Determining the Effective Diffusivity of a Network Using Transient Simulations

Background

Effective diffusivity is an important property for a diffusing species in a porous material, and is defined by the following equation:

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

Pore Network Modeling can be used to determine the effective diffusivity value for a given network and diffusing species. A previous notebook (Finding Effective Diffusivity and Tortuosity of a Network) demonstrated finding the effective diffusivity using OpenPNM's `FickianDiffusion` algorithm, which is a steady state algorithm.

This notebook will attempt to obtain the effective diffusivity using OpenPNM's transient `TransientFickianDiffusion` algorithm on a 2D pore network with the length l.

To simulate this, a system was chosen with a constant nonzero concentration at the left boundary, zero concentration at the right boundary, and an initial value of 0, illustrated below:

This method of determining the effective diffusivity involves employing a different strategy, utilizing the concentration profile for each pore with time to fit an analytical solution, with D_{eff} as the fitting parameter. Given the boundary conditions above, the analytical solution is obtained [1], described by the following simplified equation:

$$C(t) = C_l - C_l\frac{x}{l} - \frac{2}{\pi}\sum_{n=1}^{\infty}\frac{C_l}{n}\sin\left(\frac{n\pi x}{l}\right)\exp\left(-D_{eff}\frac{n^2\pi^2}{l^2}t\right)$$

The following sections explore this method by showing several simulations for the diffusion of oxygen in air, starting with a simulation for a bulk domain (no pores), then testing on a pore network with uniformly-sized pores, and finally, a pore network with a pore size distribution.

Part 1: Simulating the Bulk

First, the model is employed on a pore network representing a "bulk" domain to test

whether the curve fitting for D_{eff} would result in the bulk diffusivity of oxygen in air.

The `cubes_and_cuboids` geometry on a 40 by 40 pore network was used, with throat length being set to 0.

In [10]:
```
C_l = 1
length = 0.01
pore_num = 40
spacing = length/pore_num

import openpnm as op
import numpy as np

pn = op.network.Cubic(shape=[pore_num, pore_num, 1], spacing = spacing)

f = op.models.collections.geometry.cubes_and_cuboids
pn.add_model_collection(f)
pn.regenerate_models()

def pore_diameter(network):
    return spacing

def throat_diameter(network):
    return spacing

def throat_length(network):
    return 0

pn.add_model(propname='pore.diameter', model=pore_diameter)
pn.add_model(propname='throat.diameter', model=throat_diameter)
pn.add_model(propname='throat.length', model=throat_length)
pn.regenerate_models()
```

Next, the phase is defined, including the diffusion model to be used.

In [12]:
```
air = op.phase.Air(network=pn)

f_diffusive = op.models.physics.diffusive_conductance.generic_diffusive
air.add_model(propname='throat.diffusive_conductance', model=f_diffusive)
```

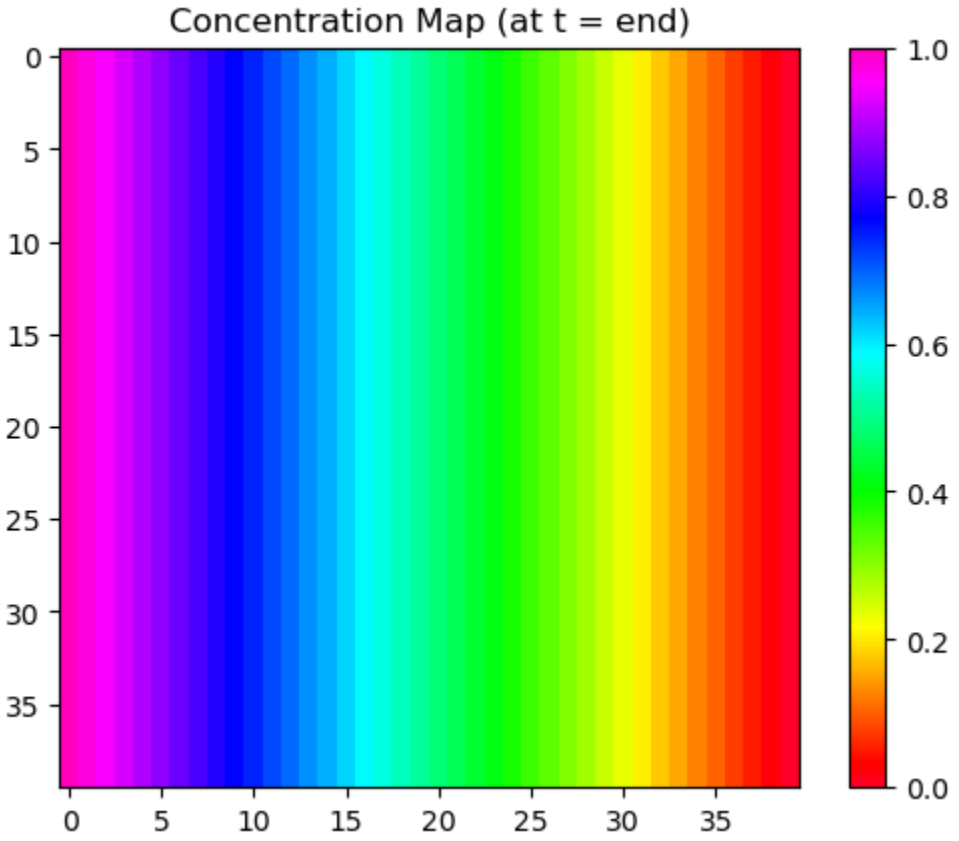
Then, the transient algorithm is defined, along with the simulation settings and the initial and boundary conditions.

In [14]:
```
tfd = op.algorithms.TransientFickianDiffusion(phase=air, network=pn)
start_time = 0     # [s]
end_time = 5     # [s]
time_span = (start_time, end_time)

initial_concentration = 0

left_value = C_l
right_value = 0
tfd.set_BC(pores=pn.pores('left'), bctype='value', bcvalues=left_value)
```

```
tfd.set_BC(pores=pn.pores('right'), bctype='value', bcvalues=right_value)

times_to_save = np.linspace(start_time, end_time, 100)
```

Running the transient simulation would result in the following concentration profile

In [16]:
```
tfd.run(x0=initial_concentration, tspan=time_span, saveat=times_to_save)
concentrations = np.transpose(tfd.soln['pore.concentration'])

import matplotlib.pyplot as plt

concs = np.rot90(np.reshape(concentrations[-1].T, (pore_num,pore_num)))
loc = plt.imshow(concs, cmap='gist_rainbow')
plt.title('Concentration Map (at t = end)')
plt.colorbar(loc)
plt.show()
```



To fit for the effective diffusivity, the function for the analytical solution is created. Note that since the equation contains an infinite series, the maximum value for n was bounded at 100.

