_seq
step = max_seq//Snwp_num
Snwparr = []
relperm_nwp = []
relperm_wp = []
```

[24]:
```
for i in range(start, stop, step):
    air.regenerate_models();
    water.regenerate_models();
    sat = sat_occ_update(network=pn, nwp=air, wp=water, ip=ip, i=i)
    Snwparr.append(sat)
    Rate_abs_nwp = Rate_calc(pn, air, flow_in, flow_out, conductance = 'throat.
↪hydraulic_conductance')
    Rate_abs_wp = Rate_calc(pn, water, flow_in, flow_out, conductance = 'throat.
↪hydraulic_conductance')
    Rate_enwp = Rate_calc(pn, air, flow_in, flow_out, conductance = 'throat.
↪conduit_hydraulic_conductance')
    Rate_ewp = Rate_calc(pn, water, flow_in, flow_out, conductance = 'throat.
↪conduit_hydraulic_conductance')
    relperm_nwp.append(Rate_enwp/Rate_abs_nwp)
    relperm_wp.append(Rate_ewp/Rate_abs_wp)
```

Let's plot the relative permeability curves:

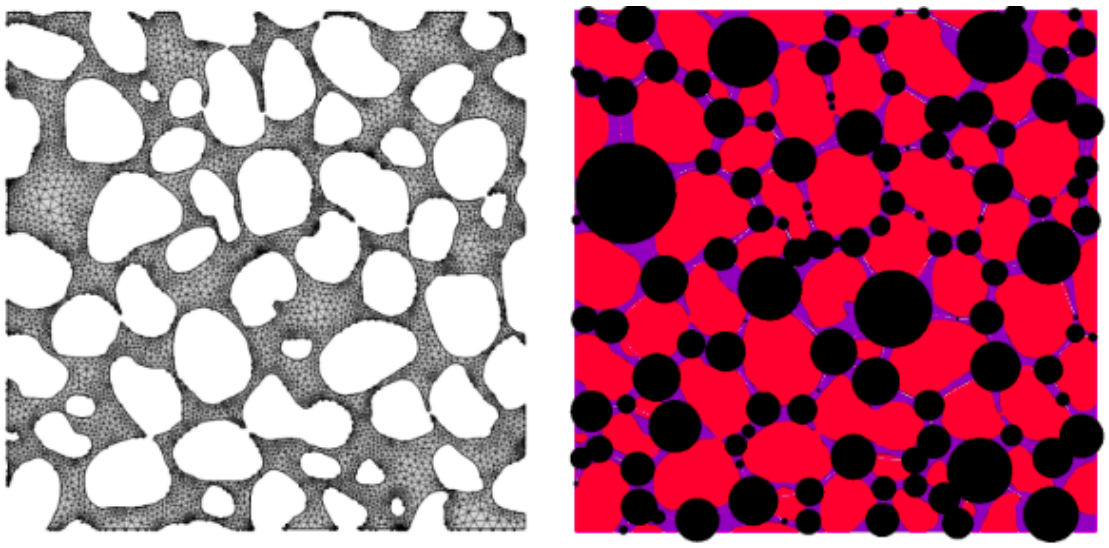
[25]:
```
plt.figure(figsize=[8,8])
plt.plot(Snwparr, relperm_nwp, '*-', label='Kr_nwp')
plt.plot(Snwparr, relperm_wp, 'o-', label='Kr_wp')
plt.xlabel('Snwp')
plt.ylabel('Kr')
plt.title('Relative Permeability in x direction')
plt.legend()
```

[25]:
```
<matplotlib.legend.Legend at 0x7f6836162a00>
```


Relative Permeability in x direction
Kr_nwp
Kr_wp
Kr
Snwp
0.0
0.2
0.4
0.6
0.8
1.0
0.4
0.5
0.6
0.7
0.8
0.9
1.0

[2]:
```
from IPython.display import Image
Image(filename='pnm.png')
```

[2]:



(a) FEM mesh (b) Equivalent PNM

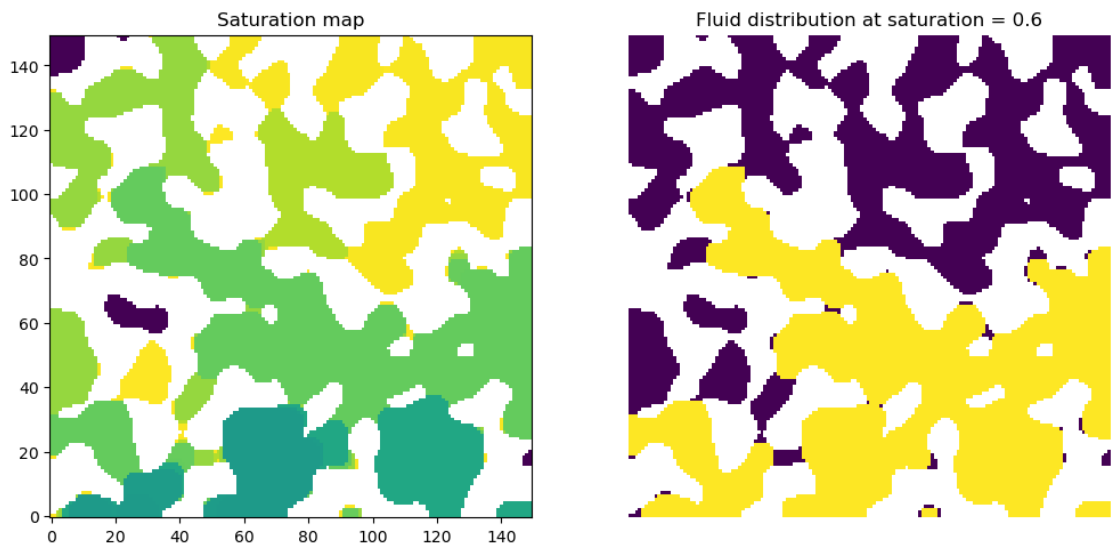
[3]:
```
import os
import imageio
import scipy as sp
import numpy as np
import openpnm as op
import openpnm.models.geometry as gmods
from openpnm.models import physics as pm
import porespy as ps
import matplotlib.pyplot as plt
np.set_printoptions(precision=4)
import openpnm.models as mods
np.random.seed(10)
%matplotlib inline
```

[4]:
```
np.random.seed(1)
im = ps.generators.blobs(shape=[150, 150], porosity=0.6, blobiness=1)
inlets = np.zeros_like(im)
inlets[0, :,] = True
inv = ps.simulations.drainage(im=im, inlets=inlets, voxel_size=1, g=0)
```

```
  0%|          | 0/25 [00:00<?, ?it/s]

