Numerical Modeling of Cake Formation and Permeate Flux Decline in Membrane Filtration Using OpenFOAM

by

Jueming Hu

A Thesis Presented in Partial Fulfillment
of the Requirements for the Degree
Master of Science

Approved April 2018 by the
Graduate Supervisory Committee:

Marcus Herrmann, Chair
Huei-Ping Huang
Taewoo Lee

ARIZONA STATE UNIVERSITY

May 2018

ABSTRACT

Membrane filtration is an important technology in industry. In the past few decades, equations have been developed from experimental results to predict cake formation and permeate flux decline in the membrane filtration process. In the current work, the block of particles on membrane surface is achieved by setting surface flux on membrane surface zero. This approach is implemented for both microfiltration and nanofiltration using OpenFOAM. Moreover, a new method to deal with cake resistance for nanofiltration is introduced. Cake resistance is applied to both cake and membrane. To validate the new techniques, results of crossflow microfiltration are compared to theoretical results and results of two crossflow nanofiltration cases are compared to experimental data. In addition, the new techniques are applied to dead end filtration to observe the different structure of the cake and explore the effect of resistance on velocity profile.

ACKNOWLEDGMENTS

I would first like to thank my thesis advisor Prof. Marcus Herrmann. The door to Prof. Herrmann office was always open whenever I ran into a trouble spot or had a question about my research or writing. He guided me in the right direction when I was confused about what to do and doubted what had been done.

I would also like to acknowledge my lab mates Hooman Farsani, Sugajen Narayana Perumal and Erik Scougal. I am gratefully indebted to them for their very valuable help on this thesis. Hooman gave me a lot of useful advices and has been very helpful in encouraging me to be optimistic. In addition, Sugajen is the person who taught me how to use OpenFOAM. Moreover, Erik helped me understand numerical methods better. Without their passionate participation and input, this thesis could not have been successfully conducted.

Finally, I must express my loving thanks to my parents and friends. Without their encouragement and understanding it would have been impossible for me to finish this work.

TABLE OF CONTENTS

Page

CHAPTER Page

LIST OF TABLES

LIST OF FIGURES

Figure Page

CHAPTER 1

INTRODUCTION

1.1 Membrane Filtration

Membrane filtration is a pressure technology to separate solids from fluids by adding a medium through which only the fluid can pass. Its different modalities are microfiltration, ultrafiltration, nanofiltration and reverse osmosis. In detail, the operating pressure and the size of particles in the fluids of the four modalities are different. The differences in operating conditions are tabulated in Table 1.1 ("Industrial applications of membrane filtration", 2018).

Table 1.1 Characteristics of membrane processes

Process	Size of pore(μm)	Size of particle(μm)	Operating pressure(psi)	Application
Reverse Osmosis	0.0001	0.0001-0.001	400-1200	Dewatering(ions)
Nanofiltration	0.001	0.001-0.01	100-600	Desalination
Ultrafiltration	0.01	0.01-1	25-150	Purification of suspended solids, viruses, macromolecules
Microfiltration	1	1-10	20-150	Remove bacteria, suspended solids

Membrane filtration can also be classified according to the direction of the feed stream, namely dead end and crossflow membrane filtration. They are illustrated in the Figure 1.1.

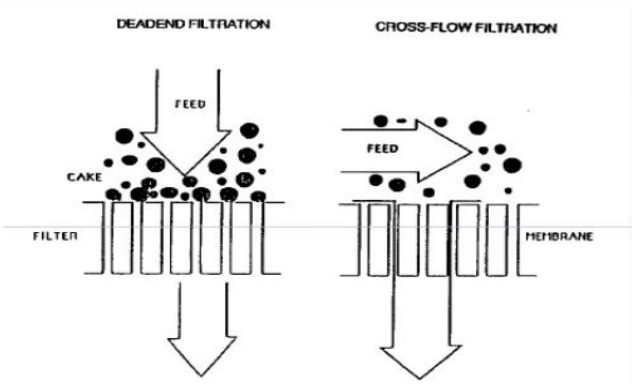

Figure 1.1 Dead end and crossflow membrane filtration (Christodoss, 2013)

Filtration by membrane technology is increasingly being used in productive processes of numerous industries. Its capacity to separate extracts and specific natural essences at low or ambient temperatures makes filtration by membranes a more profitable technology than other traditional methods. The food industry, with important specifications in the dairy and sugar, pharmaceutical, biotechnological and chemical sectors, is area in which filtration by membrane can be very useful.

During crossflow membrane filtration, suspended particles are transported to the membrane surface by permeate flow due to an imposed pressure drop. The layer with particles is named concentration polarization layer. Because of the finite size of colloidal particles, the particle concentration on the membrane surface reaches its maximum value after a short period of time, and a cake layer starts to form as shown in Figure 1.2. The resulting cake layer on the membrane surface increases the hydraulic resistance to permeate flow and, thus, reduces permeate flow through the membrane.

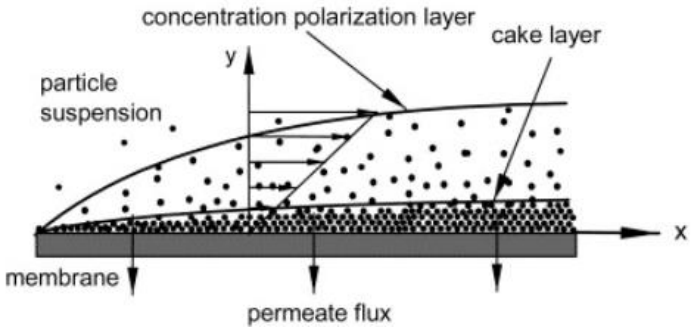

Figure 1.2 Concentration polarization and cake layer (Thiam Teik Wan, 2015)

1.2 Thesis Outline

The objective of this thesis is to model the membrane filtration process using appropriate numerical techniques. In detail, cake formation and permeate flux predicting models are implemented for microfiltration and nanofiltration respectively. The simulations are all conducted by OpenFOAM using a finite volume method. A procedure to deal with the flux of particles on the membrane surface is developed. The filtration process is simulated by solving solute and solvent in a coupled manner. Additionally, the total cake resistance is applied to both the cake and membrane in nanofiltration which results in a smaller specific cake resistance. To validate the new techniques, results of the microfiltration model are compared to theoretical results and the permeate flux of nanofiltration is compared to experimental data. At last, the methods are applied to dead end filtration to observe the different structure of the cake and the velocity profile as compared to crossflow filtration.

CHAPTER 2

THEORY OF PIMPLEFOAM TO SOLVE NAVIER-STOKES PROBLEM

2.1 Discretisation Procedure for the Navier-Stokes System in PimpleFOAM

PimpleFOAM is a solver in OpenFOAM for transient problems. In this section, the governing equations that PimpleFOAM solves are presented. The governing equations for unsteady and incompressible flow are as follows:

$$\nabla \cdot \mathbf{U} = 0 \tag{2.1}$$

$$\frac{\partial \mathbf{U}}{\partial t} + \nabla \cdot (\mathbf{U}\mathbf{U}) - \nabla \cdot (\nu \nabla \mathbf{U}) = -\nabla p \tag{2.2}$$

where p is the ratio of real pressure to density.

The discretized form of the continuity equation is as follows:

$$\nabla \cdot \mathbf{U} = \sum_f \mathbf{S} \cdot \mathbf{U}_f = 0 \tag{2.3}$$

where $\mathbf{S}$ is the outward-pointing face area vector, $\mathbf{U}_f$ is the face velocity. The nonlinear term in Eqn. (2.2) can be discretized as follows:

$$\begin{aligned} \nabla \cdot (\mathbf{U}\mathbf{U}) &= \sum_f \mathbf{S} \cdot (\mathbf{U})_f (\mathbf{U})_f \\ &= \sum_f F(\mathbf{U})_f \\ &= a_p \mathbf{U}_p + \sum_N a_N \mathbf{U}_N \end{aligned} \tag{2.4}$$

where a_P, a_N are functions of $\mathbf{U}$, $\mathbf{U}_P, \mathbf{U}_N$ are the face velocity. F represents the flux through the face:

$$F = \mathbf{S} \cdot (\mathbf{U})_f \tag{2.5}$$

The flux F needs to satisfy the continuity equation (Eqn. (2.1)).

In order to derive the pressure equation, a semi-discretized form of the momentum equation is used (Jasak, 1996):

$$a_P \mathbf{U}_P = \mathbf{H(U)} - \nabla p \tag{2.6}$$

$\mathbf{U}_P$ can be written as:

$$\mathbf{U}_P = \frac{\mathbf{H(U)}}{a_P} - \frac{1}{a_P}\nabla p \tag{2.7}$$

The **H(U)** term includes the matrix coefficients for all neighbor faces multiplied by the corresponding velocities and the source part of the transient term and all other source terms apart from the pressure gradient:

$$\mathbf{H(U)} = -\sum_N a_N \mathbf{U}_N + \frac{\mathbf{U}^0}{\Delta t} \tag{2.8}$$

Velocities on the cell face are expressed as the face interpolate:

$$(\mathbf{U}_P)_f = (\frac{\mathbf{H(U)}}{a_P})_f - (\frac{1}{a_P})_f (\nabla p)_f \tag{2.9}$$

This is used to calculate the face fluxes. The following form of the pressure equation is obtained by substituting Eqn. (2.9) into Eqn. (2.1).

$$\begin{aligned} \nabla \cdot (\frac{1}{a_P}\nabla p) &= \nabla \cdot (\frac{\mathbf{H(U)}}{a_P}) \\ &= \sum_f \mathbf{S} \cdot (\frac{\mathbf{H(U)}}{a_P})_f \end{aligned} \tag{2.10}$$

When Eqn. (2.10) is satisfied, the face fluxes are guaranteed to be conservative.

The final form of the discretized incompressible Navier-Stokes system is:

$$a_P \mathbf{U}_P = \mathbf{H(U)} - \sum_f \mathbf{S}(p)_f \tag{2.11}$$

$$\sum_f \mathbf{S} \cdot [(\frac{1}{a_P})_f (\nabla p)_f] = \sum_f \mathbf{S} \cdot (\frac{\mathbf{H(U)}}{a_P})_f \tag{2.12}$$

2.2 PimpleFOAM Algorithm

The principal of the algorithm is as follows: Within one time step, a solution can be found with under relaxation which means (Jasak, 1996):

$$p^{new} = p^{old} + \alpha_p (p^p - p^{old}) \tag{2.13}$$

$$\mathbf{U}^{new} = \mathbf{U}^{old} + \alpha_U (\mathbf{U}^p - \mathbf{U}^{old}) \tag{2.14}$$

In Eqn. (2.13), p^{new} is the approximation of the pressure field that is used in the next momentum predictor, p^{old} is the pressure field used in the momentum predictor, α_p is the pressure under-relaxation factor which is between 0 to 1, p^p is the solution of the pressure equation. Similarly, in Eqn. (2.14), $\mathbf{U}^{new}$ is the approximation of the velocity field that is used in the next pressure predictor, $\mathbf{U}^{old}$ is the velocity field used in the pressure predictor, α_U is the velocity under-relaxation factor which is between 0 to 1, $\mathbf{U}^p$ is the solution of the momentum equation. The solution is required to reach a defined tolerance criterion (Holtzman, 2017).