In [18]:
```
import math

sum_num = 100
def analytical_func(t, diff_eff):
    effective_length = pn['pore.coords'][-1][0] - pn['pore.coords'][0][0]
    x = pn['pore.coords'][pore][0] - pn['pore.coords'][0][0]
```

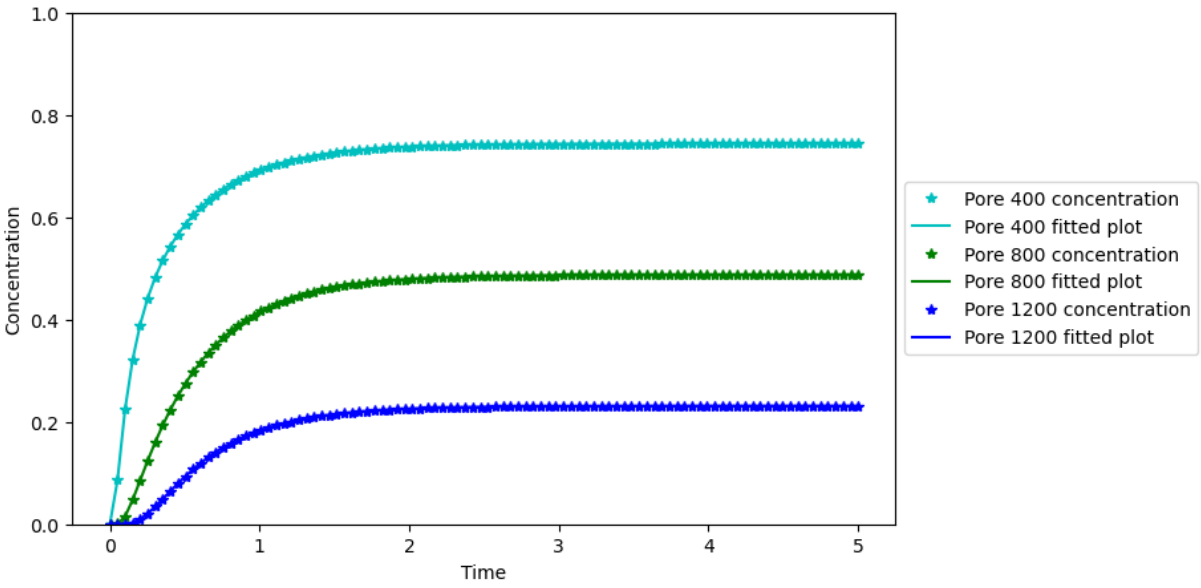
```
    sum_value = 0
    for i in range(1, sum_num):
        sum_value = sum_value + (C_l/i)*np.sin(i*math.pi*x/effective_length)
    conc = C_l - C_l*x/effective_length - (2/(math.pi))*(sum_value)
    return conc
```

With the concentration profiles and the analytical solution function available, D_{eff} can now be determined. The `scipy` function `curve_fit` was used to fit the concentration data with the analytical model. As a start, 3 sample pores were selected to observe the effectiveness of the fitting.

Note that since the effective diffusivity controls the shape of the concentration rise, and the $C(t)$ data from the simulation approaches a plateau, the data to fit was limited to the first 90% of the rise to ensure correct fitting.

In [20]:
```
from scipy.optimize import curve_fit

sample_pores = [400, 800, 1200]
fig, ax = plt.subplots(figsize=(8, 5))
f = analytical_func
data_colors = ['c*', 'g*', 'b*', 'm*']
curve_colors = ['c-', 'g-', 'b-', 'm-']
ctr0 = 0

for pore in sample_pores:
    label = 'Pore ' + str(pore) + ' concentration'
    ax.plot(times_to_save, concentrations[:,pore], data_colors[ctr0], label=

    # 1D approximation
    idx_start = len(concentrations[:,pore][concentrations[:,pore] < 0*max(co
    idx_stop = len(concentrations[:,pore][concentrations[:,pore] < 0.9*max(c
    popt, pcov = curve_fit(f, times_to_save[idx_start:idx_stop,], concentrat
    print('D_eff for pore',  + pore, ':', '\t', popt[0])

    # plot approximated plot
    label = 'Pore ' + str(pore) + ' fitted plot'
    ax.plot(times_to_save, analytical_func(times_to_save, popt[0]), curve_co

    ctr0 += 1


ax.set_ylabel('Concentration')
ax.set_xlabel('Time')
ax.set_ylim([0, 1])
ax.legend(loc='center left', bbox_to_anchor=(1, 0.5))
plt.show()
```

```
D_eff for pore 400 :     2.0948816829585677e-05
D_eff for pore 800 :     2.094797939153065e-05
D_eff for pore 1200 :    2.0947777505756078e-05
```

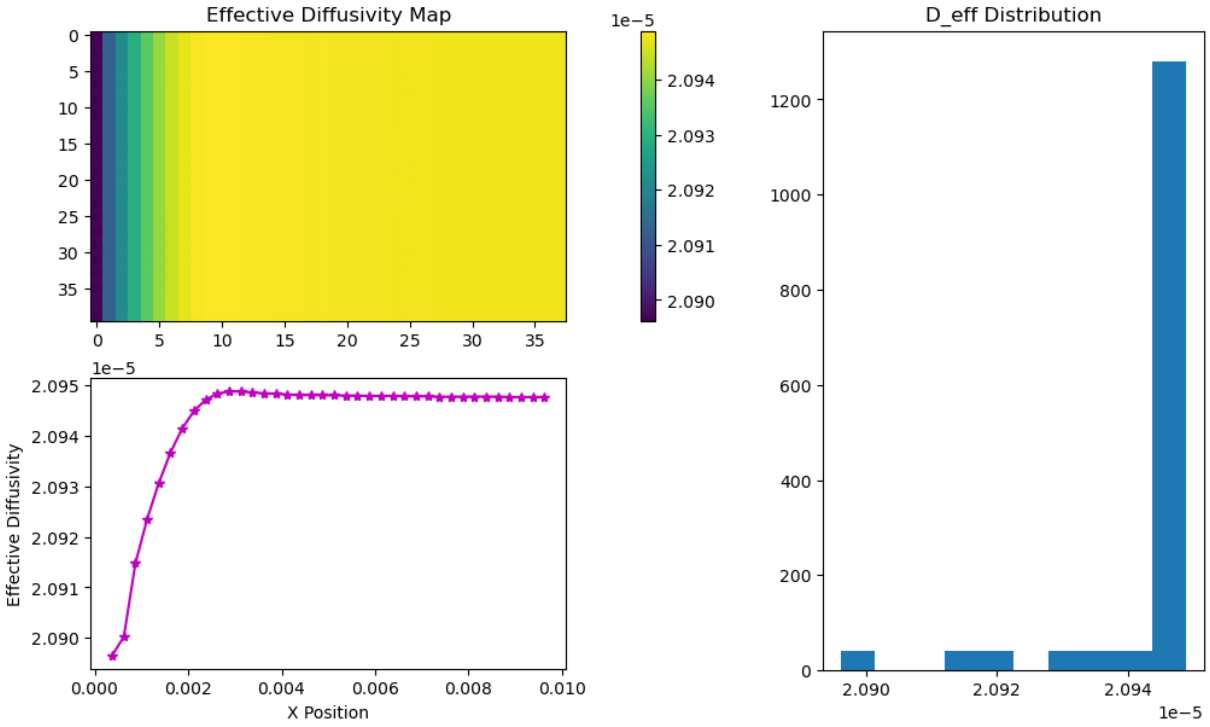


We can see here that the effective diffusivities of each pore are around $2.095 \cdot 10^{-5} \frac{m^2}{s}$, which is the expected value for the bulk diffusivity of oxygen in air. This means that the fitting is effective for a bulk domain

Next, the effective diffusivity values are obtained for every pore (except the boundary pores) to see the variation and distribution of the calculated values.