  0%|          | 0/12 [00:00<?, ?it/s]
```

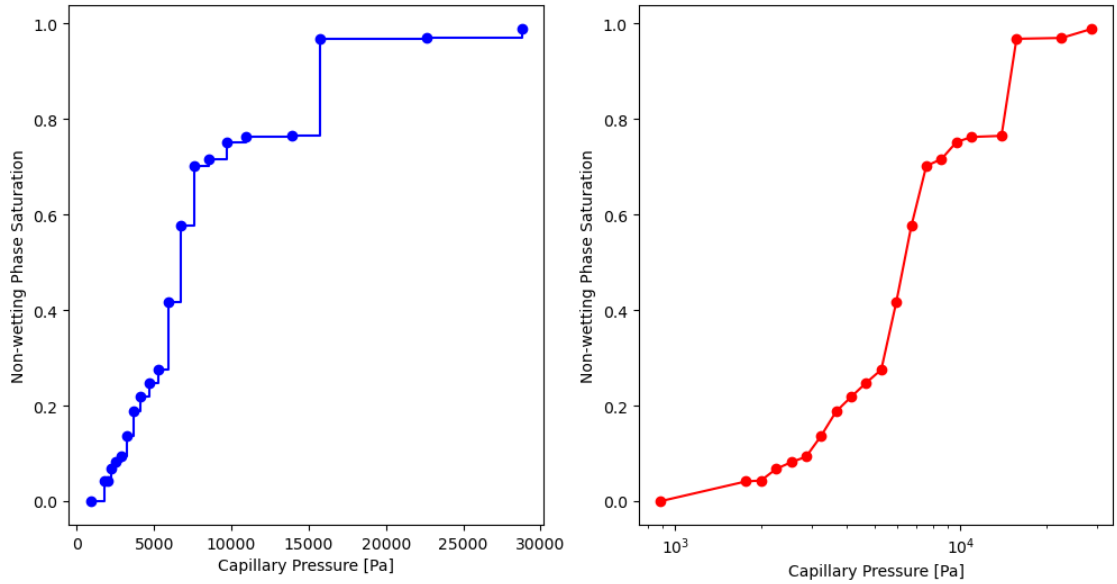
[5]:
```
fig, ax = plt.subplots(1, 2, figsize=[12, 6])
ax[0].imshow(inv.im_satn/im, interpolation='none', origin='lower')
#ax[0].axis(:)
ax[0].set_title('Saturation map')
ax[1].imshow((inv.im_satn < 0.6)*(inv.im_satn > 0)/im, interpolation='none',␣
 ↪origin='lower')
```

```
ax[1].axis(False)
ax[1].set_title('Fluid distribution at saturation = 0.6');
```



[7]:
```
drainage = ps.filters.porosimetry(im)

data = ps.metrics.pc_curve(im=im,sizes=drainage,voxel_size=5.e-6)

fig, ax = plt.subplots(1, 2, figsize=[12, 6])
ax[0].step(data.pc, data.snwp, 'b-o', where='post')
ax[0].set_xlabel('Capillary Pressure [Pa]')
ax[0].set_ylabel('Non-wetting Phase Saturation')
ax[1].semilogx(data.pc, data.snwp, 'r-o')
ax[1].set_xlabel('Capillary Pressure [Pa]')
ax[1].set_ylabel('Non-wetting Phase Saturation');
```

```
0%|          | 0/25 [00:00<?, ?it/s]

0%|          | 0/21 [00:00<?, ?it/s]
```

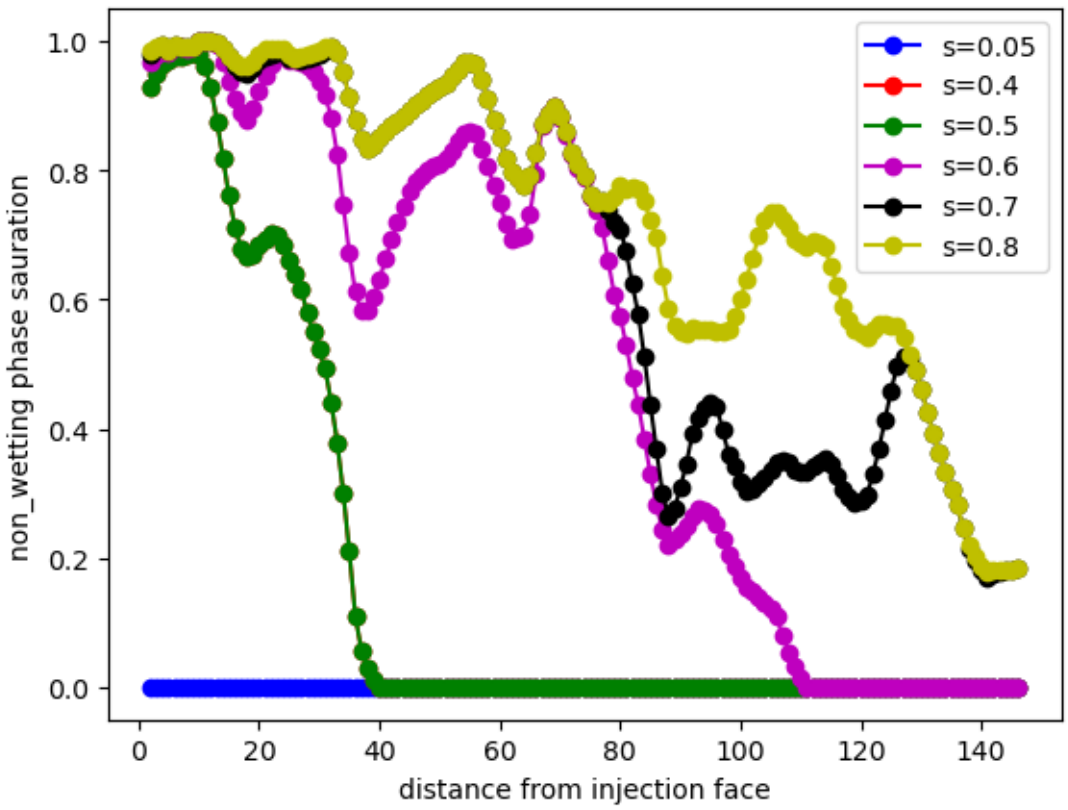


[8]:
```
inlets = np.zeros_like(im)
inlets[0,:] = True
inv = ps.simulations.drainage(im=im, inlets=inlets, voxel_size=1e-5,g=0)
```

```
  0%|          | 0/25 [00:00<?, ?it/s]