To find the solution within each time step, the so called outer corrector and inner corrector are needed. The number of outer correctors of the PimpleFOAM algorithm defines how many outer iterations to perform. In other words, it defines how many times the coupled velocity and pressure equations are performed before it is forced to move onto

the next time step, regardless of whether that time step has converged or not. However, the number of outer correctors does not mean that it performs all of these iterations. It simply iterates until time step convergence is reached and move on to the next time step. The criterion for time step convergence is defined as the absolute tolerance of the solver. By choosing a suitable outer corrector, it can be ensured that explicit parts of the equations are converged. Another thing to note about the outer corrector is that it solves with relaxation factors until the last outer iteration. Thus, if one runs for 50 iterations, then the algorithm runs the first 49 with relaxation factors and 1 without. The number of inner corrector is the number of times the pressure is corrected within an iteration which is suggested to set low as 1-3 in the PimpleFOAM guide. A large number of inner correctors rarely improve results and should conform to this recommendation. The stability of the PimpleFOAM algorithm can be improved by reducing the relaxation factor, by doing this convergence takes longer and therefore more outer correctors should be added. This is useful in protecting against unstable solutions where the maximum Courant number is likely to spike. There are limits to this and sometimes it is better to simply reduce the time step size or if auto time stepping reduces the maximum Courant number("CFD: PIMPLE Algorithm", 2018).

CHAPTER 3

FINITE VOLUME METHOD FOR CONVECTION DIFFUSION PROBLEM

In this chapter, the finite volume method for convection diffusion equations is introduced. In addition, for membrane filtration simulations, an upwind differencing scheme is chosen. The introduction and reason for the upwind differencing scheme are illustrated in Chapter 3.2.

3.1 Basics of Finite Volume Method for Convection Diffusion Equation

The convection diffusion equation is as follows:

$$\frac{\partial \phi}{\partial t} + \nabla \cdot (\mathbf{U}\phi) = \nabla \cdot (D \nabla \phi) \tag{3.1}$$

where ϕ is volume fraction of solute and D is the diffusion coefficient of solute. The discretisation of the convection term is obtained using Eqn. (3.2):

$$\begin{aligned} \int_V \nabla \cdot (\mathbf{U}\phi) dV &= \sum_f \mathbf{S} \cdot (\mathbf{U}\phi)_f \\ &= \sum_f \mathbf{S} \cdot (\mathbf{U})_f \phi_f \\ &= \sum_f F \phi_f \end{aligned} \tag{3.2}$$

where F is the face flux which is shown in Eqn. (2.5). The face value $\mathbf{U}_f$ is determined by an interpolation scheme. Additionally, the face value ϕ_f is calculated from the values at the cell centres, which are obtained using the convection differencing scheme. The discretisation of the diffusion term is obtained using Eqn. (3.3) (Jasak, 1996):

$$\int_V \nabla \cdot (D \nabla \phi) dV = \sum_f \mathbf{S} \cdot (D \nabla \phi)_f$$

$$= \sum_f (D)_f \mathbf{S} \cdot (\nabla \phi)_f \tag{3.3}$$

Since the mesh is orthogonal, vector d which points from the owner to its neighbor cell and face normal vector S are parallel.

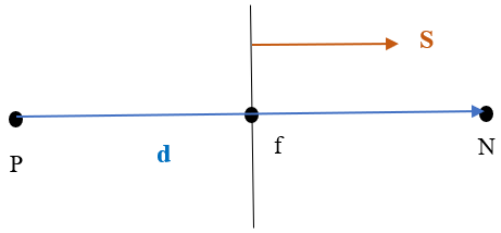

Figure 3.1 Vectors d and S on an orthogonal mesh

It is possible to use the following expression for the gradient of the scalar ϕ:

$$\mathbf{S} \cdot (\nabla \phi)_f = |\mathbf{S}| \frac{\phi_N - \phi_P}{|\mathbf{d}|} \tag{3.4}$$

Using Eqn. (3.4), the face gradient of ϕ can be calculated from the two values around the face.

3.2 Upwind Differencing Scheme

The role of the convection differencing scheme is to determine the value of ϕ on the face from the values in the cell centers. The upwind differencing scheme is shown in Figure 3.2:

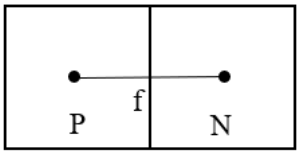

Figure 3.2 Upwind differencing scheme with face normal velocity

When the normal velocity $\mathbf{U}_P$ in Figure 3.2 at cell P points from P to N, the volume fraction at face f is equal to volume fraction at cell P. By contrast, the volume fraction at face f is equal to volume fraction at cell N when the velocity $\mathbf{U}_P$ at cell P points from N to P. In other words, the volume fraction at face f is equal to that at the upstream cell center.

There are mainly two reasons to choose upwind differencing scheme. Firstly, the scheme accounts the direction of flow. In this way, volume fraction on the membrane surface is equal to the value at the first cell center above the membrane. This is consistent with the idea that membrane surface rejects all the coming particles when calculating convection term. Secondly, the first order upwind scheme is bounded which means it gives physically bounded results. In other words, the volume fraction is always between 0 and 1.

CHAPTER 4

METHODOLOGY

This chapter introduces the techniques to simulate the crossflow membrane filtration process. Pure water is considered as the solvent and particles are solute. Membrane surface prevents particles from passing through it. This thesis focuses on predicting concentration polarization, cake formation and permeate flux for microfiltration and nanofiltration. Different strategies and conditions are applied to microfiltration and nanofiltration. In detail, a modified diffusion coefficient and a method to deal with cake resistance are introduced separately for each type of filtration. Additionally, the way to simulate the blockage of the solute on the membrane surface is the same for all filtration models and is illustrated in the part on microfiltration.

4.1 Cake Predicting Model for Microfiltration

In microfiltration, a shear induced diffusion coefficient is used and the cake resistance is applied directly to the cake cells. Other details about the physical property of fluid and the algorithm are introduced in Chapter 4.1.3.

4.1.1 Shear Induced Diffusion Coefficient

Shear induced diffusion dominates when particle sizes are larger than 0.5 μm (Cho, Kim, Moon & Kwon, 2006). In the microfiltration model, since the particle radius is around 2.5 μm, a shear induced diffusion coefficient is used. The shear-induced hydrodynamic diffusion coefficient, D , has the following form (Romero & Davis, 1988):

$$D = 0.33\dot{\gamma}a^2\phi^2(1+0.5e^{8.8\phi}) \tag{4.1}$$

where $\dot{\gamma} = \frac{du}{dy}$ is the local shear rate in the layer and a is the particle radius. It can be seen from Eqn. (4.1) that the shear induced diffusion coefficient is dependent on velocity and volume fraction. As a result, the diffusion coefficient is a scalar field instead of a constant and needs to update at every time step.

4.1.2 Cake Resistance of Microfiltration

In the microfiltration model, once the volume fraction at a cell center exceeds 0.6, a cake is formed. There is additional resistance offered by the cake. The value of the specific cake resistance is obtained through the Carman-Kozeny relation:

$$r_c = \frac{C(1-\varepsilon_c)^2 S_c^{\,2}}{\varepsilon_c^{\,3}} \tag{4.2}$$

where the constant C is 5, ε_c is the porosity of the randomly packed cake layer equal to $1-\phi_{\max}$ and S_c is the specific surface, $S_c = \frac{3}{a}$. The unit of specific cake resistance is m^{-2}. The cake resistance is only applied to the cake cells. Another important parameter is cake resistance, R_c, whose unit is m^{-1} .

$$R_c = r_c \delta_c \tag{4.3}$$

where δ_c is cake thickness. R_c means the total resistance to the fluid caused by the cake while r_c is a thickness averaged resistance and is a physical characteristic of cake.

To take cake resistance into account when solving hydrodynamics, cake resistance is transferred to a source term in the momentum equation. The new momentum equation is as follows:

$$\frac{\partial \mathbf{U}}{\partial t} + \nabla \cdot (\mathbf{UU}) - \nabla \cdot (\nu \nabla \mathbf{U}) = -\nabla p - \nu r_c \mathbf{U} \tag{4.4}$$

4.1.3 Implementation of Model

There are mainly two parts in the simulation: Navier-Stokes equation and convection diffusion equation of volume fraction. The coupling from the Navier-Stokes equation to convection diffusion equation is done by using the velocity which is used to calculate convection term of volume fraction. The coupling from the convection diffusion equation to the Navier-Stokes equation is achieved by kinematic viscosity, ν, which is a function of concentration (Paipuri, 2014). The functional form of $\nu(\phi)$ was determined empirically from viscosity measurements of suspensions of rigid spheres in a Couette viscometer (Leighton & Acrivos, 1986):

$$\nu = \nu_0 (1 + 1.5 \frac{\phi}{1 - \frac{\phi}{\phi_c}})^2 \tag{4.5}$$

where ν_0 is the viscosity of the particle free fluid and the solvent is water in all models in this thesis, ϕ_c is the value of volume fraction larger than which cake is formed. After the cake is formed, cake is treated as an extra porous medium. As a result, the viscosity inside the cake is no longer a function of concentration and is equal to the viscosity of the particle free fluid.

The membrane resistance, R_m, is transferred to a source term for the momentum equation similar to the cake resistance. The specific membrane resistance, r_m, is calculated in Eqn. (4.6):

$$r_m = \frac{R_m}{\delta_m} \tag{4.6}$$

The unit of r_m is m^{-2}. The momentum equation including membrane resistance is shown in Eqn. (4.7):

$$\frac{\partial \mathbf{U}}{\partial t} + \nabla \cdot (\mathbf{U}\mathbf{U}) - \nabla \cdot (\nu \nabla \mathbf{U}) = -\nabla p - \nu r_m \mathbf{U} \tag{4.7}$$

Both water and particles are considered flowing at the same velocity. Thus there is only one velocity in the simulation. After solving the Navier-Stokes system, surface flux and surface diffusion coefficients are used in the convection diffusion equation. To make surface flux on the membrane surface equal to the flux at the above cell center, upwind scheme is used to calculate flux. To prevent particles passing through the membrane surface, both surface flux and surface diffusion coefficient are set to zero. In OpenFOAM, the cell centers are numbered from left to right and bottom to top. Each face has an owner cell center and a neighbor cell center. The owner cell is the cell with the smaller number. Since the membrane surface is horizontal, when the owner cell of a face has a membrane resistance while there is no membrane resistance at the neighbour cell, the face is identified within the membrane surface. Before the convection diffusion equation, the velocity is calculated and at the beginning of next time step the velocity is updated according to the new pressure profile. So it is safe to directly set the surface flux on the membrane surface to zero.