In [23]:
```
full_pores = np.arange(pore_num*1, len(pn.pores()) - pore_num)
d_eff_array = []
d_eff_profile = []
x_arr = []
d_eff_profile_buffer = []
ctr0 = 0
for pore in full_pores:
    # 1D approximation
    idx_start = len(concentrations[:,pore][concentrations[:,pore] < 0*max(co
    idx_stop = len(concentrations[:,pore][concentrations[:,pore] < 0.9*max(c
    popt, pcov = curve_fit(f, times_to_save[idx_start:idx_stop,], concentrat
    d_eff_array.append(popt[0])
    d_eff_profile_buffer.append(popt[0])

    ctr0 += 1

    if ctr0 % pore_num == 9:
        d_eff_profile.append(np.mean(d_eff_profile_buffer))
        x_arr.append(pn['pore.coords'][pore][0])
        d_eff_profile_buffer = []

fig, ax = plt.subplots(2, 2, figsize=(10, 6), layout='constrained')
d_eff_map = np.rot90(np.reshape(d_eff_array, (pore_num-2,pore_num)))
pos = ax[0][0].imshow(d_eff_map)
ax[0][0].set_title('Effective Diffusivity Map')
ax[0][0].set_aspect('auto')
ax[1][0].plot(x_arr, d_eff_profile, 'm*-')
ax[1][0].set_xlabel('X Position')
```

```
ax[1][0].set_ylabel('Effective Diffusivity')
ax[0][1].axis('off')
ax[1][1].axis('off')
fig.colorbar(pos, ax=ax[0][0])
plt.subplot(1, 3, 3)
plt.hist(d_eff_array, 10)
plt.title('D_eff Distribution')
plt.locator_params(axis='x', nbins=5)
plt.show()
```



As we can see, all pores have effective diffusivities that are close to the expected value of $2.095 \cdot 10^{-5} \frac{m^2}{s}$. We can see that the effective diffusivities get lower the nearer we get to the left boundary. However, since the values are still very close to the expected value, this is still acceptable since the error is relatively small. This could be coming from numerical errors, such as errors related to the transient solver, and the fact that fewer points for fitting are obtained for the pores near the left boundary, since the rise is more rapid there.

Part 2: Simulating a Pore Network with Uniform Sizes

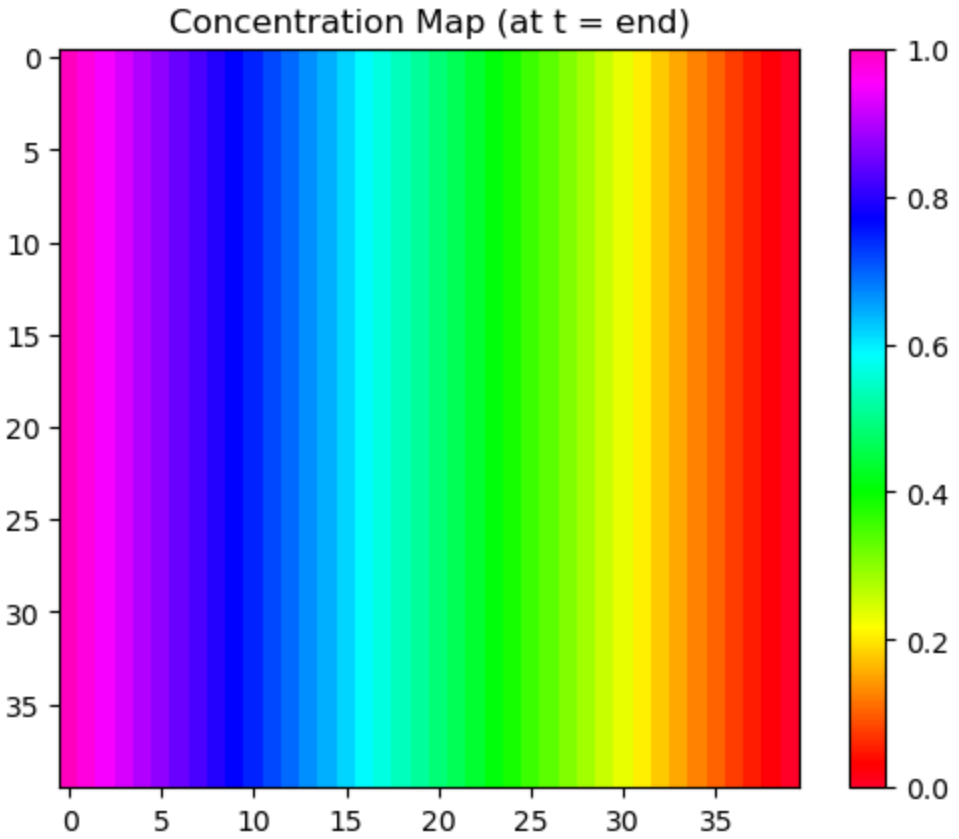
The method will then be tested for an actual pore network with a uniform pore size. For this, the `cubes_and_cuboids` geometry will still be used for simplicity, and the pore and throat diameters were arbitrarily defined.

In [27]:
```
pore_diam = spacing*0.75
throat_diam_factor = 0.5

def pore_diameter(network):
    return pore_diam
```

```
def throat_length(network):
    return spacing - pn['pore.diameter'][pn['throat.conns'][:,0]]/2 - pn['po

def throat_diameter(network):
    return pore_diam*throat_diam_factor


pn = op.network.Cubic(shape=[pore_num, pore_num, 1], spacing = spacing)

f = op.models.collections.geometry.cubes_and_cuboids
pn.add_model_collection(f)
pn.add_model(propname='pore.diameter', model=pore_diameter)
pn.add_model(propname='throat.diameter', model=throat_diameter)
pn.add_model(propname='throat.length', model=throat_length)
pn.regenerate_models()
```

The same procedures are done with the same settings for running the transient simulation.