  0%|          | 0/12 [00:00<?, ?it/s]
```

[9]:
```
sp = 5
color_list = ['b-o','r-o','g-o','m-o','k-o','y-o','r--^']
jj=0
for ii in [0.05,0.4,0.5,0.6,0.7,0.8]:
    s = ps.metrics.satn_profile(satn=inv.im_satn,s=ii,span=sp,mode='slide')
    plt.plot(s.position,s.saturation, color_list[jj],label= f"s={ii}")
    jj=jj+1
plt.legend()
plt.xlabel("distance from injection face")
plt.ylabel("non_wetting phase sauration")
```

[9]: Text(0, 0.5, 'non_wetting phase sauration')

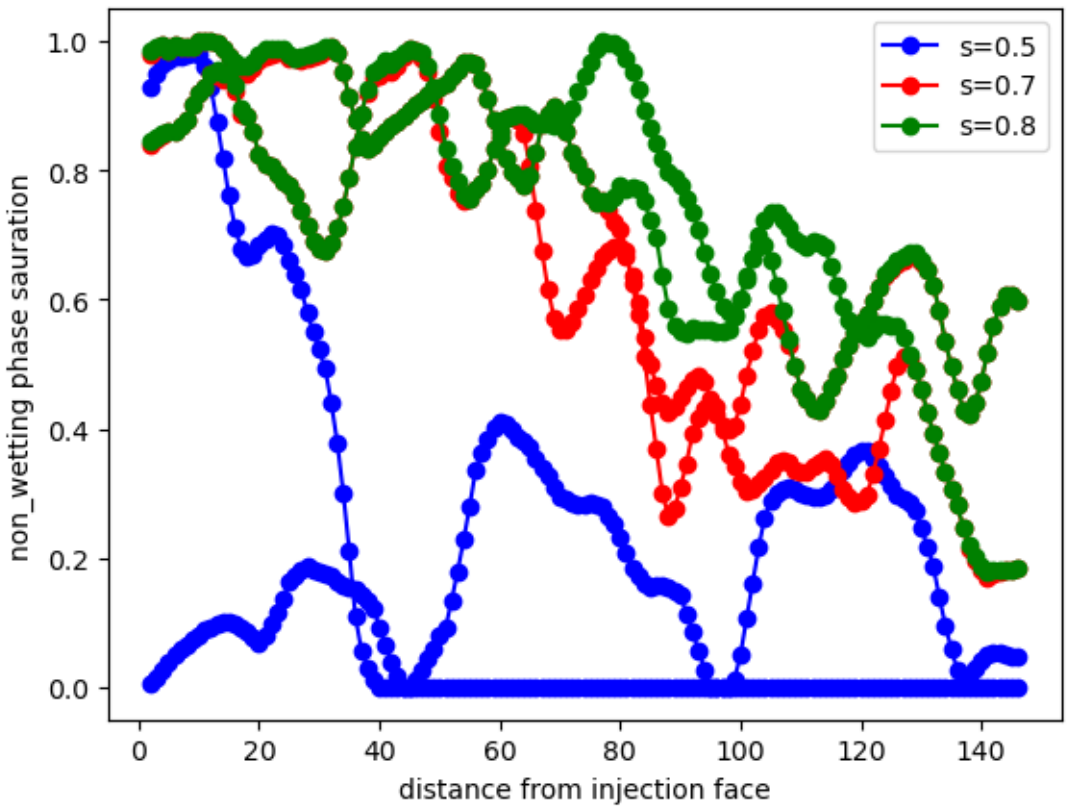


7.2 Comparison of flow in multiple directions

7.2.1 Is the flow isotropic or anisotropic ?

[10]:
```
sp= 5
color_list = ['b-o','r-o','g-o','m-o','k-o','y-o','r--^']
jj=0
for ii in [0.5,0.7,0.8]:
    # s = ps.metrics.satn_profile(satn=inv.im_satn, s=0.6, span=s)
     s1 = ps.metrics.satn_profile(satn=inv.im_satn,s=ii,span=sp,mode='slide',axis=0)
     plt.plot(s1.position,s1.saturation, color_list[jj],label= f"s={ii}")
     s2 = ps.metrics.satn_profile(satn=inv.im_satn,s=ii,span=sp,mode='slide',axis=1)
     plt.plot(s2.position,s2.saturation, color_list[jj])
     jj=jj+1
plt.legend()
plt.xlabel("distance from injection face")
plt.ylabel("non_wetting phase sauration")
```

[10]: Text(0, 0.5, 'non_wetting phase sauration')

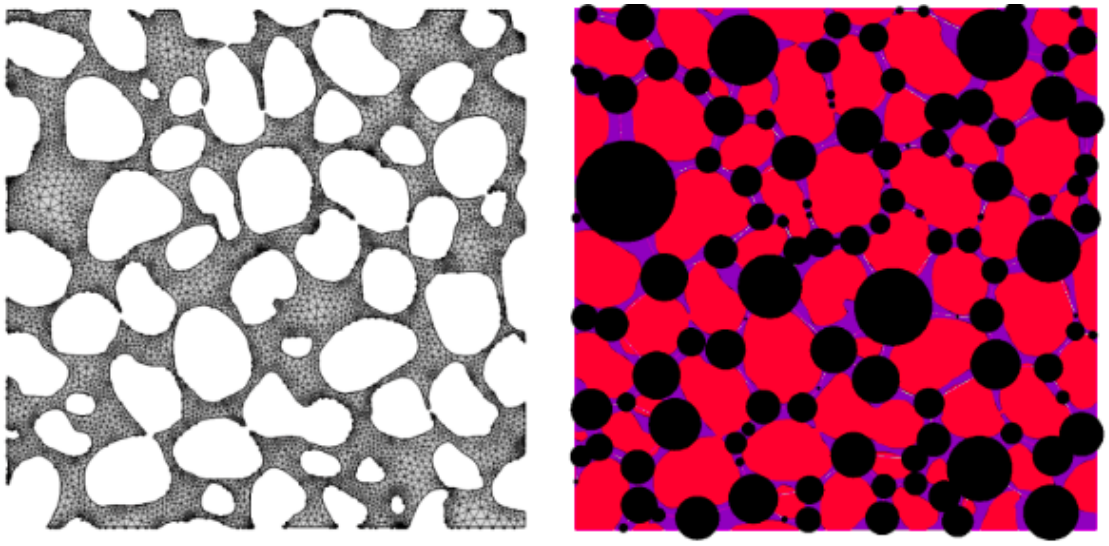

s=0.5
s=0.7
s=0.8
non_wetting phase sauration
distance from injection face
0.0
0.2
0.4
0.6
0.8
1.0
0
20
40
60
80
100
120
140