After the cake is formed, the cake cell has reached its maximum capacity of particles which means no more particles can come into the cell. To achieve this, the cell faces are treated as walls. To express this mathematically, the surface flux and diffusion

coefficient on all faces of the cake cell are set to zero. In this way, the volume fraction calculated from the convection diffusion equation does not increase.

In terms of boundary conditions, a no-slip condition is used at walls. In addition, there are one inlet and two outlets in the following crossflow models. Pressure at the bottom outlet is set to zero and a parabolic velocity profile is set for the inlet which is calculated using Eqn. (4.8).

$$U_x(y) = \frac{1}{2\mu}\frac{dP}{dx}(H^2 - y^2) \tag{4.8}$$

where η is the dynamic viscosity, P is the pressure difference in the x direction between inlet and outlet, H is half of the width of up channel and y is the distance away from centerline. Eqn. (4.8) is for a rectangular channel flow. Moreover, the right outlet has a fixed value of pressure based on a given transmembrane pressure which is described in detail in Chapter 5.1.1. In OpenFOAM, both pressure and velocity of the boundary surface need to set. The velocity or pressure which is not set to a fixed value at inlet or outlet is set zero gradient.

The entire simulation process can be summarized using the following algorithm:

Algorithm 1: Cake predicting model for microfiltration

Initialize the flow solver (velocity, pressure, ϕ, ν, r_m, r_c, D)

Set boundary conditions

While (time $\leq$ end time) do

Compute r_m

Solve Navier-Stokes equation

While (a parabolic flow is reached in the up channel) do

If the parabolic flow was just reached, set initial ϕ for the upper channel

Compute D

Interpolate D to faces to get D_f

Set both F and D_f on membrane surface and faces of cake cell zero

Solve convection diffusion equation

Compute r_c

Update ν

End

End

At the beginning, the simulation is solving for the flow only with membrane resistance. After a parabolic flow is reached in the upper channel, the convection diffusion equation starts to solve. This is to avoid start error in OpenFOAM. Also in experiment inlet is away from the membrane to wait for fully developed flow.

4.2 Cake predicting model for Nanofiltration

In nanofiltration, a Brownian diffusion coefficient is used and the total cake resistance is applied to both cake and membrane. Other details about the physical property of the fluid and the algorithm are introduced in Chapter 4.2.3.

4.2.1 Brownian Diffusion Coefficient

Brownian diffusion dominates when particle sizes are smaller than 0.5 μm (Cho, Kim, Moon & Kwon, 2006). In the nanofiltration model, since the particle radius is on the order of nanometers, a Brownian diffusion coefficient is used. Brownian diffusion comes from Brownian motion. The diffusion coefficient for a single particle is given by the Stokes-Einstein equation:

$$D = \frac{kT}{6\pi\eta a} \tag{4.9}$$

where k is the Boltzman constant which is 1.38×10^{-23} and T is the absolute temperature. It is shown that the variation of diffusion with particle volume fraction is not significant in this model (Kim, Marion, Jeong & Hoek, 2006). As a result, the diffusion coefficient at a cell center can be obtained by using Eqn. (4.9).

4.2.2 Cake Resistance of Nanofiltration

In the nanofiltration model, once the volume fraction at a cell center exceeds 0.63, the cake is formed. The value of specific cake resistance is also obtained through Eqn. (4.2). It is around $10^{17}\ m^{-2}$. Then when using the Eqn. (4.4), the result of permeate flux is extremely smaller than expected. In this way, it is hard to see an increase of the volume fraction and cake growth.

The new technique to express the cake resistance is applying the total cake resistance to both cake and membrane. The new specific cake resistance is marked as r_c' which is obtained using Eqn. (4.10):

$$r_c' = \frac{r_c \delta_c}{\delta_c + \delta_m} \tag{4.10}$$

r_c' is applied to the cake cell and the cells inside the membrane which have the same tangential coordinate as the cake cell. δ_c is the cake thickness and its increases with the growth of the cake. To ensure the total cake resistance does not change, the specific membrane resistance needs to be updated by Eqn. (4.11):

$$r_m' = r_m + r_c' \tag{4.11}$$

The momentum equation with both cake and membrane resistance in nanofiltration is shown in Eqn. (4.12):

$$\frac{\partial \mathbf{U}}{\partial t} + \nabla \cdot (\mathbf{U}\mathbf{U}) - \nabla \cdot (\nu \nabla \mathbf{U}) = -\nabla p - \nu r_c' \mathbf{U} - \nu r_m' \mathbf{U} \tag{4.12}$$

To validate the special treatment of cake resistance, it is applied to two different models. The results are shown in Chapter 5.2.

4.2.3 Implementation of Model

The simulation process for nanofiltration is almost the same as microfiltration. The idea to set the flux and diffusion coefficient on the membrane surface and faces of cake cells is exactly the same. The main differences are the technique to apply the new specific cake resistance, membrane resistance and the way to calculate the diffusion coefficient. Additionally, in Eqn. (4.5) when the concentration is 0.62, $\frac{\phi}{\phi_c}$ is very close to 1. To avoid viscosity going to infinity, Eqn. (4.5) is used only when volume fraction is smaller than 0.62.

The entire simulation process can be summarized using the following algorithm:

Algorithm 2: Cake predicting model for nanofiltration

Initialize the flow solver (velocity, pressure, ϕ, ν, r_m', r_c', D)

Set boundary conditions

While (time $\leq$ end time) do

Compute r_m'

Solve Navier-Stokes equation

While (a parabolic flow is reached in the up channel) do

If the parabolic flow was just reached, set initial ϕ for the upper channel

Compute D

Interpolate D to faces to get D_f

Set both F and D_f on membrane surface and faces of cake cell zero

Solve convection diffusion equation

Compute r_c'

Update ν

End

End

CHAPTER 5

RESULTS AND DISCUSSION

This chapter presents results of different models discussed in Chapter 4. It starts with the results of microfiltration with four different applied pressures. Phenomena of concentration polarization, cake formation and flux decline are discussed. Then two simulations of nanofiltration are given. Numerical results of nanofiltration models are compared to experimental data to validate the technique of dealing with cake resistance for nanofiltration.

5.1 Microfiltrtaion Results

This section describes microfiltration results. The geometry of the filtration device and exact values of physical properties of the solvent and solute are included. The effect of transmembrane pressure applied to the device on permeate flux and cake layer is studied.

5.1.1 Model Description

The geometry of the filtration device is shown in Figure 5.1. There are two outlets. Flow entries from the left side. Part of the flow leaves the device at the right side and part leaves the device through bottom. The membrane location is indicated by dashed lines. The wall location is indicated by bold line. The unit of lengths in Figure 5.1 is μm.

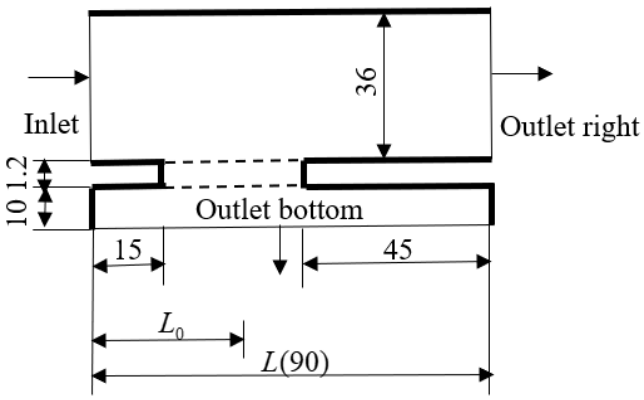

Figure 5.1 Geometry of microfiltration device

Transmembrane pressure (TMP) is defined as pressure difference between either side of membrane at its half way point.

$$TMP = p_{in} - \frac{L_0}{L}(p_{in} - p_{out}) - p_{per} \tag{5.1}$$

In Eqn. (5.1), p_{in} is pressure at the inlet, L_0 is the distance of the middle of membrane to inlet, L is the length of the channel, p_{out} is the pressure at the right outlet, p_{per} is the pressure at the bottom outlet. The radius of solute in this model is 2.5 μm and therefore the specific cake resistance can be calculated using Eqn. (4.2) with the porosity of the randomly packed cake layer at 0.4. A pressure drop between the inlet and the right outlet of 270 Pa is used. This results in a crossflow with a parabolic velocity profile calculated using Eqn. (4.8). A fixed inlet concentration of 0.05 is used which is the same as the initial condition for the upper channel. The particle free medium is considered as water. Therefore ν_0 in Eqn. (4.5) is $10^{-6} m^2 / s$. For this filtration device the resistance of the membrane R_m is $1.62 \times 10^8 m^{-1}$.

5.1.2 Results

For microfiltration, simulations with four TMPs were conducted using one geometry. Other parameters are keep constant to analyze the effect of pressure on the cake layer and permeate flux.

The four TMPs are 1180, 1780, 2980, 4180 Pa. The model is run at each TMP until steady state is reached. Among the four cases, only the pressure at the right outlet is changed. To test the approach to block particles on the membrane surface, one needs to capture the increase in concentration. So for microfiltration methods and equations are solved in non-dimensional form.

Concentration distribution at 0.02s is shown in Figure 5.2.

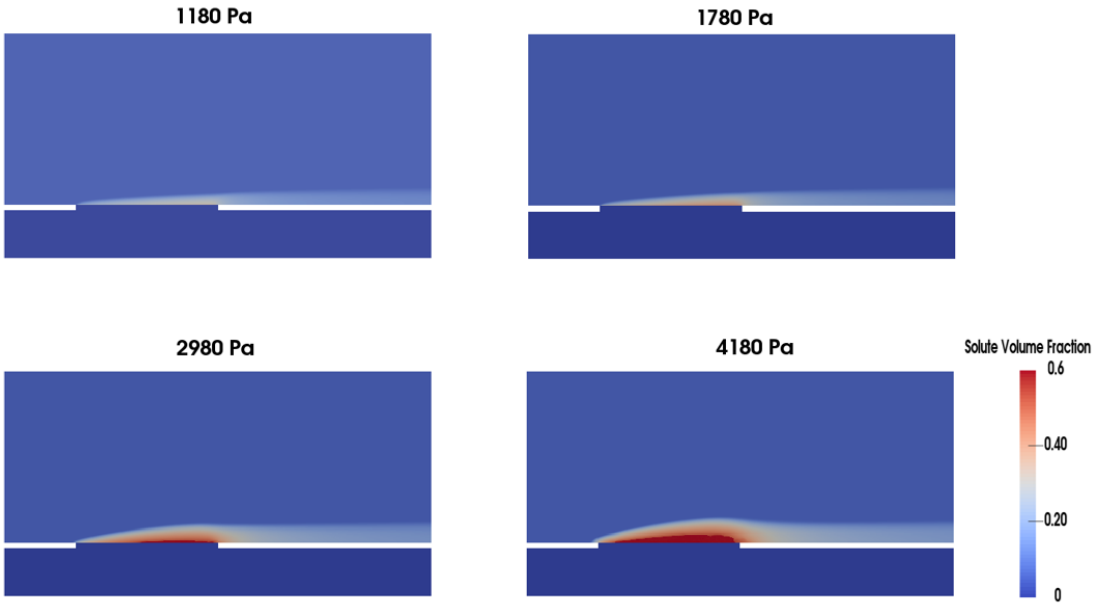

Figure 5.2 Concentration distribution at different TMPs in microfiltration

From the color legend, it can be concluded that firstly when TMPs are 1180 Pa and 1780 Pa, there is no cake layer formed. Only concentration polarization is built up. Secondly, the cake at TMP of 4180 Pa is thicker than the one at 2980 Pa. Concentration polarization

is the first stage of cake formation. With higher TMP, the velocity coming to the membrane surface is larger which means within unit time, the flow brings more particles toward the membrane surface. Moreover, particles have the ability to diffuse and can be blown away by the tangential flow. So the increase in volume fraction depends on the speed of blocking particles over the membrane and the speed of particle blowing away and diffusing away. In the four cases, the tangential velocity of crossflow is similar and the diffusion coefficient is also the same at a same concentration. As a result, concentration over the membrane increases faster and more with larger permeate flux which is caused by larger TMP. Another important point to make is that the thickest concentration polarization or cake in each case is at the end of the membrane. This is possibly due to a higher convective force by the fluid on the solute particles in the crossflow direction (Paipuri, 2014). Also the formation of the cake can be observed during the simulation to start to form from the end of membrane.