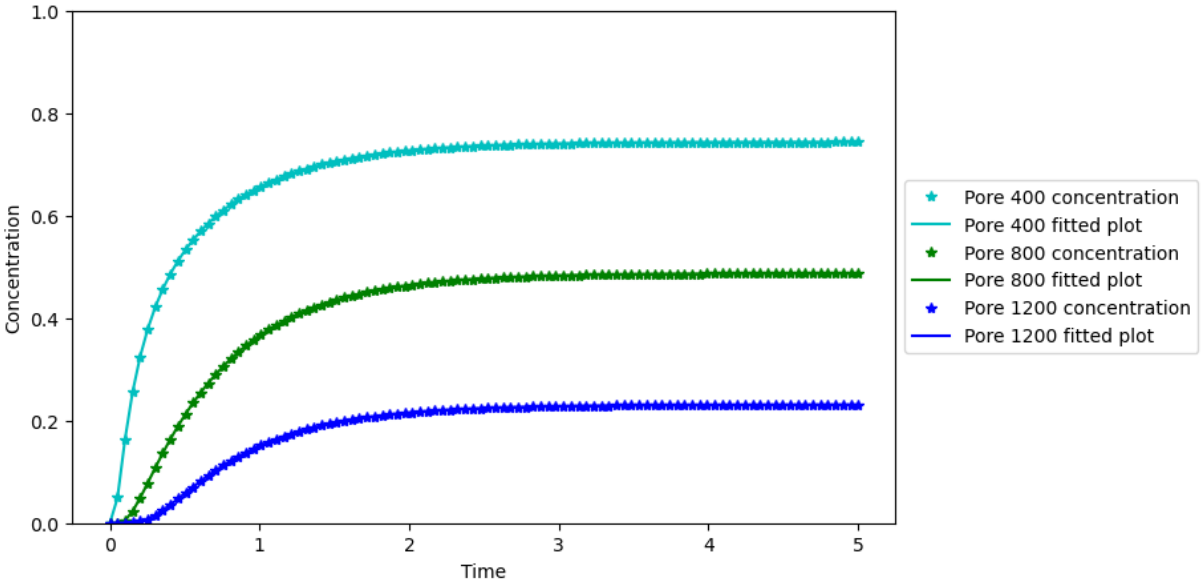
In [29]:
```
air = op.phase.Air(network=pn)

f_diffusive = op.models.physics.diffusive_conductance.generic_diffusive
air.add_model(propname='throat.diffusive_conductance', model=f_diffusive)

tfd = op.algorithms.TransientFickianDiffusion(phase=air, network=pn)

tfd.set_BC(pores=pn.pores('left'), bctype='value', bcvalues=left_value)
tfd.set_BC(pores=pn.pores('right'), bctype='value', bcvalues=right_value)

tfd.run(x0=initial_concentration, tspan=time_span, saveat=times_to_save)
concentrations = np.transpose(tfd.soln['pore.concentration'])

concs = np.rot90(np.reshape(concentrations[-1].T, (pore_num,pore_num)))
loc = plt.imshow(concs, cmap='gist_rainbow')
plt.colorbar(loc)
plt.title('Concentration Map (at t = end)')
plt.show()
```



Here, we can see that the concentration gradient, similar to the bulk case, still has no variation in the y-direction because of the homegeneity of the pore sizes.

In [31]:
```
fig, ax = plt.subplots(figsize=(8, 5))
f = analytical_func
ctr0 = 0

for pore in sample_pores:
    label = 'Pore ' + str(pore) + ' concentration'
    ax.plot(times_to_save, concentrations[:,pore], data_colors[ctr0], label=

    # 1D approximation
    idx_start = len(concentrations[:,pore][concentrations[:,pore] < 0*max(co
    idx_stop = len(concentrations[:,pore][concentrations[:,pore] < 0.9*max(c
    popt, pcov = curve_fit(f, times_to_save[idx_start:idx_stop,], concentrat
    print('D_eff for pore',  + pore, ':', '\t', popt[0])

    # plot approximated plot
    label = 'Pore ' + str(pore) + ' fitted plot'
    ax.plot(times_to_save, analytical_func(times_to_save, popt[0]), curve_co

    ctr0 += 1


ax.set_ylabel('Concentration')
ax.set_xlabel('Time')
ax.set_ylim([0, 1])
ax.legend(loc='center left', bbox_to_anchor=(1, 0.5))
plt.show()
```

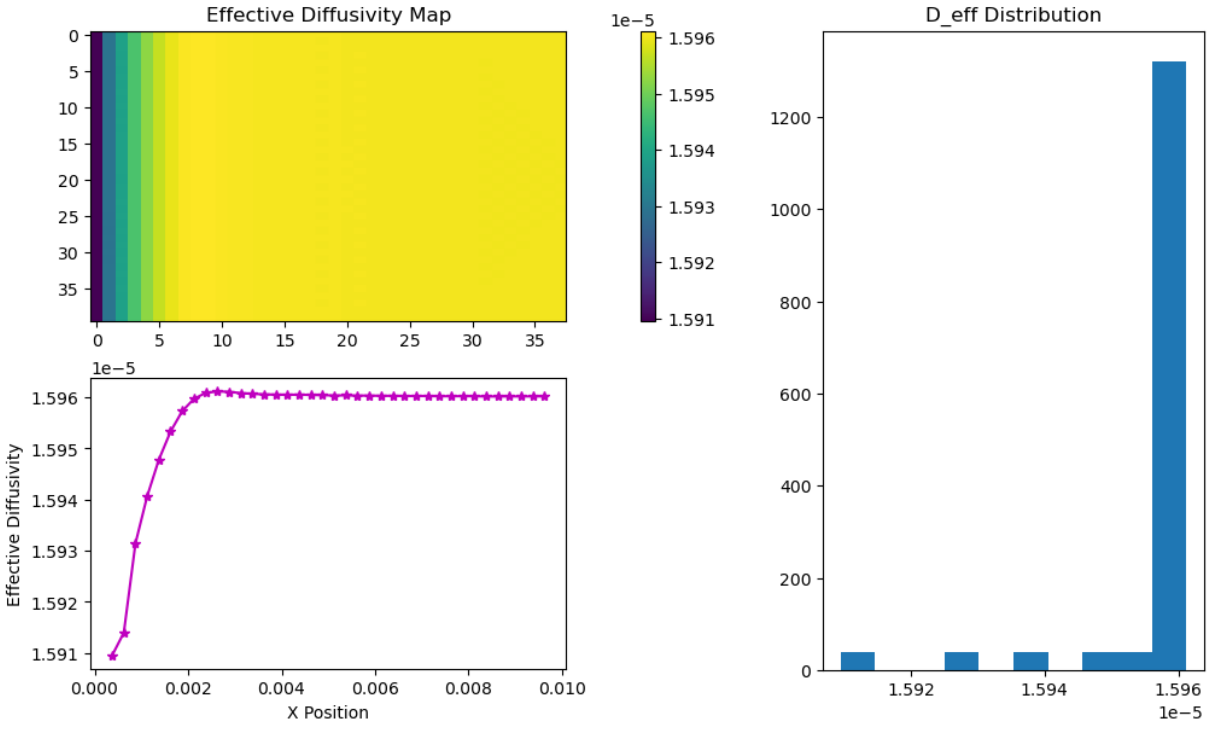
```
D_eff for pore 400 :     1.5961155664950516e-05
D_eff for pore 800 :     1.5960399913097988e-05
D_eff for pore 1200 :    1.596023455584744e-05
```



We can also see that the fitting works well for this case.