```
[ ]:
```

7.3 Tomography image based simulation

[36]:
```
from IPython.display import Image
Image(filename='pnm.png')
```

[36]:



(a) FEM mesh (b) Equivalent PNM

[37]:
```
import os
import imageio
import scipy as sp
import numpy as np
import openpnm as op
import openpnm.models.geometry as gmods
from openpnm.models import physics as pm
import porespy as ps
import matplotlib.pyplot as plt
np.set_printoptions(precision=4)
import openpnm.models as mods
np.random.seed(10)
%matplotlib inline
```

7.4 Invasion simulation

7.5 Computing relative permeability and diffusivity from real tomography images

Liquid water distribution

To validate the distribution of liquid water in the domain, a one-to-one comparison was done between the experimental and numerical reconstructions. Therefore, in order to generate the images in Fig. Water distribution, the labels of saturated pores in PNM simulations from the extracted network and the same labels in t watershed segmented reconstruction were marked to obtain saturated reconstructions from PNM. To compare the water distribution within the domain, the numerical water distributions are extracted at similar saturation values to the experimental reconstruction.

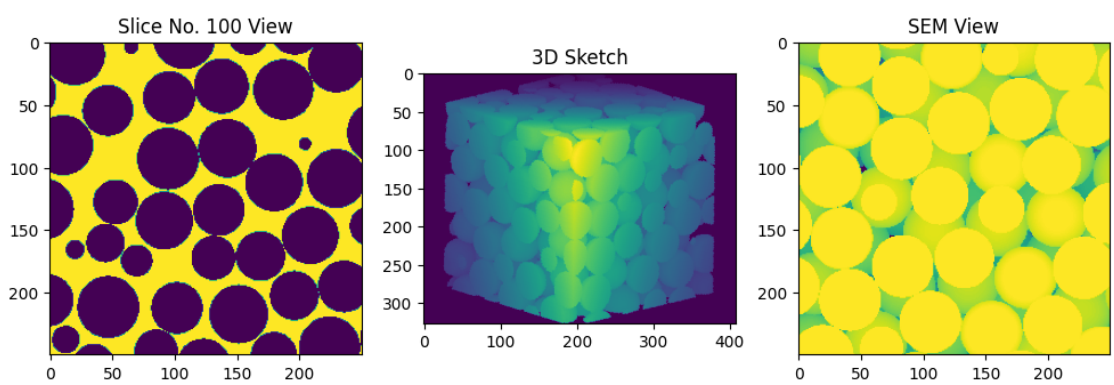
Figure... shows a comparison of the experimental and simulated liquid water distribution in the x-y plane at 50% and 80% through-plane depth from the inlet face. The distributions from PNM in Fig. 6c and 6f show larger clusters similar to the experimental water distributions

PNM segments the entire void part into discrete pore regions and each region is fully invaded in a single step. This resulted in the larger water clusters which are visually closer to the experimental reconstruction.

In order to assess the accuracy of the simulated intrusion results, the local saturation profiles in the through-plane direction are shown in Fig... Local saturation profiles were obtained by the ratio of the number of pixels assigned to the water phase to the sum of pixels in the void and water phase at a given distance from the bottom face

[38]:
```
path = '/home/justin/Documents/redaction/data_tomography/'
file_format = '.tif'
#file_name = 'Berea_sandstone'
file_name = 'beadpack'
file = file_name + file_format
fetch_file = os.path.join(path, file)
im = imageio.mimread(fetch_file)
im_org =im.copy()
im = ~np.array(im, dtype=bool)[:250, :250, :250] # Make image a bit smaller
```

[ ]:
```
# NBVAL_IGNORE_OUTPUT
fig, ax = plt.subplots(1, 3, figsize=(12,5))
ax[0].imshow(im[:, :, 100]);
ax[1].imshow(ps.visualization.show_3D(im));
ax[2].imshow(ps.visualization.sem(im));
ax[0].set_title("Slice No. 100 View");
ax[1].set_title("3D Sketch");
ax[2].set_title("SEM View");
```

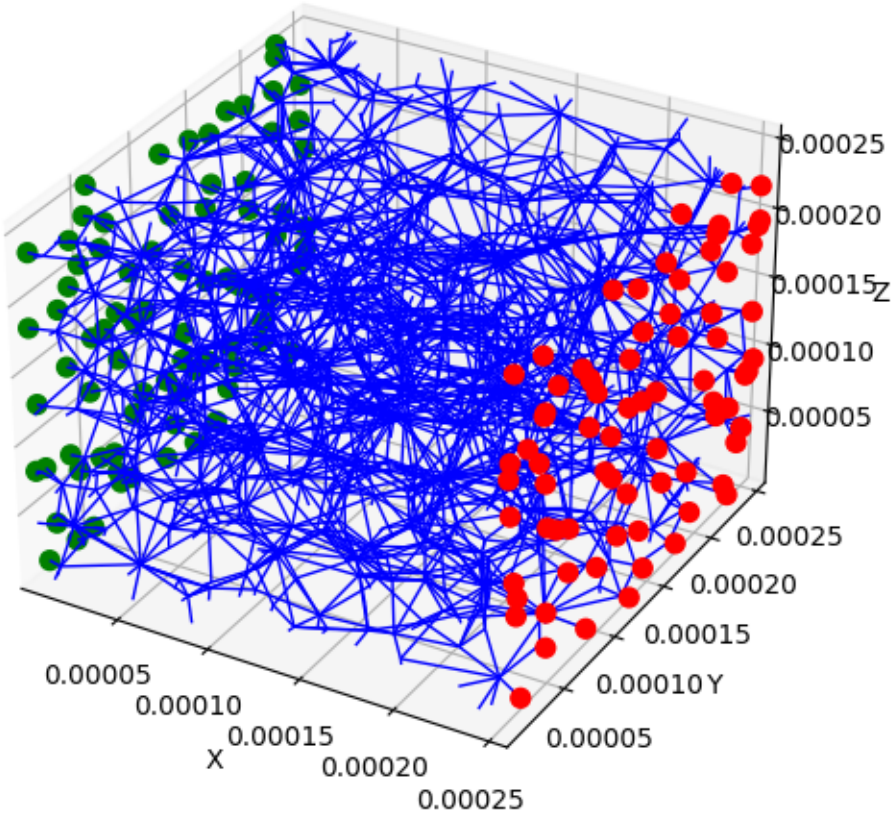


[40]:
```
settings = {'pore_shape': 'pyramid',
            'throat_shape': 'cuboid',

            'pore_diameter': 'equivalent_diameter',
            'throat_diameter': 'inscribed_diameter'}


#NBVAL_IGNORE_OUTPUT
```

```
resolution = 1e-6
snow = ps.networks.snow2(im, voxel_size=resolution,accuracy='standard')
proj= op.io.PoreSpy.import_data(snow.network, settings=settings)
net = proj[0]
geo =proj[1]