The other important result is a permeate flux decline. Permeate flux is the flux coming out of the membrane. In the simulations, the average velocity at the first cell center normal to the membrane is taken as the permeate flux. The average velocity can be calculated using Eqn. (5.3):

$$v_w = \frac{\int_0^{L_m} U_y(x)dx}{L_m} \tag{5.3}$$

where L_m is the length of the membrane. The initial and final permeate flux at different TMP are shown in Figure 5.3.

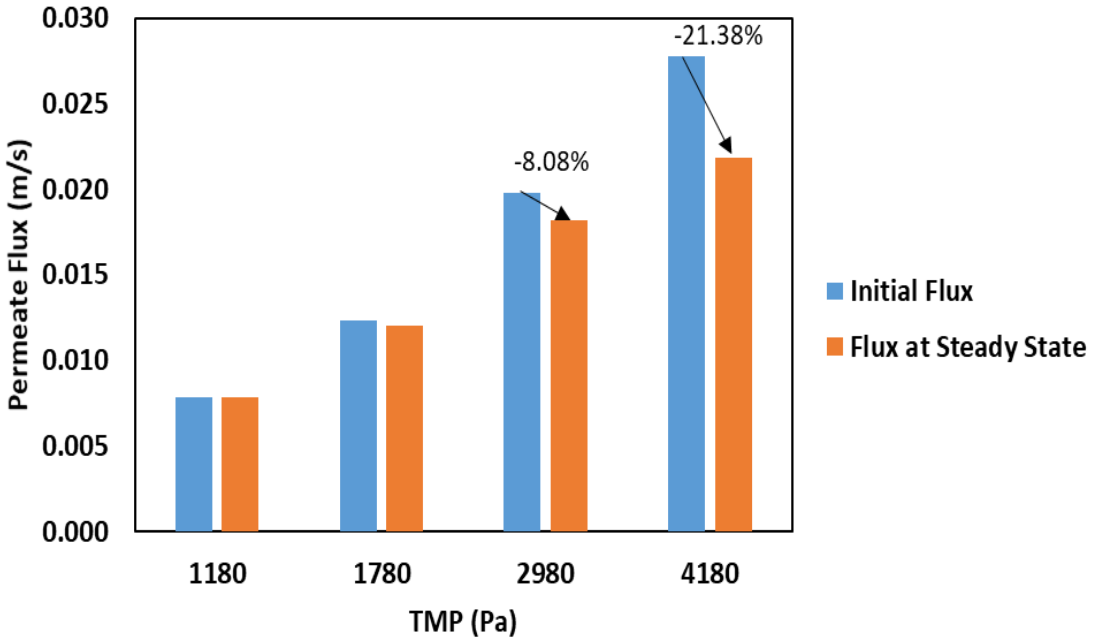

Figure 5.3 Initial and final permeate flux at different TMP

It can be seen there is a nonnegligible decrease in permeate flux with TMPs at 2980 Pa and 4180 Pa. This is due to the additional resistance of the cake. This can also explain why the final permeate flux is almost same as initial flux with TMPs at 1180 Pa and 1780 Pa. The value of the final permeate flux is compared to the theoretical data using Eqn. (5.4) (Kromkamp et al., 2005):

$$v_w = 0.072\dot{\gamma}_{wall}\left(\frac{\phi_c a^4}{\phi_b L_m}\right)^{1/3} \tag{5.4}$$

where $\dot{\gamma}_{wall}$ is the nominal shear rate at the wall, ϕ_b is the bulk particle volume fraction and ϕ_c is the maximal volume fraction. The results are shown in Table 5.1.

Table 5.1 Permeate flux in microfiltration

TMP	$\dot{\gamma}_{wall}(10^4\ s^{-1})$	v_w (m/s, calculated)	v_w (m/s, simulated)	error
1180	5.4	-0.0097	-0.008	17.5%
1780	6.2	-0.011	-0.012	9.1%
2980	7	-0.013	-0.017	30.7%
4180	9	-0.016	-0.021	31.25%

$\dot{\gamma}_{wall}$ is obtained from the simulation results. When a cake is formed, the simulation results of the permeate flux are larger than theoretical data. The possible reason is that the flow is not steady state yet and the cake is still growing. At steady state, the transport of particles from the bulk to the concentration polarization layer by permeate flow is balanced by the transport of particles due to crossflow (Hong, Faibish & Elimelech, 1997). When doing the simulation, the case is stopped by the end time which comes from a plot of flux as a function of time in Kromkamp's work (Kromkamp et al., 2005). In Kromkamp's work (Kromkamp et al., 2005), the velocity has already reached steady state at time 0.02 s. Thus, the end time is set to 0.2. But the length of the membrane in the current work is smaller than in Kromkamp's model (Kromkamp et al., 2005). Another possible reason is that the velocity at the cell center is taken as the permeate flux while the exact permeate flux is the velocity at the membrane. If using an interpolation method to compute the velocity at the membrane surface, the permeate flux from the simulation would be smaller than what is shown in Table 5.1 since the velocity inside the membrane region is smaller than that above the membrane due to the membrane resistance.

5.2 Nanofiltration Results

This section describes two models of nanofiltration. The approach to represent cake resistance for nanofiltration is applied to different cases for validation. Both cases have experimental data to compare.

5.2.1 Simulation of Suhan Kim's Model

Work in this chapter follows Suhan Kim's model (Kim, Marion, Jeong & Hoek, 2006).

5.2.1.1Model Description

The geometry of the filtration device is shown in Figure 5.4. The fluid flows in the same direction as in the microfiltration. Part of the flow leaves the device at the right side and part leaves the device through bottom. The membrane location is indicated by dashed lines. The wall location is indicated by bold line. The unit of lengths in Figure 5.4 is cm. The widths of upper and bottom channel are the same as in Kim's model. However, the length of the channel and membrane are shorter to save computational cost but this does not have impact on the results. Moreover, in Paipuri's work (Paipuri, 2014), the length also has no effect for this case. To get stable results, the inlet and right outlet are set away from the membrane.

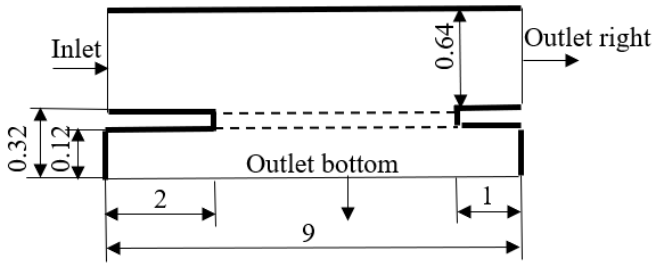

Figure 5.4 Geometry of Suhan Kim's model

The particle radius in this model is $60\,nm$ and the temperature is $20^{\circ}C$, therefore the diffusion coefficient can be calculated using Eqn. (4.9). The thickness of the membrane is not given in Kim's model (Kim, Marion, Jeong & Hoek, 2006) and is set to be about half the width of the bottom channel. From the experimental data (Kim, Marion, Jeong & Hoek, 2006), the volume fraction of randomly packed cake layer is 0.67 so the porosity, ε_c is set 0.33. Specific cake resistance, r_c' can be calculated using Eqn. (4.10) and is $1.6\times10^{17}\,m^{-2}$. When the cake is formed, the specific membrane resistance in the membrane cells below the cake is updated through Eqn. (4.11).

In this model, TMP is 70 kPa and the crossflow velocity is $6\,cm/s$. A fixed inlet concentration of 4×10^{-5} is used which is the same as the initial condition for the upper channel. For this filtration device the resistance of the membrane R_m is $8.11\times10^{11}\,m^{-1}$.

Since the thickness of the cake formed in the experiment is in the order of a few tenths of μm, the vertical mesh size inside the cake region which is predicted to be no thicker than $30\,\mu m$ is set to be $0.75\,\mu m$ to capture the cake. Outside the cake region and above the membrane, graded mesh where cell-to-cell expansion ratio is about 1.1 is used. To see the meshes clearly, meshes around membrane surface are zoomed in to show in

Figure 5.5.

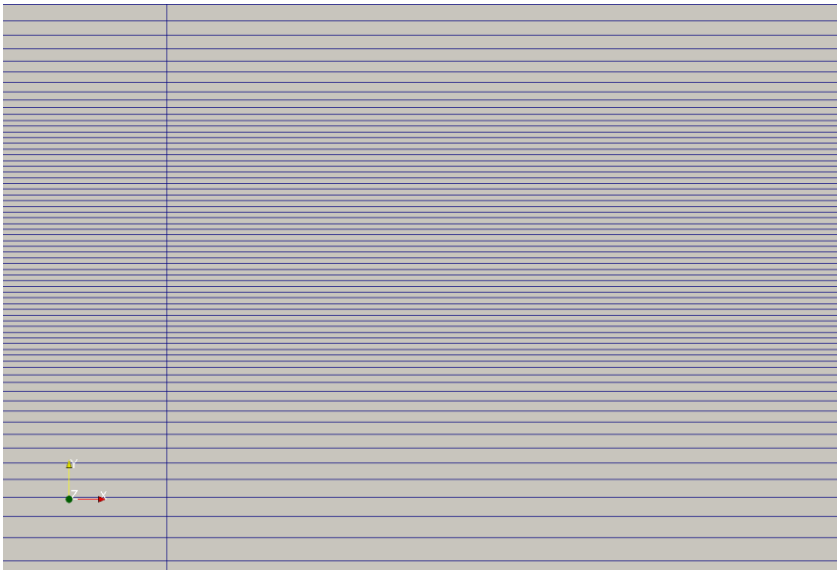

Figure 5.5 Meshes around membrane surface

5.2.1.2 Results

In this model, there are mainly two parts of results, permeate flux decline and cake layer formation. The value of TMP is given in the experimental data. In the simulation, the inlet velocity is set as the crossflow velocity and pressures at the two outlets are set. The pressures are calculated using Eqn. (5.1).