In [33]:
```
d_eff_array = []
d_eff_profile = []
x_arr = []
d_eff_profile_buffer = []
ctr0 = 0
```

```
for pore in full_pores:
    # 1D approximation
    idx_start = len(concentrations[:,pore][concentrations[:,pore] < 0*max(co
    idx_stop = len(concentrations[:,pore][concentrations[:,pore] < 0.9*max(c
    popt, pcov = curve_fit(f, times_to_save[idx_start:idx_stop,], concentrat
    d_eff_array.append(popt[0])
    d_eff_profile_buffer.append(popt[0])

    ctr0 += 1

    if ctr0 % pore_num == 9:
        d_eff_profile.append(np.mean(d_eff_profile_buffer))
        x_arr.append(pn['pore.coords'][pore][0])
        d_eff_profile_buffer = []

fig, ax = plt.subplots(2, 2, figsize=(10, 6), layout='constrained')
d_eff_map = np.rot90(np.reshape(d_eff_array, (pore_num-2,pore_num)))
pos = ax[0][0].imshow(d_eff_map)
ax[0][0].set_aspect('auto')
ax[0][0].set_title('Effective Diffusivity Map')
ax[1][0].plot(x_arr, d_eff_profile, 'm*-')
ax[1][0].set_xlabel('X Position')
ax[1][0].set_ylabel('Effective Diffusivity')
ax[0][1].axis('off')
ax[1][1].axis('off')
fig.colorbar(pos, ax=ax[0][0])
plt.subplot(1, 3, 3)
plt.hist(d_eff_array, 10)
plt.title('D_eff Distribution')
plt.locator_params(axis='x', nbins=5)
plt.show()
```



As we can see, the trend for the profile and distribution of the calculated effective

diffusivities follow the bulk trend as well, approaching a value of around $1.596 \cdot 10^{-5} \frac{m^2}{s}$.

Comparison with Steady State

These results are compared with the effective diffusivity for the same network obtained using the steady-state simulation.

In [37]:
```
air = op.phase.Air(network=pn)

f_diffusive = op.models.physics.diffusive_conductance.generic_diffusive
air.add_model(propname='throat.diffusive_conductance', model=f_diffusive)


fd = op.algorithms.FickianDiffusion(network=pn, phase=air)
inlet = pn.pores('left')
outlet = pn.pores('right')
fd.set_value_BC(pores=inlet, values=C_l)
fd.set_value_BC(pores=outlet, values=0)

fd.run();

rate_inlet = fd.rate(pores=inlet)[0]
L = pore_num*spacing
A = (pore_num * 1)*(spacing**2)
D_eff_ss = rate_inlet * L / (A * C_l)
print('D_eff_steady_state: ', "{0:.6E}".format(D_eff_ss))
```

```
D_eff_steady_state:  6.905584E-06
```

Here, we can see that the calculated D_{eff} is very different from the ones calculated from the transient simulation.

The equation used to calculate diffusivity for the steady-state simulation is the following:

$$D_{eff,ss} = J_{bulk} \frac{L}{\Delta C_{pore}} = \frac{N_A}{A_{bulk}} \frac{L}{\Delta C_{pore}}$$

As an idea, we can change the concentration in the equation to the bulk concentration, which would seem more consistent:

$$D_{eff,ss} = \frac{N_A}{A_{bulk}} \frac{L}{\Delta C_{bulk}}$$

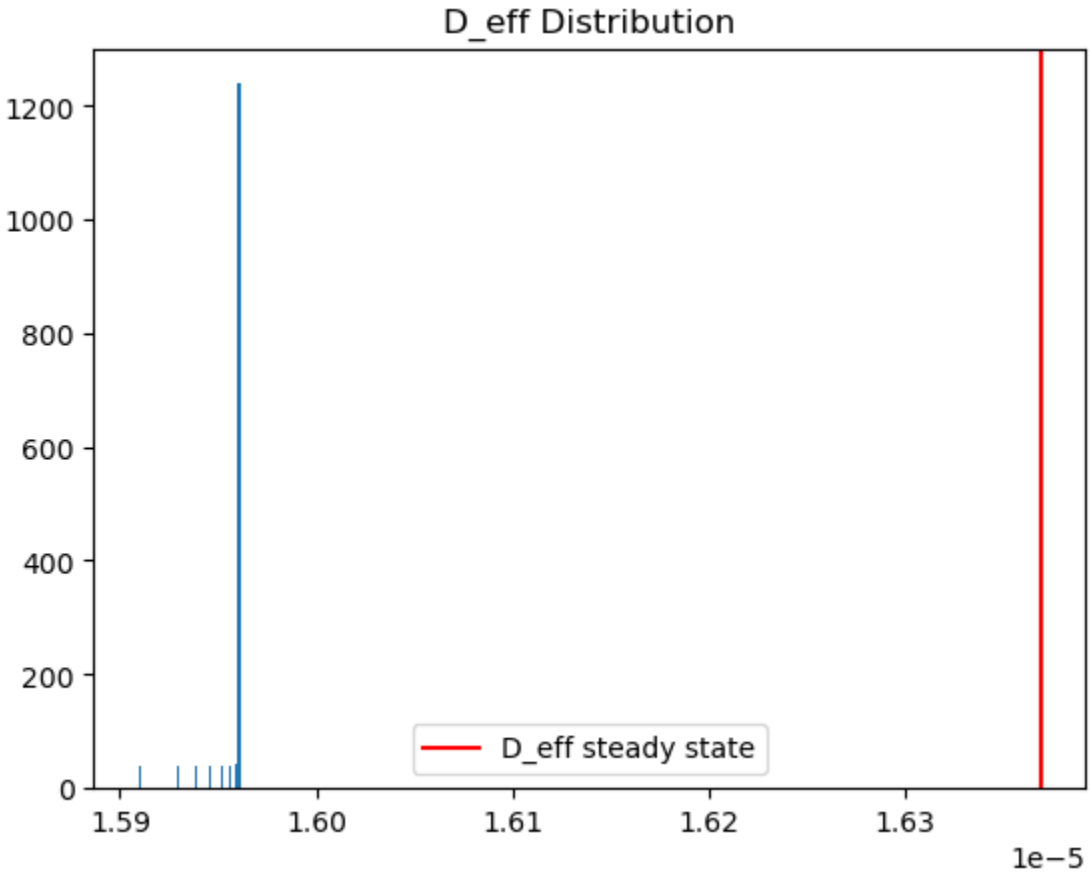
This is by converting the pore concentration to the bulk concentration using the following equation:

$$C_{bulk} = C_{pore} \frac{V_{pore}}{V_{bulk}}$$

In [39]:
```
V_cube = spacing**3
pn['pore.volume_ratio'] = pn['pore.volume'] / V_cube
```

```
D_eff_ss = rate_inlet * L / (A * np.sum(C_l*pn['pore.volume_ratio@left'])/(p
print('D_eff_steady_state: ', "{0:.6E}".format(D_eff_ss))

fig, ax = plt.subplots()
ax.hist(d_eff_array, 50)
ax.axvline(x=D_eff_ss, label='D_eff steady state', c='r')
ax.legend()
ax.set_title('D_eff Distribution')
plt.show()
```

```
D_eff_steady_state:  1.636879E-05
```