#net = op.io.network_from_porespy(snow.network)

pn = ps.networks.label_boundaries(net,labels=[['left', 'right'], ['front', 'back'], [␣
↪'bottom','top']], tol=1e-9)
```

[41]:
```
proj= pn.project

geo['pore.area']=geo['pore.surface_area']

geo['throat.area'] =  geo['throat.cross_sectional_area']

geo['throat.surface_area'] = geo['throat.volume']/geo['throat.equivalent_diameter']

geo['throat.conduit_lengths.pore1']= geo['throat.length']
geo['throat.conduit_lengths.pore2']= geo['throat.length']

geo.add_model(propname='throat.endpoints',
              model=mods.geometry.throat_endpoints.spherical_pores)
geo.add_model(propname='pore.area',
              model=mods.geometry.pore_cross_sectional_area.sphere)
geo.add_model(propname='pore.volume',
              model=mods.geometry.pore_volume.sphere)
geo.add_model(propname='throat.length',
              model=mods.geometry.throat_length.ctc)
geo.add_model(propname='throat.volume',
              model=mods.geometry.throat_volume.cylinder)
geo.add_model(propname='throat.area',
              model=mods.geometry.throat_cross_sectional_area.cylinder)
geo.add_model(propname='throat.surface_area',
              model=mods.geometry.throat_surface_area.cylinder)
geo.add_model(propname='throat.endpoints',
              model=mods.geometry.throat_endpoints.spherical_pores)
geo.add_model(propname='throat.conduit_lengths',
              model=mods.geometry.throat_length.conduit_lengths)
```

[42]:
```
fig, ax = plt.subplots()
op.topotools.plot_connections(network=pn, ax=ax)
op.topotools.plot_coordinates(network=pn,
                              ax=ax,
                              pores=pn['pore.right'],
                              markersize=50, c='r');
op.topotools.plot_coordinates(network=pn,
                              ax=ax,
                              pores=pn['pore.left'],
                              markersize=50, c='g');
```

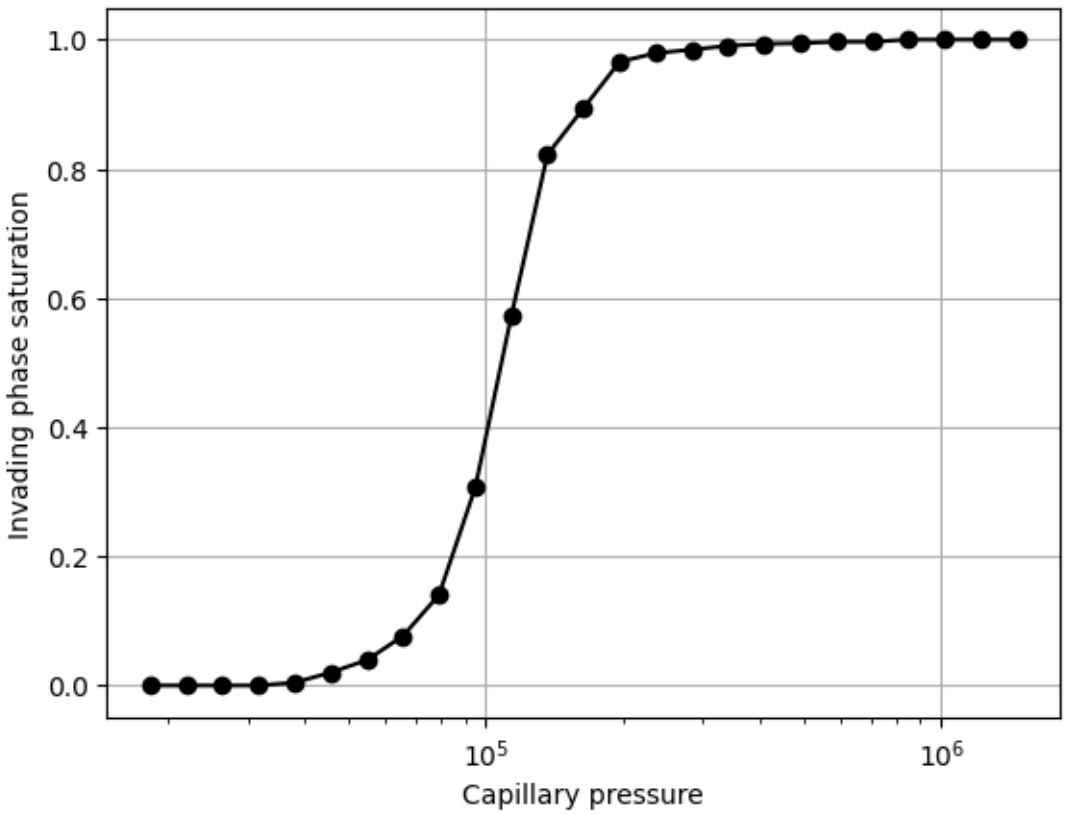


[43]:
```
h = pn.check_network_health()
op.topotools.trim(network=pn, pores=h['trim_pores'])
h = pn.check_network_health()
print(h)

# Handle Boundary Pores - Zero Volume for saturation but not zero diam and area
# For flow calculations
geo['pore.volume'][geo['pore.area'] == 0.0] = 0
geo['pore.diameter'][geo['pore.diameter'] == 0.0] = geo['pore.diameter'].max()
geo['pore.area'][geo['pore.area'] == 0.0] = geo['pore.area'].max()
```

```
--------------------------------------------------------------------------------
key                                 value
--------------------------------------------------------------------------------
disconnected_clusters               []
isolated_pores                      []
trim_pores                          []
duplicate_throats                   []
bidirectional_throats               []
headless_throats                    []
looped_throats                      []
--------------------------------------------------------------------------------
```

[44]:
```
Hg = op.phases.Mercury(network=pn)
phys = op.physics.Standard(network=pn, phase=Hg, geometry=geo)
# Create algorithm and run simulation
mip = op.algorithms.Porosimetry(network=pn, phase=Hg)
mip.set_inlets(pores=pn.pores(['left', 'right', 'top', 'bottom']))
mip.run()
mip.plot_intrusion_curve()
```

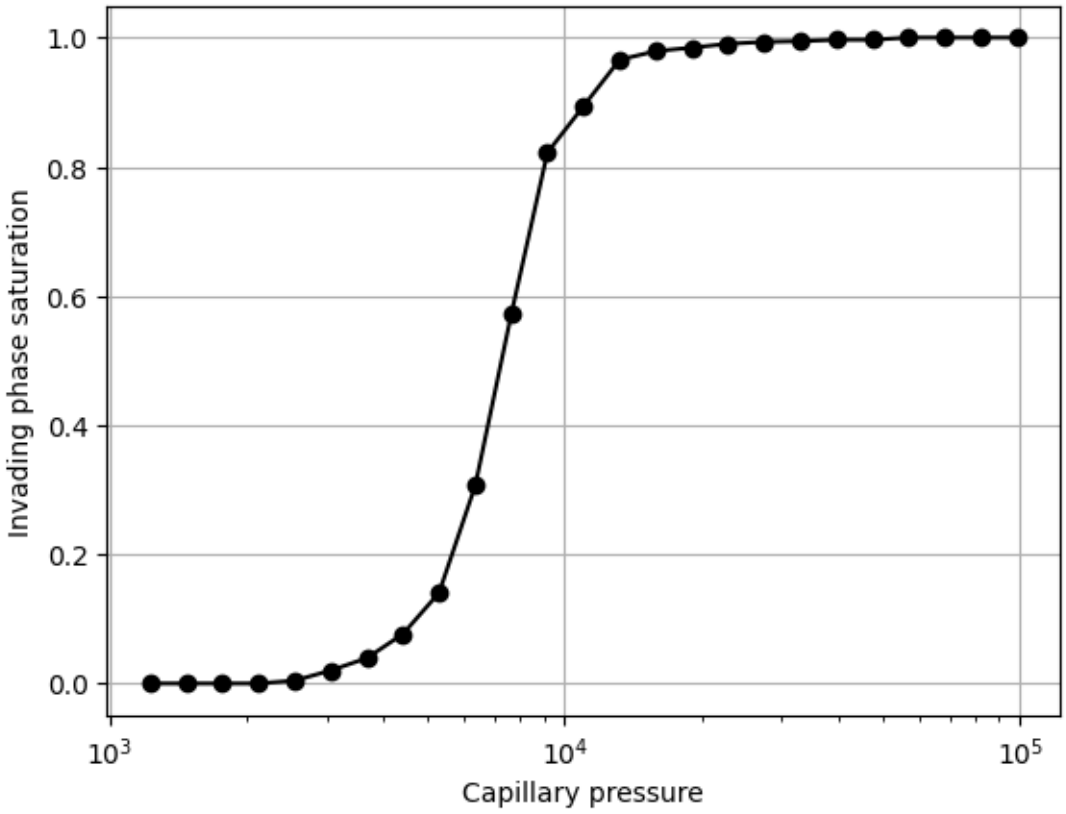
```
------------------------------------------------------------
WARNING    : Inlets have non-zero volume, percolation curve won't start at 0.
SOURCE     : openpnm.algorithms.OrdinaryPercolation.get_intrusion_data
TIME STAMP : 2022-11-03 15:23:55,326
------------------------------------------------------------
```



[45]:
```
water = op.phases.Water(network=pn)
phys = op.physics.Standard(network=pn, phase=water, geometry=geo)
# Create algorithm and run simulation
mip = op.algorithms.Porosimetry(network=pn, phase=water)
mip.set_inlets(pores=pn.pores(['left', 'right', 'top', 'bottom']))
mip.run()
mip.plot_intrusion_curve()
```