The plot of permeate flux versus time is shown in Figure 5.6.

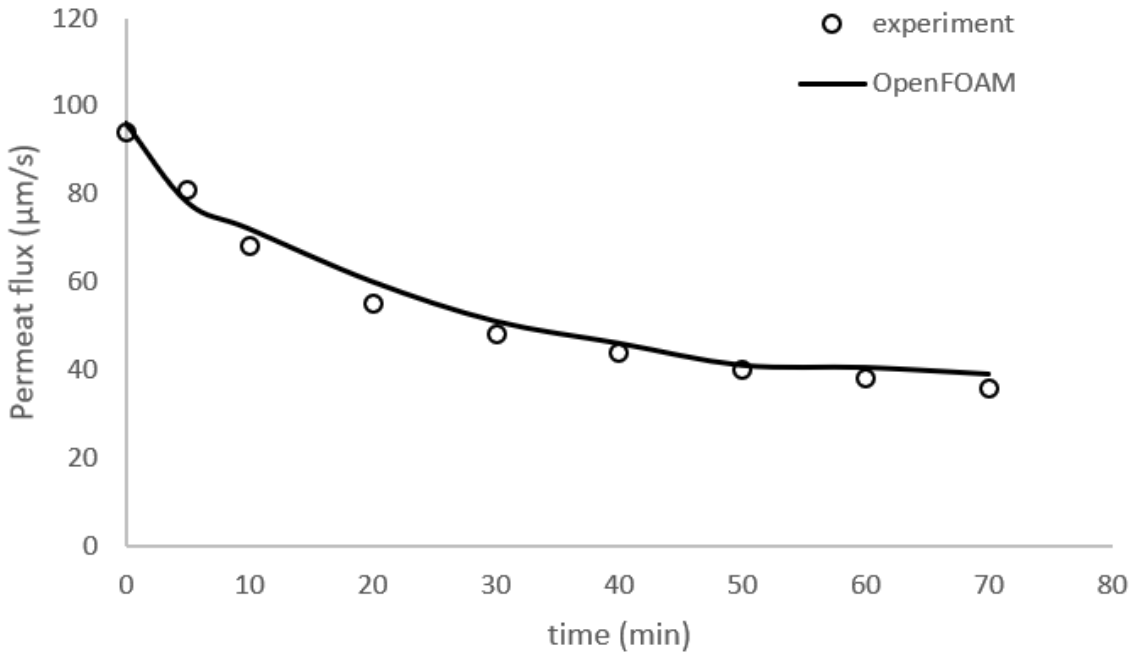

Figure 5.6 Permeate flux with time at TMP 70 kPa

It can be observed that the model and experimental data are very much in agreement. But at the beginning, the flux calculated in the simulation decreases sharper than in the experiment. The possible reason is the porosity ε_c used to calculate the specific cake resistance by Eqn. (4.2) is not accurate. The maximum volume fraction is used to get porosity. In other words, the porosity is smaller than the actual value and therefore the specific cake resistance used in the simulation is larger. Additionally, constant porosity with time can cause over estimation of the permeate flux decline (Kim, Marion, Jeong & Hoek, 2006). The other error shown in Figure 5.6 is that the flux obtained from the simulation is larger than in the experimental. This is the outcome of the above first error. As the flux is smaller at the beginning, it causes slower growth of the cake which results in smaller cake resistance according to Eqn. (4.10) which shows cake resistance depends on cake thickness.

Concentration inside the cake region at 70 min is shown in Figure 5.7.

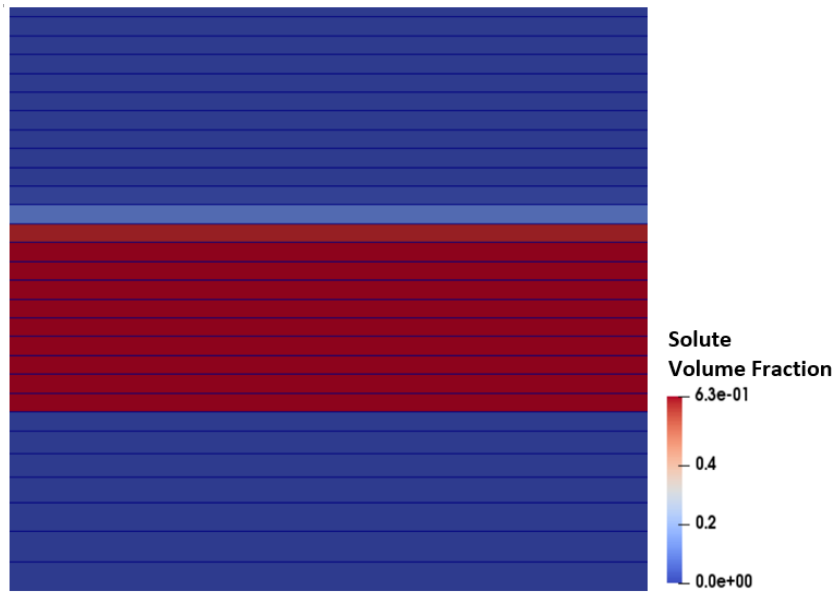

Figure 5.7 Concentration inside the cake region at 70 min

To validate the blockage of particles by membrane surface, a paticle mass balance equation is used (Hong, Faibish & Elimelech, 1997):

$$(\varphi_c - \varphi_0)\frac{d\delta_c}{dt} = v_w\varphi_0 \tag{5.5}$$

where φ_c is the volume fraction inside the concentration polarization layer, φ_0 is the bulk concentration, δ_c is the thickness of the concentration polarization layer and v_w is the permeate flux. After particles accumulate for 18.3 second, the volume fraction is 0.07 as shown in Figure 5.8. At this time, δ_c is considered as the vertical mesh size and v_w is considered as the initial permeate flux. φ_c obtained by using Eqn. (5.5) is 0.09. Thus the difference is 22.2%.

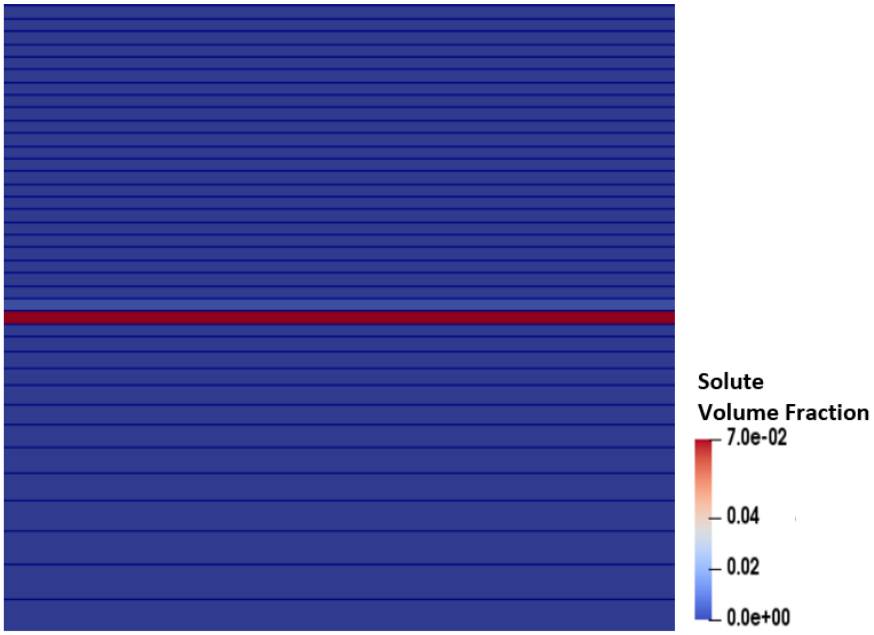

Figure 5.8 Concentration at 18.3 s

5.2.2 Simulation of Seungkwan Hong's Model

Work in this chapter follows Seungkwan Hong's model (Hong, Faibish & Elimelech, 1997).

5.2.2.1 Model Description

The geometry of the filtration device is shown in Figure 5.9. Fluid flows in the same direction as Kim's model (Kim, Marion, Jeong & Hoek, 2006). The width of upper channel is the same as Hong's model. However, the length of the channel and the membrane is shorter to save computational cost but this does not have impact on the results. To get stable results, the inlet and right outlet are set away from the membrane.

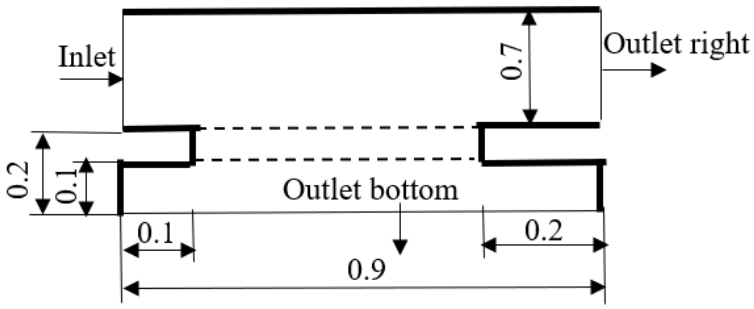

Figure 5.9 Geometry of Hong's model

The particle radius in this model is $100\,nm$ and the temperature is $20^{\circ}C$, therefore the diffusion coefficient can be calculated by using Eqn. (4.9). The thickness of the membrane is not given in Kim's model and is set to be the width of the bottom channel. The porosity of the randomly packed cake layer, ε_c is set to 0.36 which is the same as in Hong's model. The specific cake resistance, r_c' can be calculated using Eqn. (4.10) and is $3.95\times10^{16}\,m^{-2}$. When the cake is formed, the specific membrane resistance at the membrane cells below the cake is updated through Eqn. (4.11).

In this model, TMP is 40 kPa. The crossflow velocity is set $0.002\,m/s$ which is one tenth of the velocity in Hong's model. Since in Eqn. (3.1) tangential velocity has the potential to blow away cake, velocity is set small to capture cake. In Hong's work, crossflow velocity has no significant effect on cake formation. Fixed inlet concentration of 1×10^{-4} is used which is the same as initial condition for upper channel. For this filtration device, the resistance of the membrane R_m is $1.25\times10^{12}\,m^{-1}$.

Similar to Kim's model, the vertical mesh size inside the cake region is set to be 1 μm to capture the cake. Outside the cake region and above the membrane, graded mesh where cell-to-cell expansion ratio is about 1.1 is used.

5.2.2.2 Results

Concentration inside the cake region at 25min is shown in Figure 5.9.