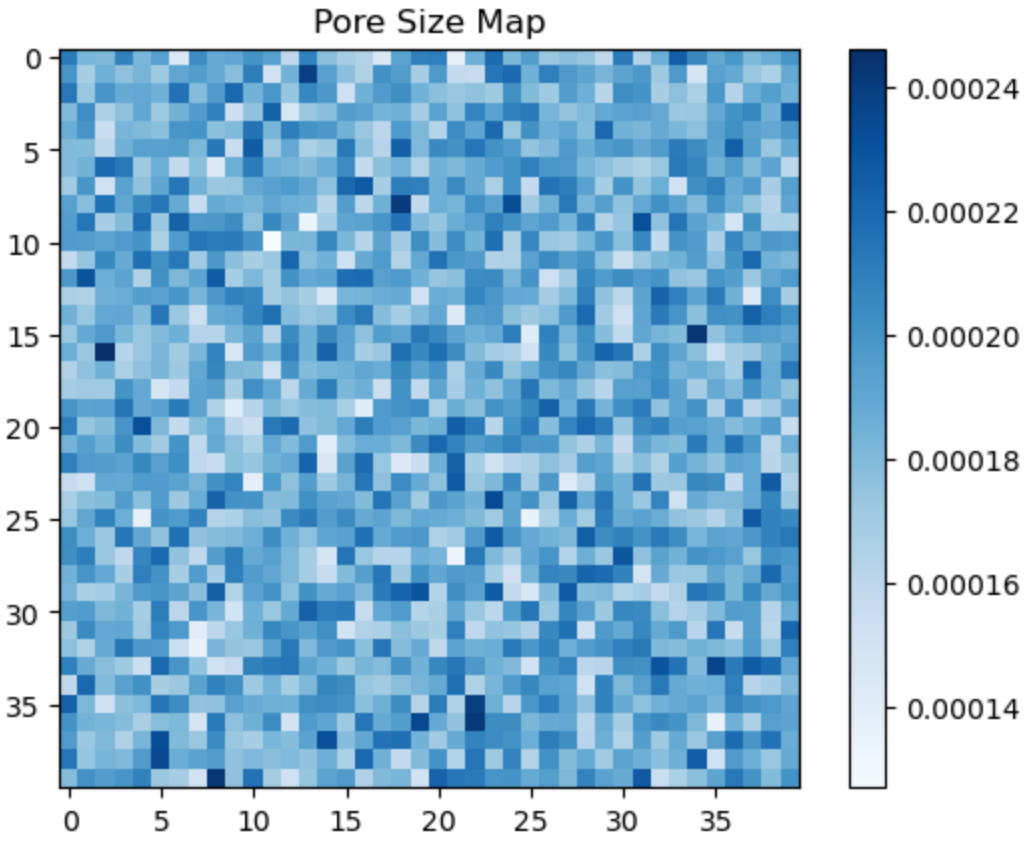
Interestingly, we can see that the effective diffusivity calculated here is much closer to the transient values, which means that the difference between the bulk and pore concentration probably plays a huge role in this discrepancy. This implies that converting the concentrations in either the steady-state method or the transient method is necessary to obtain the correct D_{eff} value. The question of which one is more correct will be investigated further in the future.

For the purpose of illustrating the next part, however, the transient solution will stay the same and the modified steady-state equation will be used.

Part 3: Simulating a Pore Network with a Size Distribution

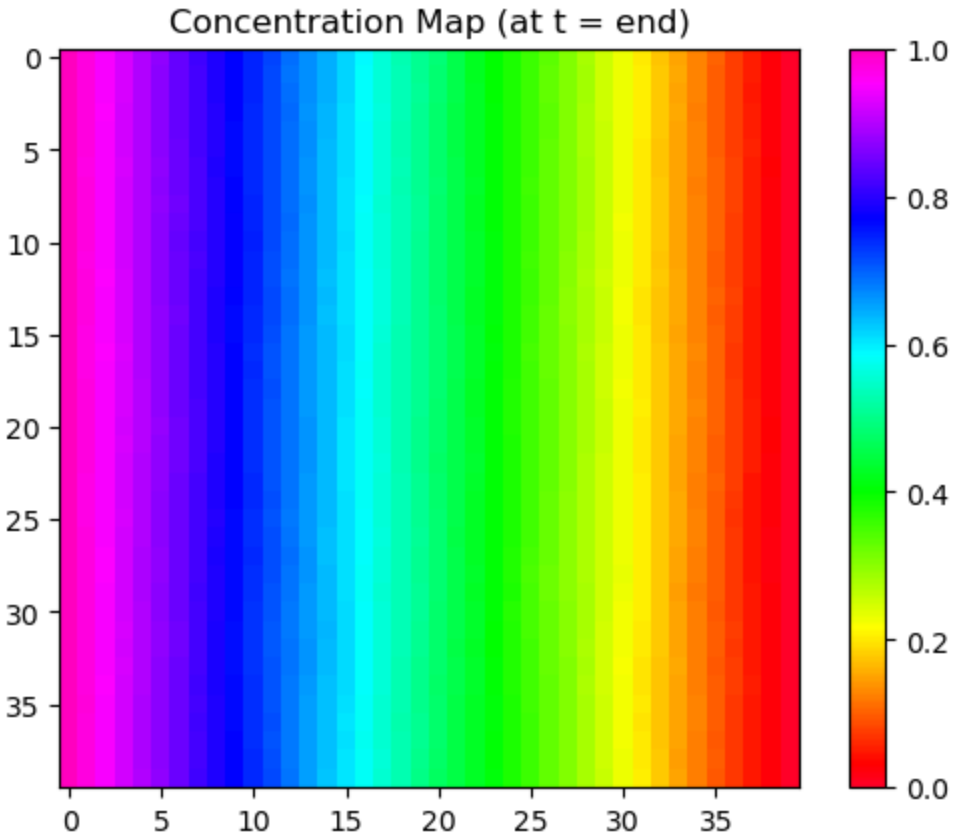
The same strategy as Parts 1 and 2 will be applied to simulate a pore network with a size distribution to see how it affects the effective diffusivities. The size distribution was assumed to be normal.

In [43]:
```
pore_diam = spacing*0.75
throat_diam_factor = 0.5

def pore_diameter(network):
    return pore_diam*np.random.normal(1, 0.1, size=(pore_num*pore_num))

def throat_length(network):
    return spacing - pn['pore.diameter'][pn['throat.conns'][:,0]]/2 - pn['po

def throat_diameter(network):
    return pore_diam*throat_diam_factor


pn = op.network.Cubic(shape=[pore_num, pore_num, 1], spacing = spacing)

f = op.models.collections.geometry.cubes_and_cuboids
pn.add_model_collection(f)
pn.add_model(propname='pore.diameter', model=pore_diameter)
pn.add_model(propname='throat.diameter', model=throat_diameter)
pn.add_model(propname='throat.length', model=throat_length)
pn.regenerate_models()


sizes = np.rot90(np.reshape(pn['pore.diameter'], (pore_num,pore_num)))
loc = plt.imshow(sizes, cmap='Blues')
plt.colorbar(loc)
plt.title('Pore Size Map')
plt.show()
```