```
------------------------------------------------------------
WARNING    : Inlets have non-zero volume, percolation curve won't start at 0.
SOURCE     : openpnm.algorithms.OrdinaryPercolation.get_intrusion_data
```

```
TIME STAMP : 2022-11-03 15:23:55,629
------------------------------------------------------------
```



Assign phases

[46]:
```
air = op.phases.Air(network=pn,name='air')
water = op.phases.Water(network = pn, name = 'water')
# Reset pore contact angle
water['pore.contact_angle'] = 110.0
#water['pore.temperature'] = 353
```

[47]:
```
phys_air = op.physics.Standard(network=pn, phase=air, geometry=geo)
phys_water = op.physics.Standard(network=pn, phase=water, geometry=geo)
#phys_water = op.physics.Basic(network=pn, phase=water, geometry=geo)
```

[48]:
```
OP_1 = op.algorithms.OrdinaryPercolation(network=pn)
OP_1.set_inlets(pores=pn.pores('bottom'))
OP_1.set_outlets(pores=pn.pores('top'))
OP_1.settings['trapping']=True
OP_1.setup(phase=water, pore_volume='pore.volume', throat_volume='throat.
 ↪volume',access_limited=True)
OP_1.run(points=100)
```

[49]:
```
data = OP_1.get_intrusion_data()

# Filter for evenly spaced sat inc. first and last

filter_pc = [data.Pcap[0]]
sat = [data.Snwp[0]]
for i, pc in enumerate(data.Pcap):
    if  data.Snwp[i] - sat[-1] > 0.01:
        filter_pc.append(pc)
        sat.append(data.Snwp[i])
filter_pc.append(data.Pcap[-1])
sat.append(data.Snwp[-1])
```

```
------------------------------------------------------------
WARNING    : Inlets have non-zero volume, percolation curve won't start at 0.
SOURCE     : openpnm.algorithms.OrdinaryPercolation.get_intrusion_data
TIME STAMP : 2022-11-03 15:23:56,094
------------------------------------------------------------
```

[50]:
```
def update_phase_and_phys(results):
    water['pore.occupancy'] =results['pore.occupancy']
    air['pore.occupancy'] = 1-results['pore.occupancy']
    water['throat.occupancy']= results['throat.occupancy']
    air['throat.occupancy'] = 1-results['throat.occupancy']
    # Add multiphase conductances
    mode='loose'
    phys_air.add_model(model=mods.physics.multiphase.conduit_conductance,
                       propname='throat.conduit_diffusive_conductance',
                       throat_conductance='throat.diffusive_conductance',
                       mode=mode)
    phys_water.add_model(model=mods.physics.multiphase.conduit_conductance,
                         propname='throat.conduit_diffusive_conductance',
                         throat_conductance='throat.diffusive_conductance',
                         mode=mode)
    phys_air.add_model(model=mods.physics.multiphase.conduit_conductance,
                       propname='throat.conduit_hydraulic_conductance',
                       throat_conductance='throat.hydraulic_conductance',
                       mode=mode)
    phys_water.add_model(model=mods.physics.multiphase.conduit_conductance,
                         propname='throat.conduit_hydraulic_conductance',
                         throat_conductance='throat.hydraulic_conductance',
                         mode=mode)
```

[51]:
```
update_phase_and_phys(OP_1.results(Pc=1e3))
```

[52]:
```
atm1=0
atm2=1
```

[53]:
```
perm_air = {'0': [], '1': [], '2': []}
diff_air = {'0': [], '1': [], '2': []}
perm_water = {'0': [], '1': [], '2': []}
diff_water = {'0': [], '1': [], '2': []}

max_Pc = max(OP_1['throat.invasion_pressure'])
```

```python
num_seq = 20
pore_volumes = pn['pore.volume']
throat_volumes = pn['throat.volume']
totV = np.sum(pore_volumes) + np.sum(throat_volumes)

K_air_single_phase = [None, None, None]
D_air_single_phase = [None, None, None]
K_water_single_phase = [None, None, None]
D_water_single_phase = [None, None, None]

bounds = [['xmin', 'xmax'], ['ymin', 'ymax'], ['zmax', 'zmin']]

for bound_increment in range(len(bounds)):
    # Run Single phase algs effective properties
    BC1_pores = pn.pores(labels=bounds[bound_increment][0])
    BC2_pores = pn.pores(labels=bounds[bound_increment][1])

    # Effective permeability : air
    sf_air = op.algorithms.StokesFlow(network=pn, phase=air)
    sf_air.setup(conductance='throat.hydraulic_conductance')
    sf_air.set_value_BC(values=atm1, pores=BC1_pores)
    sf_air.set_value_BC(values=atm2, pores=BC2_pores)
    sf_air.run()
    K_air_single_phase[bound_increment] = sf_air.calc_effective_permeability()
    proj.purge_object(obj=sf_air)

    # Effective diffusivity : air
    fd_air = op.algorithms.FickianDiffusion(network=pn,phase=air)
    fd_air.setup(conductance='throat.diffusive_conductance')

    ## Setting Value Boundary Conditions
    fd_air.set_value_BC(values=atm1, pores=BC1_pores)
    fd_air.set_value_BC(values=atm2, pores=BC2_pores)
    fd_air.run()
    D_air_single_phase[bound_increment] = fd_air.calc_effective_diffusivity()
    proj.purge_object(obj=fd_air)