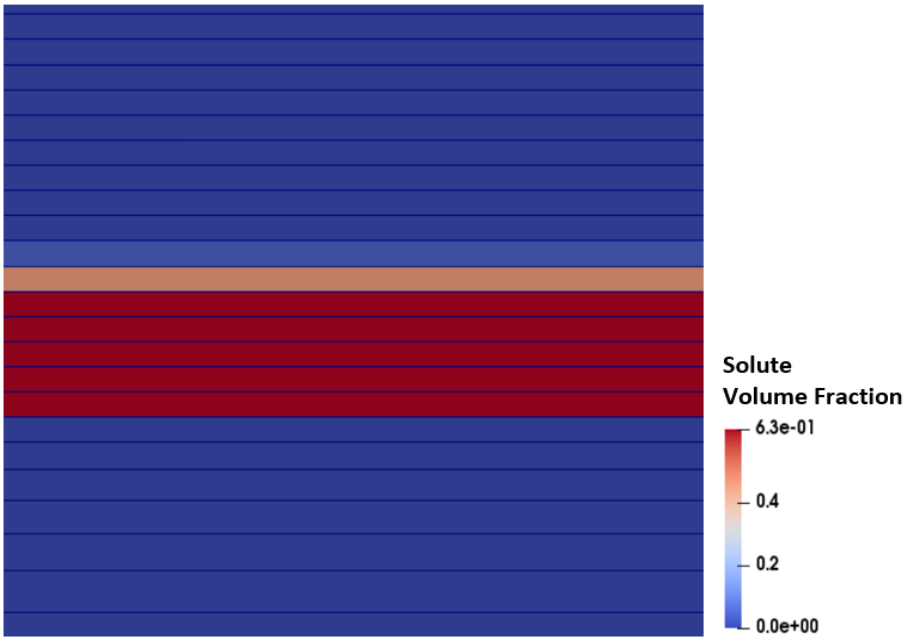

Figure 5.10 Concentration inside the cake region at 25min

Permeate flux with time is shown in Figure 5.11.

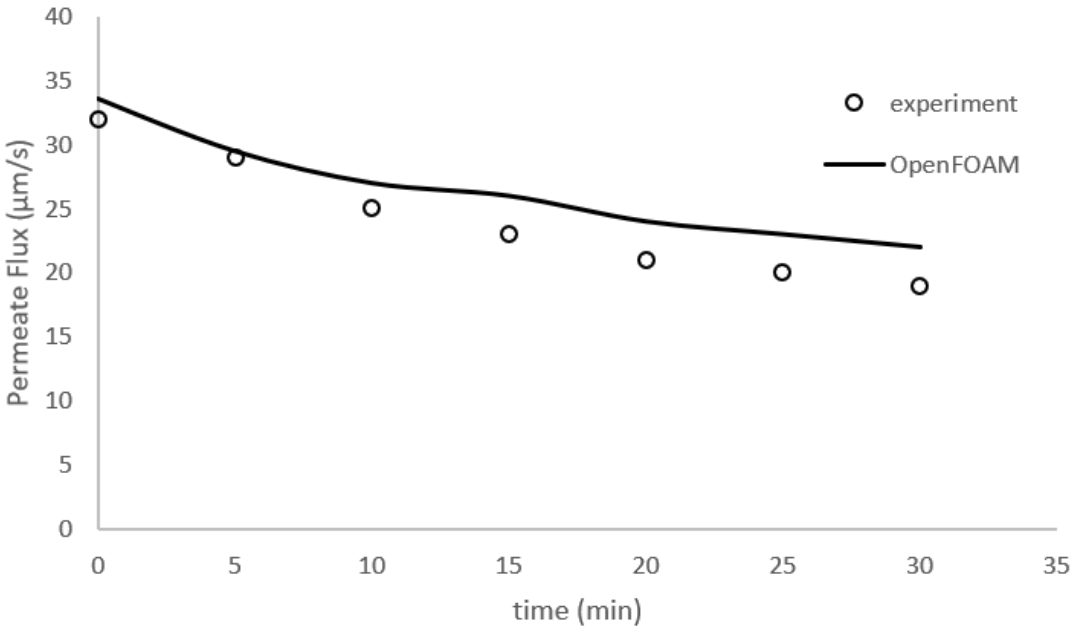

Figure 5.11 Flux with time in the case of 100 *nm* particle size under TMP of 40 kPa

Compared to experimental data, the flux obtained from the simulation is larger and deceases slower. The slow decrease in the flux is because slow growth of the cake. The possible reason is the convection diffusion equation in the simulation has axial diffusion and axial convection. In Hong's numerical model, axial diffusion is neglected because it is much smaller than the other terms under typical conditions of crossflow filtration. Additionally, the assumption that the longitudinal transport of excess particles by crossflow is negligible is used (Hong, Faibish & Elimelech, 1997). Since two terms remain in the simulation, they would have impact on the concentration. To verify the way to represent cake resistance for the flow, the theoretical relation between permeate flux and resistance is described by a Darcy type expression of the form (Cho, Kim, Moon & Kwon, 2006):

$$v = \frac{\triangle p}{\mu(R_m + r_c \delta_c)} \tag{5.6}$$

At 25min, there are five cake layers and thus the thickness of cake is 5 μm and permeate flux is 23 $\mu m / s$ as it is shown in Figure 5.11. From Eqn. (5.6), permeate flux is 27 $\mu m / s$ .So the difference is about 15%.

5.3 Application to Dead End Filtration

As demonstrated in the above two nanofiltration cases, the new model works well and simulation results agree with experimental results.

The methods are applied to dead end filtration. The complete feed flow is forced through the membrane. There are two models in this section. One model uses the microfiltration membrane condition at TMP 4180 Pa introduced in Chapter 5.1.1 and the other one uses the membrane condition in Hong's model introduced in Chapter 5.2.2.1.

The variable of interest in dead end filtration is velocity inside the membrane and cake structure. In terms of initial condition, pressures at both inlet and outlet are set at the beginning. In this way, it is a pressure driven flow. The general geometry is shown in Figure 5.12. The membrane is in the middle of the device and surfaces expect inlet and outlet are walls.

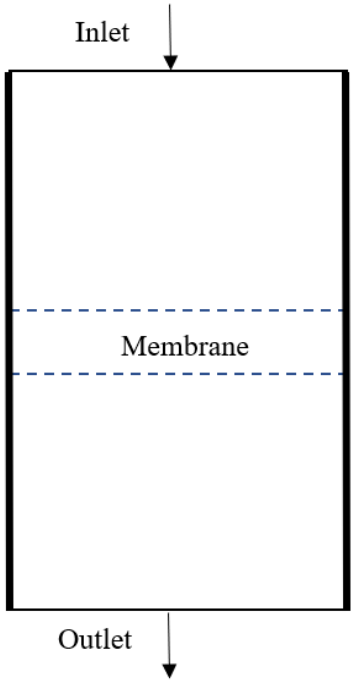

Figure 5.12 Geometry of dead end filtration

For incompressible flow inside the membrane, the following momentum equation in y direction can be obtained:

$$\frac{Dv}{Dt} = -\nabla p + gY - \nu r_m v + \nu \frac{\partial^2 v}{\partial x^2} + \nu \frac{\partial^2 v}{\partial y^2} \tag{5.7}$$

where y is vertical direction, x is tangential direction, v is velocity in y direction, u is velocity in x direction, g is acceleration of gravity and Y is elevation. When it is steady

state and flow is fully developed, $\frac{Dv}{Dt}$ and $\nu\frac{\partial^2 v}{\partial y^2}$ vanish. The aim of solving this equation is to find how v inside the membrane behaves along the membrane length. Also velocity coming to and out of membrane are same based on mass conservation. Eqn. (5.7) can be written as:

$$\nu\frac{\partial^2 v}{\partial x^2} - \nu r_m v = \nabla p - gY \tag{5.8}$$

with no slip condition:

$$v(x=0)=0$$

$$v(x=L)=0 \tag{5.9}$$

The exact solution to Eqn. (5.9) is as follows:

$$v(x) = C_1 e^{\sqrt{r_m}x} + C_2 e^{-\sqrt{r_m}x} + v_0$$

$$C_1 = -v_0 - C_2$$

$$C_2 = -\frac{v_0(e^{\sqrt{r_m}L}-1)}{e^{\sqrt{r_m}L}-e^{-\sqrt{r_m}L}})$$

$$v_0 = -\frac{\nabla p - gY}{\nu r_m} \tag{5.10}$$

By ploting $v(x)$ with different r_m , the effect of r_m on the vertical velocity can be seen in Figure 5.13.

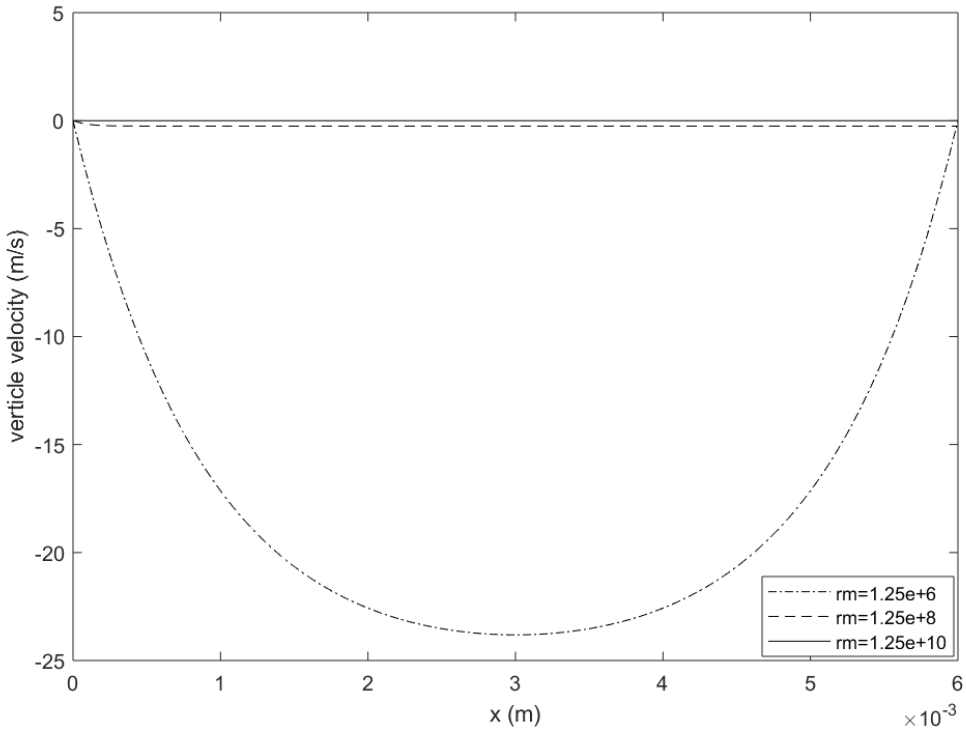

Figure 5.13 Effect of r_m on vertical velocity

With larger r_m, the vertical velocity is close to uniform along the tangential direction while it is easy to see the curved shape in the velocity profile for smaller r_m. In nanofiltration, r_m is in the order of 10^{15}. So C_2 is almost $-v_0$, C_1 is almost 0, $C_2 e^{-\sqrt{r_m}x}$ is almost 0, $v(x) \approx v_0$. This means the velocity inside the membrane region is almost constant in the tangential direction.

The plot of the vertical velocity in microfiltration is shown in Figure 5.14 and the area around the left side of the wall is zoomed in in Figure 5.15. Additionally, a plot of vertical velocity in Hong's model is shown in Figure 5.16.