In [44]:

```
air = op.phase.Air(network=pn)

f_diffusive = op.models.physics.diffusive_conductance.generic_diffusive
air.add_model(propname='throat.diffusive_conductance', model=f_diffusive)

tfd = op.algorithms.TransientFickianDiffusion(phase=air, network=pn)

tfd.set_BC(pores=pn.pores('left'), bctype='value', bcvalues=left_value)
tfd.set_BC(pores=pn.pores('right'), bctype='value', bcvalues=right_value)

tfd.run(x0=initial_concentration, tspan=time_span, saveat=times_to_save)
concentrations = np.transpose(tfd.soln['pore.concentration'])

concs = np.rot90(np.reshape(concentrations[-1].T, (pore_num,pore_num)))
loc = plt.imshow(concs, cmap='gist_rainbow')
plt.title('Concentration Map (at t = end)')
plt.colorbar(loc)
plt.show()
```

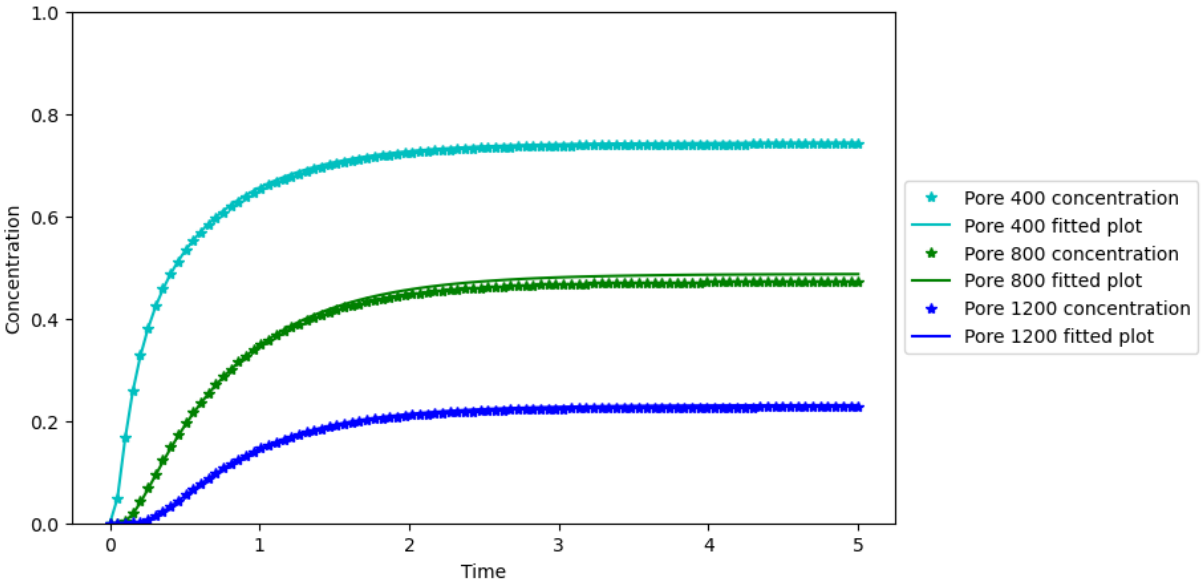


The concentration map now shows some variations in the y-direction due to the heterogeneity of the pore sizes, which means that the effective diffusivities calculated will have some variance for pores with the same x.

In [46]:
```
fig, ax = plt.subplots(figsize=(8, 5))
f = analytical_func
ctr0 = 0

for pore in sample_pores:
    label = 'Pore ' + str(pore) + ' concentration'
    ax.plot(times_to_save, concentrations[:,pore], data_colors[ctr0], label=

    # 1D approximation
    idx_start = len(concentrations[:,pore][concentrations[:,pore] < 0*max(co
    idx_stop = len(concentrations[:,pore][concentrations[:,pore] < 0.9*max(c
    popt, pcov = curve_fit(f, times_to_save[idx_start:idx_stop,], concentrat
    print('D_eff for pore',  + pore, ':', '\t', popt[0])

    # plot approximated plot
    label = 'Pore ' + str(pore) + ' fitted plot'
    ax.plot(times_to_save, analytical_func(times_to_save, popt[0]), curve_co

    ctr0 += 1


ax.set_ylabel('Concentration')
ax.set_xlabel('Time')
ax.set_ylim([0, 1])
```

```
ax.legend(loc='center left', bbox_to_anchor=(1, 0.5))
plt.show()
```

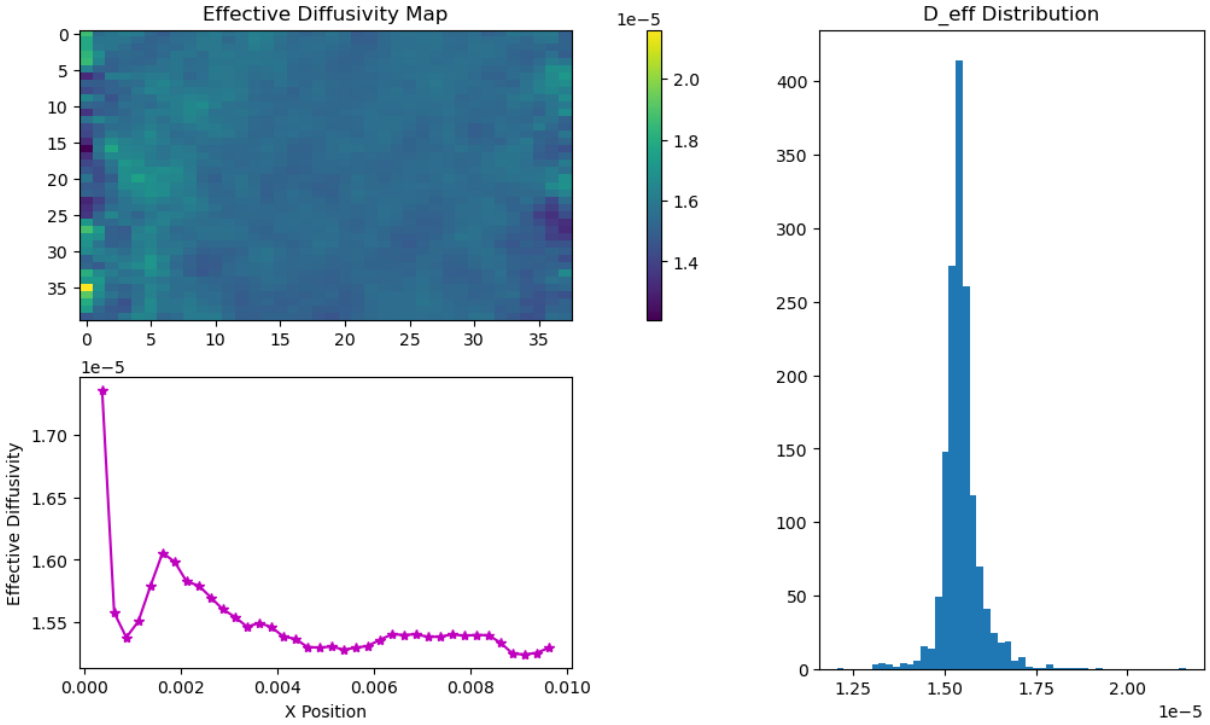
```
D_eff for pore 400 :     1.596908348204594e-05
D_eff for pore 800 :     1.4604599064936857e-05
D_eff for pore 1200 :    1.5321635852112023e-05
```



The fitting still seems to be effective, as seen in the 3-pore example above.