    # Effective permeability : water
    sf_water = op.algorithms.StokesFlow(network=pn, phase=water)
    sf_water.setup(conductance='throat.hydraulic_conductance')
    sf_water.set_value_BC(values=atm1, pores=BC1_pores)
    sf_water.set_value_BC(values=atm2, pores=BC2_pores)
    sf_water.run()
    K_water_single_phase[bound_increment] = sf_water.calc_effective_permeability()
    proj.purge_object(obj=sf_water)

    # Effective diffusivity : water
    fd_water = op.algorithms.FickianDiffusion(network=pn,phase=water)
    fd_water.setup(conductance='throat.diffusive_conductance')
    fd_water.set_value_BC(values=atm1, pores=BC1_pores)
    fd_water.set_value_BC(values=atm2, pores=BC2_pores)
    fd_water.run()
```

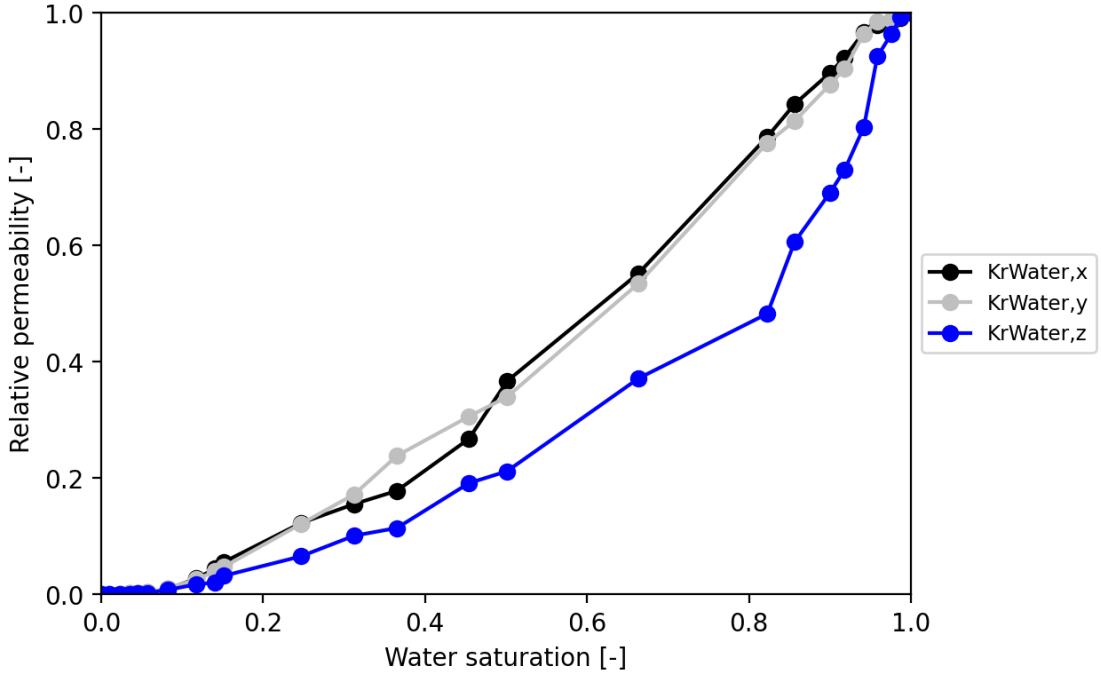
```
    D_water_single_phase[bound_increment] = fd_water.calc_effective_diffusivity()
    proj.purge_object(obj=fd_water)
```

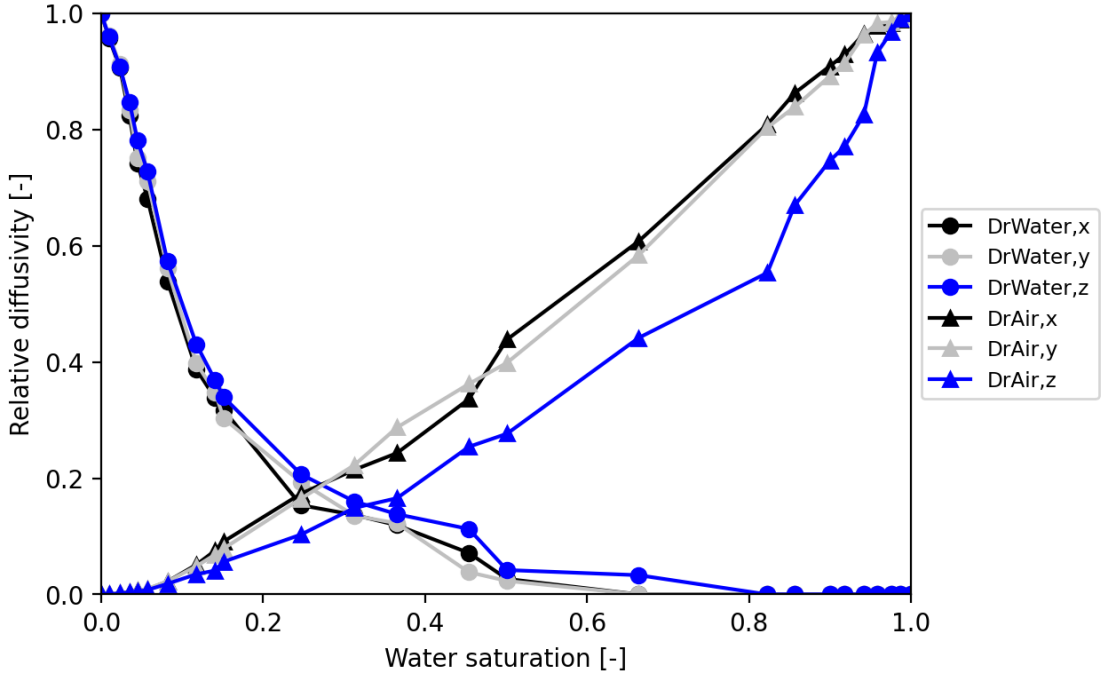
[54]:
```
for Pc in filter_pc:
    update_phase_and_phys(OP_1.results(Pc=Pc))
    print('-' * 80)
    print('Pc', Pc)
    for bound_increment in range(len(bounds)):
        BC1_pores = pn.pores(labels=bounds[bound_increment][0])
        BC2_pores = pn.pores(labels=bounds[bound_increment][1])

        # Multiphase
        sf_air = op.algorithms.StokesFlow(network=pn,phase=air)
        sf_air.setup(conductance='throat.conduit_hydraulic_conductance')
        sf_water = op.algorithms.StokesFlow(network=pn,phase=water)
        sf_water.setup(conductance='throat.conduit_hydraulic_conductance')

        fd_air = op.algorithms.FickianDiffusion(network=pn,phase=air)
        fd_air.setup(conductance='throat.conduit_diffusive_conductance')
        fd_water = op.algorithms.FickianDiffusion(network=pn,phase=water)
        fd_water.setup(conductance='throat.conduit_diffusive_conductance')