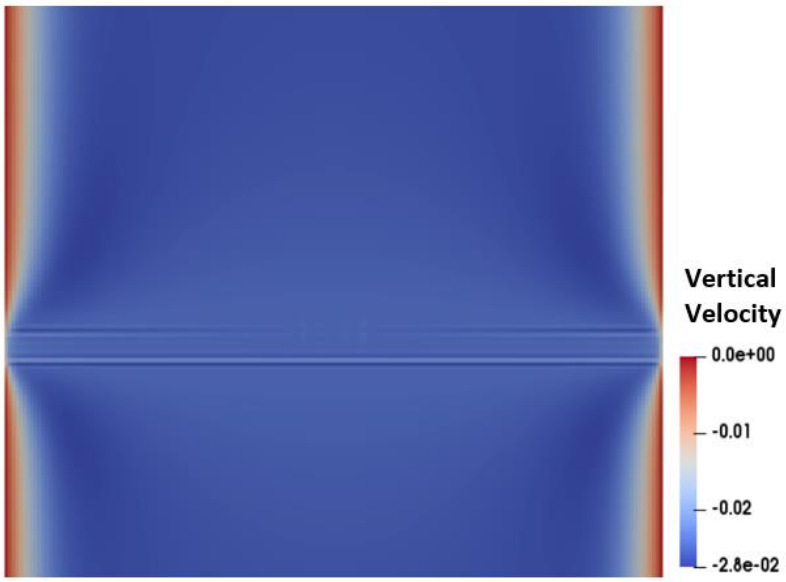

Figure 5.14 Vertical velocity in microfiltration

Figure 5.15 Area around left side of wall

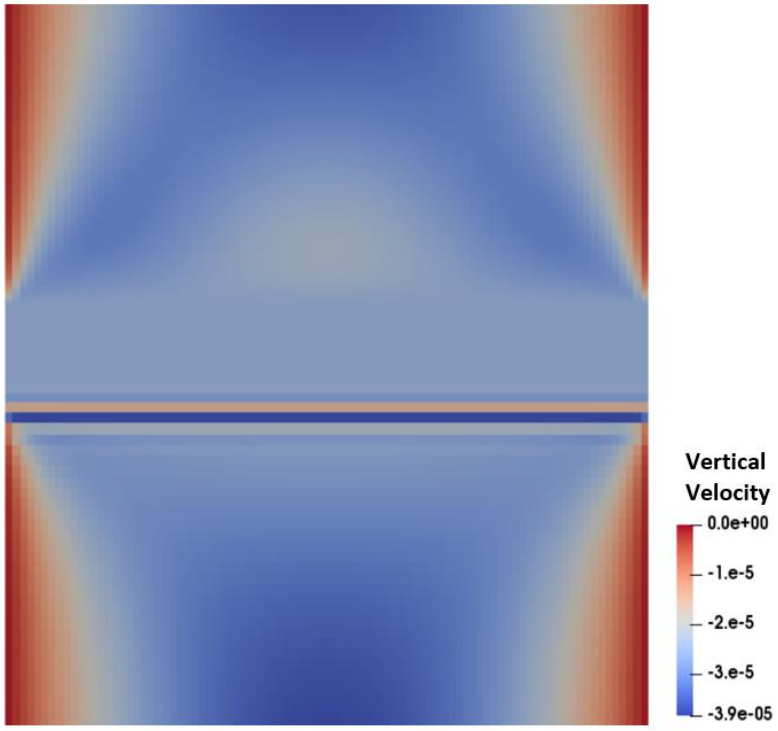

Figure 5.16 Vertical velocity in Hong's model

From both models, outside of membrane, there are symmetrical boundary layers near walls. Velocity in nanofiltration matches the analysis based on Eqn. (5.9) that it is almost constant along the membrane length.

Another important phenomenon is cake formation. A plot of volume fraction in microfiltration is shown in Figure 5.17. and plot of volume fraction in Hong's model is shown in Figure 5.18. The cake region in Figure 5.18 is zoomed in in Figure 5.19. Both Figure 5.17 and Figure 5.18 include the whole membrane length.

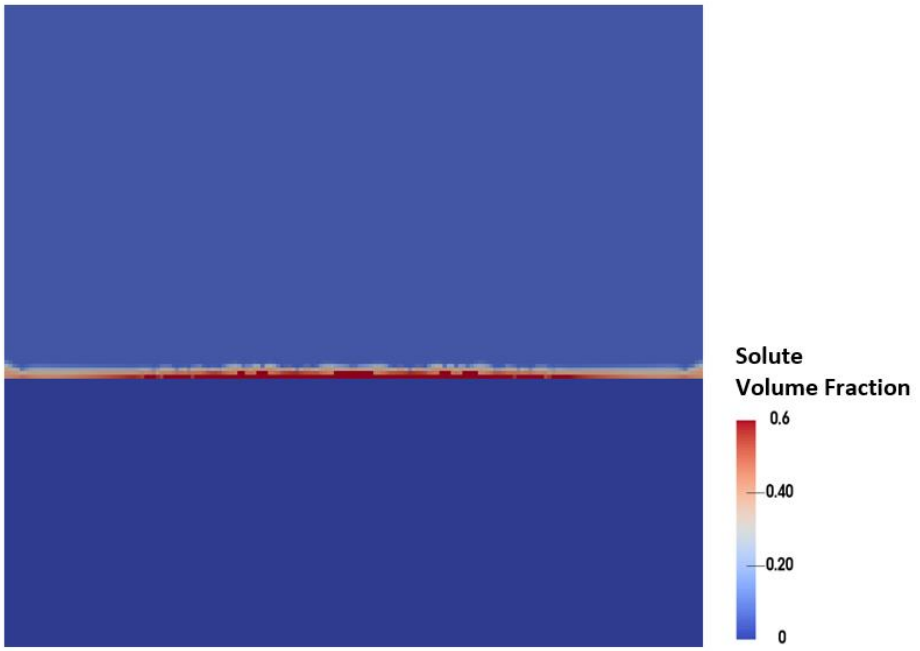

Figure 5.17 Volume fraction in microfiltration

Figure 5.18 Volume fraction in Hong's model

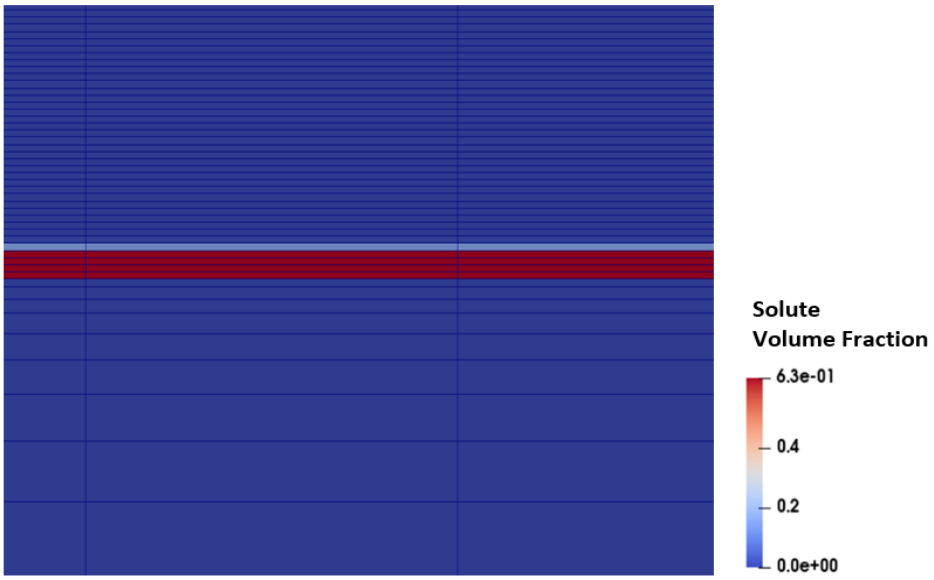

Figure 5.19 Cake region in Hong's model

Compared to the previous microfiltration cases, the thickest cake shown in Figure 5.17 is no longer at the end of the membrane while it is close to a parabolic shape. The cake in nanofiltration is almost uniform.

CHAPTER 6

CONCLUSIONS

6.1 Conclusions

In this thesis, a simple method to simulate blockage of particles on membrane surfaces is developed. The method is applied to each case and is validated by observing increase in volume fraction above the membrane and no increase below the membrane. Permeate flux in the microfiltration models is compared to theoretical data and the differences are acceptable.

Furthermore, a new technique to represent cake resistance to the flow is introduced. To validate the new technique, it is applied to Kim and Hong's model. It is seen that simulation results of permeate flux are in a very good agreement with experimental results. Concentration distribution inside the cake region is shown. Reasons for the differences in the flux are explored.

To understand the above two approaches better, simulation of dead end filtration is conducted. Velocity profiles inside the membrane region are studied. The cake structure is also observed to be consistent with velocity profile.

REFERENCES

CFD: PIMPLE Algorithm. (2018). *SimScale CAE Forum*. Retrieved 27 March 2018, from https://www.simscale.com/forum/t/cfd-pimple-algorithm/81418

Cho, J., Kim, I., Moon, J., & Kwon, B. (2006). Determining Brownian and shear-induced diffusivity of nano- and micro-particles for sustainable membrane filtration, *Desalination*, *188*(1-3),213-216. http://dx.doi.org/10.1016/j.desal.2005.04.119

Christodoss, D. (2013). *Tangential Flow Filtration to Enable High Solids Concentration, Improved Process Throughput, Capacity and Cost*. Presentation.

Holtzman, T. (2017). *Mathematics, Numerics, Derivations and OpenFOAM*.

Hong, S., Faibish, R., & Elimelech, M. (1997). Kinetics of Permeate Flux Decline in Crossflow Membrane Filtration of Colloidal Suspensions. *Journal Of Colloid And Interface Science*, *196*(2), 267-277. http://dx.doi.org/10.1006/jcis.1997.5209

Industrial applications of membrane filtration. (2018). *Industrial wastewater & air treatment*. Retrieved 30 March 2018, from https://blog-en.condorchem.com/industrial-applications-of-membrane-filtration/#.Wr7Lx-jwY2w

Jasak, H. (1996). *Error Analysis and Estimation for the Finite Volume Method with Applications to Fluid Flows*. the University of London.

Kim, S., Marion, M., Jeong, B., & Hoek, E. (2006). Crossflow membrane filtration of interacting nanoparticle suspensions. *Journal Of Membrane Science*, *284*(1-2), 361-372. http://dx.doi.org/10.1016/j.memsci.2006.08.008

Kromkamp, J., Bastiaanse, A., Swarts, J., Brans, G., van der Sman, R., & Boom, R. (2005). A suspension flow model for hydrodynamics and concentration polarisation in crossflow microfiltration. *Journal Of Membrane Science*, *253*(1-2), 67-79. http://dx.doi.org/10.1016/j.memsci.2004.12.028

Leighton, D., & Acrivos, A. (1986). Viscous resuspension. *Chemical Engineering Science*, *41*(6), 1377-1384. http://dx.doi.org/10.1016/0009-2509(86)85225-3

Paipuri, M. (2014). *Numerical Modelling of Membrane filtration using Lattice Boltzmann and Finite Volume Methods*. Swansea University.