In [48]:
```
d_eff_array = []
d_eff_profile = []
x_arr = []
d_eff_profile_buffer = []
ctr0 = 0
for pore in full_pores:
    # 1D approximation
    idx_start = len(concentrations[:,pore][concentrations[:,pore] < 0*max(co
    idx_stop = len(concentrations[:,pore][concentrations[:,pore] < 0.9*max(c
    popt, pcov = curve_fit(f, times_to_save[idx_start:idx_stop,], concentrat
    d_eff_array.append(popt[0])
    d_eff_profile_buffer.append(popt[0])

    ctr0 += 1

    if ctr0 % pore_num == 9:
        d_eff_profile.append(np.mean(d_eff_profile_buffer))
        x_arr.append(pn['pore.coords'][pore][0])
        d_eff_profile_buffer = []

fig, ax = plt.subplots(2, 2, figsize=(10, 6), layout='constrained')
d_eff_map = np.rot90(np.reshape(d_eff_array, (pore_num-2,pore_num)))
pos = ax[0][0].imshow(d_eff_map)
ax[0][0].set_aspect('auto')
ax[0][0].set_title('Effective Diffusivity Map')
ax[1][0].plot(x_arr, d_eff_profile, 'm*-')
ax[1][0].set_xlabel('X Position')
ax[1][0].set_ylabel('Effective Diffusivity')
ax[0][1].axis('off')
```

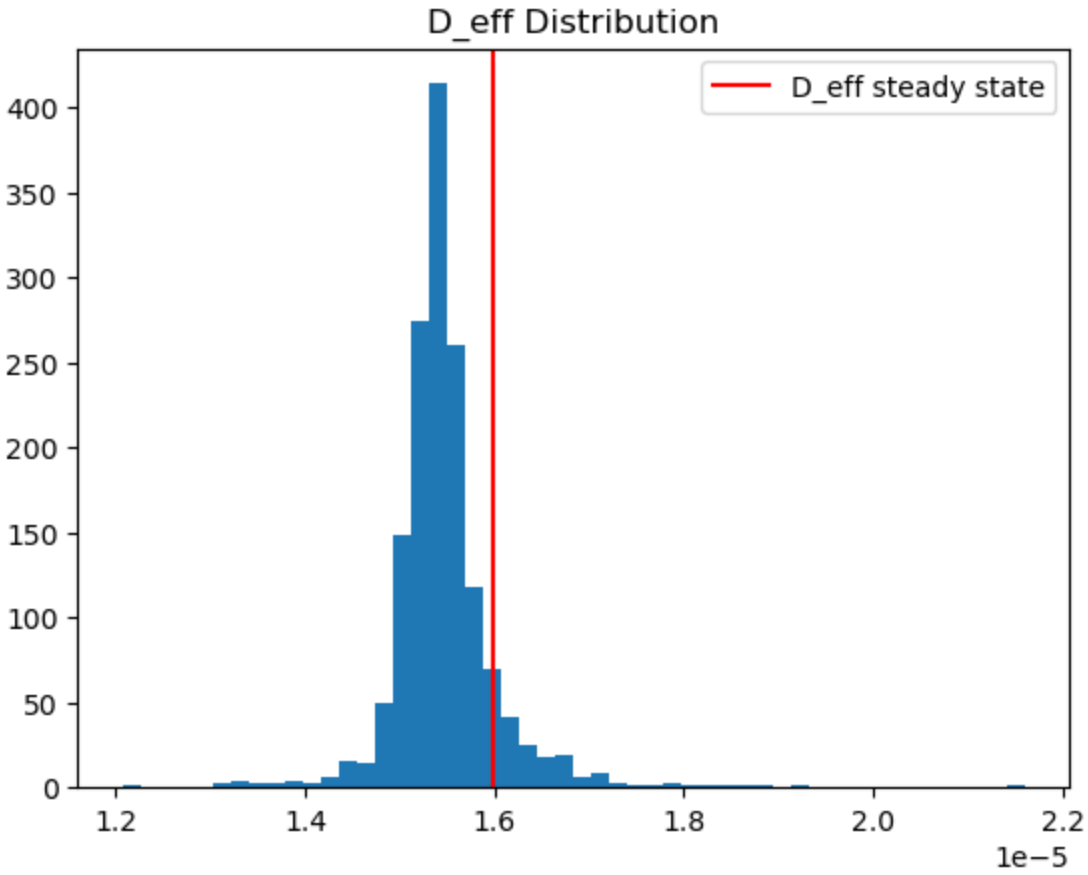
```
ax[1][1].axis('off')
fig.colorbar(pos, ax=ax[0][0])
plt.subplot(1, 3, 3)
plt.hist(d_eff_array, 50)
plt.title('D_eff Distribution')
plt.locator_params(axis='x', nbins=5)
plt.show()
```



Visualizing the effective diffusivity for each pore, we can immediately see that the values are now random and vary in both directions. The distribution does look to be normal, with most of the pores having a diffusivity of around $1.6 \cdot 10^{-5} \frac{m^2}{s}$. We can also see here that the effective diffusivities deviate more near the left boundary, as we saw in the previous cases. Again, this could possibly be attributed to fewer data points available for fitting due to the rapid rise in concentration.

In [50]:
```
V_cube = spacing**3
pn['pore.volume_ratio'] = pn['pore.volume'] / V_cube

D_eff_ss = rate_inlet * L / (A * np.sum(C_l*pn['pore.volume_ratio@left'])/(p
print('D_eff_steady_state: ', "{0:.6E}".format(D_eff_ss))


fig, ax = plt.subplots()
ax.hist(d_eff_array, 50)
ax.set_title('D_eff Distribution')
ax.axvline(x=D_eff_ss, label='D_eff steady state', c='r')
ax.legend()
plt.show()
```

```
D_eff_steady_state:  1.597858E-05
```



Comparing with the modified steady state value, we can see that it is also close to or around the range of the effective diffusivities for the transient case.

Conclusions

These results demonstrate the feasibility of determining the effective diffusivity in pore networks using OpenPNM's transient simulations, which offers an alternative and distinct approach to the steady-state technique. It was found that either the steady-state solution or the transient solution need volume-based scaling in order to match. This will be explored more in the future.

The presented method involves defining the initial and boundary conditions, determining an analytical solution, running a transient simulation, and fitting the analytical solution to the simulation results with the effective diffusivity as a fitting parameter. Thus, this method is **highly dependent on the chosen boundary and initial conditions**. Future work should then focus on testing out and comparing systems with different boundary and initial conditions to determine the most accurate method. More ideas for future work include:

- Fixing the numerical errors observed near the boundary pores.
- Varying the pore size distribution to more realistic forms, such as a Weibull distribution

- Expanding the effective diffusivity calculations to larger and more realistic pore networks

Overall, if these studies are expanded further, the results could help inform in the experimental and computational characterization of porous media, particularly developing more effective techniques for measuring diffusivity.

References

[1] J. Crank, The Mathematics of Diffusion, 2d Ed. Oxford, [Eng]: Clarendon Press, 1975.