        #BC1
        sf_air.set_value_BC(values=atm1, pores=BC1_pores)
        sf_water.set_value_BC(values=atm1, pores=BC1_pores)
        fd_air.set_value_BC(values=atm1, pores=BC1_pores)
        fd_water.set_value_BC(values=atm1, pores=BC1_pores)

        #BC2
        sf_air.set_value_BC(values=atm2, pores=BC2_pores)
        sf_water.set_value_BC(values=atm2, pores=BC2_pores)
        fd_air.set_value_BC(values=atm2, pores=BC2_pores)
        fd_water.set_value_BC(values=atm2, pores=BC2_pores)

        # Run Multiphase algs
        sf_air.run()
        sf_water.run()
        fd_air.run()
        fd_water.run()

        Keff_air_mphase = sf_air.calc_effective_permeability()
        Deff_air_mphase = fd_air.calc_effective_diffusivity()
        Keff_water_mphase = sf_air.calc_effective_permeability()
        Deff_water_mphase = fd_water.calc_effective_diffusivity()

        Kr_eff_air = Keff_air_mphase / K_air_single_phase[bound_increment]
        Kr_eff_water = Keff_water_mphase / K_water_single_phase[bound_increment]
        Dr_eff_air = Deff_air_mphase / D_air_single_phase[bound_increment]
        Dr_eff_water = Deff_water_mphase / D_water_single_phase[bound_increment]

        perm_air[str(bound_increment)].append(Kr_eff_air)
        diff_air[str(bound_increment)].append(Dr_eff_air)
        perm_water[str(bound_increment)].append(Kr_eff_water)
```

```
        diff_water[str(bound_increment)].append(Dr_eff_water)


        proj.purge_object(obj=sf_air)
        proj.purge_object(obj=sf_water)
        proj.purge_object(obj=fd_air)
        proj.purge_object(obj=fd_water)
```

[58]:
```
from matplotlib.font_manager import FontProperties
%matplotlib inline
print(sat)
print(perm_water['0'][:])

#sat = 1-np.asarray(sat)

fontP = FontProperties()
fontP.set_size('small')
# Setting up subplots
fig = plt.figure(figsize=(8, 8), dpi=200, facecolor='w', edgecolor='k')
ax1 = fig.add_subplot(211)   #top
ax2 = fig.add_subplot(212)   #bottom

x_values1 = [x/20 for x in range(21)]
z = '.75'
perm_water['0'] = perm_water['0']
# Plots for subplot1 - strict permeability
p1, = ax1.plot(sat, perm_water['0'], color = 'k', linestyle = '-', marker = 'o')
p2, = ax1.plot(sat, perm_water['1'], color = z, linestyle = '-', marker = 'o')
p3, = ax1.plot(sat, perm_water['2'], color = 'b', linestyle = '-', marker = 'o')
#p4, = ax1.plot(sat, perm_air['0'], color = 'k', linestyle = '-', marker = '^')
#p5, = ax1.plot(sat, perm_air['1'], color = z, linestyle = '-', marker = '^')
#p6, = ax1.plot(sat, perm_air['2'], color = 'b', linestyle = '-', marker = '^')
#p10, = ax1.plot(x_values1, [x**(3) for x in x_values1], 'k--')
#ax1.plot(x_values1, [(1-x)**(3) for x in x_values1], 'k--')
#gs1, = ax1.plot(gostick_saturation_1, gostick_perm_air_case1, color = 'r', linestyle␣
↪= '-', marker = 'D')
#gs2, = ax1.plot(gostick_saturation_2, gostick_perm_water, color = 'r', linestyle =␣
↪'-', marker = 'o')
ax1.set_ylabel('Relative permeability [-]')
ax1.set_xlabel("Water saturation [-]")
ax1.set_ylim([0,1])
ax1.set_xlim([0,1])

# Need to work on legend to match up with the right things
lgd1 = ax1.legend([p1, p2, p3, p4, p5, p6, p10,
                   # gs1, gs2
                  ],
                  ["KrWater,x", "KrWater,y", "KrWater,z",
                  # "KrAir,x","KrAir,y","KrAir,z", "a = 3"
                   #"Gostick et al \n KrAir,x (case 1)",
                   #"Gostick et al \n KrWater,x"
                   ],
                  loc='center left', bbox_to_anchor=(1, 0.5), prop=fontP)
```

```
#sat = 1-np.asarray(sat)
# Plots for subplot4 - diffusivity
p11, = ax2.plot(sat, diff_water['0'], color = 'k', linestyle = '-', marker = 'o')
p12, = ax2.plot(sat, diff_water['1'], color = z, linestyle = '-', marker = 'o')
p13, = ax2.plot(sat, diff_water['2'], color = 'b', linestyle = '-', marker = 'o')
p14, = ax2.plot(sat, diff_air['0'], color = 'k', linestyle = '-', marker = '^')
p15, = ax2.plot(sat, diff_air['1'], color = z, linestyle = '-', marker = '^')
p16, = ax2.plot(sat, diff_air['2'], color = 'b', linestyle = '-', marker = '^')
#p20, = ax2.plot(x_values1, [x**(2) for x in x_values1], 'k--')
#ax2.plot(x_values1, [(1-x)**(2) for x in x_values1], 'k--')
#gs3, = ax2.plot(gostick_saturation_3, gostick_diff_air_case1, color = 'r', linestyle␣
↪= '-', marker = 'D')
#gs4, = ax2.plot(gostick_saturation_4, gostick_diff_water, color = 'r', linestyle =␣
↪'-', marker = 'o')
ax2.set_ylabel('Relative diffusivity [-]')
ax2.set_xlabel("Water saturation [-]")
ax2.set_ylim([0,1])
ax2.set_xlim([0,1])

lgd2 = ax2.legend([p11, p12, p13, p14, p15, p16, p20,
                   # gs3, gs4
                    ],
                  ["DrWater,x", "DrWater,y", "DrWater,z",
                   "DrAir,x","DrAir,y","DrAir,z",
                   #"a = 2",
                  # "Gostick et al \n DrAir,x (case 1)",
                  # "Gostick et al \n DrWater,x"],
                   ],
                  loc='center left', bbox_to_anchor=(1, 0.5), prop=fontP)

fig.subplots_adjust(left=0.13, right=.7, top=0.95, bottom=0.05)

plt.show()
```


Relative permeability [-]
Water saturation [-]
KrWater,x
KrWater,y
KrWater,z


Relative diffusivity [-]
Water saturation [-]
DrWater,x
DrWater,y
DrWater,z
DrAir,x
DrAir,y
DrAir,z

References

[Gostick(2017)] J T Gostick. Versatile and efficient pore network extraction method using marker-based watershed segmentation. *Physical Review E*, 96(2):023307, 2017.