Romero, C., & Davis, R. (1988). Global model of crossflow microfiltration based on hydrodynamic particle diffusion. *Journal Of Membrane Science*, *39*(2), 157-185. http://dx.doi.org/10.1016/s0376-7388(00)80987-4

Thiam Teik Wan, W. (2015). Experimental Study of the Separation of Oil in Water Emulsions by Tangential Flow Microfiltration Process. Part 1: Analysis of Oil Rejection Efficiency and Flux Decline. *Journal Of Membrane Science & Technology*, *05*(01). http://dx.doi.org/10.4172/2155-9589.1000130

APPENDIX A

MAIN CODES OF MICROFILTRATION SOLVER

1)

```
// calculate diffusion coefficient D

vector abcdex(1,0,0);
vector abcdey(0,1,0);
//shear rate
const volScalarField shearRate = mag((fvc::grad(U & abcdex)) & abcdey);
shear=shearRate;
//D
dVf=0.33*shearRate*pow(a,2)*pow(vf,2)*(1+0.5*exp(8.8*vf));

const labelUList& owner1 = mesh.owner();
const labelUList& neighbour1 = mesh.neighbour();

//below the membrane, D=0
forAll (mesh.C(),celli)
        {
                if (mesh.C()[celli].y()<=11.2)
                {
                        dVf[celli]=0;
                }
        }
//when there is a cake, D=0
forAll (mesh.C(),celli)
        {
                if (srcu[celli]>0)
                {
                        dVf[celli]=0;
                }
        }
```

2)

```
// define the source term for membrane
int fg2=0;
double Rm=1.62e+2;
double thi=1.2;
const volScalarField nu2 = turbulence->nu();

forAll (mesh.C(),celli)
{
//find membrane region
          if (mesh.C()[celli].x()>=(15)  && mesh.C()[celli].x()<=(45) &&
mesh.C()[celli].y()>(10) && mesh.C()[celli].y()<(11.2))
          {
                fg2=1;
```

```
            }
        srcm[celli]=fg2*nu2[celli]*Rm/thi;
            fg2=0;
}


3)
// define the source term for U
int fg=0;
const volScalarField nu1 = turbulence->nu();

forAll (mesh.C(),celli)
{

            if (vf[celli]>=0.6)
            {
                  fg=1;
                  srcu[celli]=fg*nu1[celli]/nu1[celli]*5.0*pow((1-
0.4),2)*pow((3/a.value()),2)/pow(0.4,3);
            }

            fg=0;

}


forAll (mesh.boundaryMesh(), patchI)
{
            forAll(srcu.boundaryField()[patchI], faceI)
        {

                if (vf.boundaryField()[patchI][faceI]>=0.6)
                {
                      fg=1;

srcu.boundaryFieldRef()[patchI][faceI]=fg*nu1.boundaryField()[patchI][faceI]/nu1.boun
daryField()[patchI][faceI]*5.0*pow((1-0.4),2)*pow(3/a.value(),2)/pow(0.4,3);
                }

            fg=0;
            }

}
```

4)
```
// Solve the Momentum equation

MRF.correctBoundaryVelocity(U);

tmp<fvVectorMatrix> tUEqn
(
    fvm::ddt(U) + fvm::div(phi, U)
  + MRF.DDt(U)
  + turbulence->divDevReff(U)
 ==
    fvOptions(U)-fvm::Sp(srcm, U) -fvm::Sp(srcu, U)
);
fvVectorMatrix& UEqn = tUEqn.ref();

UEqn.relax();

fvOptions.constrain(UEqn);

if (pimple.momentumPredictor())
{
    solve(UEqn == -fvc::grad(p));

    fvOptions.correct(U);
}
int num=0;
int num2=0;
double sum=0.0;
double sum2=0.0;
double ave=0.0;
double ave2=0.0;
double sum3=0.0;
double ave3=0.0;
double sum4=0;
double sum5=0;
const labelUList& owner = mesh.owner();
const labelUList& neighbour = mesh.neighbour();

forAll(owner, facei)
            {
                //look for membrane surface
        if  ( (srcm[owner[facei]]!=0 && srcm[neighbour[facei]]==0) ||
(srcm[owner[facei]]==0 && srcm[neighbour[facei]]!=0))
                {
                        //top membrane surface
```

```
                        if (mesh.C()[neighbour[facei]].y() > 11.2 ||
mesh.C()[owner[facei]].y() > 11.2)
                        {
                                sum=sum+U[neighbour[facei]].x();
                                sum2=sum2+U[neighbour[facei]].y();
                                sum3=sum3+U[owner[facei]].y();
                                num=num+1;
                        }
                        //bottom membrane surface
                        if (mesh.C()[neighbour[facei]].y() > 10 &&
mesh.C()[owner[facei]].y() < 10)
                        {
                                sum4=sum4+U[neighbour[facei]].y();
                                sum5=sum5+U[neighbour[facei]].x();
                                num2=num2+1;
                        }

                }
        }
reduce(sum,sumOp<scalar>());
reduce(sum2,sumOp<scalar>());
reduce(sum3,sumOp<scalar>());
reduce(num,sumOp<scalar>());
ave=sum2;
ave2=sum2/num;
ave3=sum3/num;

//Info<< ave <<endl;
Info<< ave2 <<endl;
Info<< num <<endl;
//Info<< ave3 <<endl;
Info<< sum4 <<endl;
Info<< num2 <<endl;
Info<< sum5 <<endl;
```

5) viscosity

```
namespace Foam
{
namespace viscosityModels
{
   defineTypeNameAndDebug(changenutwo, 0);
   addToRunTimeSelectionTable(viscosityModel, changenutwo, dictionary);
}
}
```

```
Foam::volScalarField
Foam::viscosityModels::changenutwo::calcNu() const
{

        const fvMesh& mesh_ =  U_.mesh();

        const volScalarField& vf =  mesh_.lookupObject<volScalarField>("vf");
        volScalarField newNu=nu0_*vf;

        forAll (mesh_.C(),celli)

        {
                if (vf()[celli]<0.6)
                {
                        newNu[celli]= nu0_.value()*pow(1+1.5*vf()[celli]/(1-
vf()[celli]/0.6),2);
                }
                else
                {
                        newNu[celli]= nu0_.value()*1;
                }

        }


        forAll (mesh_.boundaryMesh(), patchI)
        {
          forAll(vf.boundaryField()[patchI], faceI)
      {
                if (vf.boundaryField()[patchI][faceI]<0.6)
                {

        newNu.boundaryFieldRef()[patchI][faceI]=nu0_.value()*pow(1+1.5*vf.boundary
Field()[patchI][faceI]/(1-vf.boundaryField()[patchI][faceI]/0.6),2);
                }
                else
                {
                        newNu.boundaryFieldRef()[patchI][faceI]= nu0_.value()*1;
                }
      }
      }

        return newNu;
}
```

6) main function

```
int main(int argc, char *argv[])
{
    #include "postProcess.H"

    #include "setRootCase.H"
    #include "createTime.H"
    #include "createMesh.H"
    #include "createControl.H"
    #include "createTimeControls.H"
    #include "createFields.H"
    #include "createFvOptions.H"
    #include "initContinuityErrs.H"

    turbulence->validate();

    // * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

    Info<< "\nStarting time loop\n" << endl;
    //dimensionSet::debug = 0;
    while (runTime.run())
    {
        #include "readTimeControls.H"
        #include "CourantNo.H"
        #include "setDeltaT.H"

        runTime++;

        Info<< "Time = " << runTime.timeName() << nl << endl;
            const labelUList& owner6 = mesh.owner();
            const labelUList& neighbour6 = mesh.neighbour();
        // --- Pressure-velocity PIMPLE corrector loop
        while (pimple.loop())
        {


            #include "sourcemem.H"
            #include "UEqn.H"

            // --- Pressure corrector loop
            while (pimple.correct())
            {
                #include "pEqn.H"
            }
```

```
        if (pimple.turbCorr())
        {
            laminarTransport.correct();
            turbulence->correct();
        }
    }

        // wait for fuuly developed
        if (mesh.time().value() > 17907) //17910(4180pa)16480(1780pa)17980(1180pa)
        {
          abc=abc+1;
          if (abc==1)
          {
//interfield
                forAll (mesh.C(),celli)
        {
                  if (mesh.C()[celli].y()>11.2)
                  {
                        vf[celli]=0.05;

                  }
                  if (mesh.C()[celli].y()<=11.2)
                  {
                        vf[celli]=0;

                  }
                }
          }

          #include "dVfcal.H"

           surfaceScalarField dVff = fvc::interpolate(dVf);
           forAll(neighbour6, facen)
           {
//all surfaces inside membrane including top
        if  (srcm[neighbour6[facen]]!=0 || srcm[owner6[facen]]!=0 ||
srcu[neighbour6[facen]]!=0 || srcu[owner6[facen]]!=0)
                {
                        phi[facen]=0;
                        dVff[facen]=0;

                }
           }

//convection diffusion equation
```

```
            fvScalarMatrix vfEqn
            (
                fvm::ddt(vf)
              + fvm::div(phi, vf)
              - fvm::laplacian(dVff, vf)
              ==
                fvOptions(vf)
            );

            vfEqn.relax();
            fvOptions.constrain(vfEqn);
            vfEqn.solve();
            fvOptions.correct(vf);
                #include "sourceU.H"
    }


        runTime.write();

        Info<< "ExecutionTime = " << runTime.elapsedCpuTime() << " s"
            << "  ClockTime = " << runTime.elapsedClockTime() << " s"
            << nl << endl;
        //Info<< mesh.time().value() << endl;

    }

    Info<< "End\n" << endl;

    return 0;
}
```

APPENDIX B

MAIN CODES OF SOLVER FOR KIM'S MODEL

1)
// define the equation for D

```
forAll (mesh.C(),celli)
        {
                dVf[celli]=3.6e-12;
              }
const labelUList& owner1 = mesh.owner();
const labelUList& neighbour1 = mesh.neighbour();

forAll (mesh.C(),celli)
        {
                if (mesh.C()[celli].y()<=0.0032)
                {
                        dVf[celli]=0;
                }
              }

forAll (mesh.C(),celli)
        {
                if (srcu[celli]>0)
                {
                        dVf[celli]=0;
                }
              }
```

2)
// define the source term for membrane

```
int fg2=0;
double Rm=8.11e+11;
double thi=0.0012;
int numb=0;
const volScalarField nu2 = turbulence->nu();
//find the thickest cake

forAll(mesh.C(),cellh)
{
        if(vf[cellh]>0.63)
        {
                ymax[cellh]=mesh.C()[cellh].y();
                forAll(mesh.C(),cellj)
                {
                        if ( abs(mesh.C()[cellj].x()-mesh.C()[cellh].x())<1e-8 &&
vf[cellj]>0.63 && mesh.C()[cellj].y()>ymax[cellh])
                        {
```

```
                                ymax[cellh]=mesh.C()[cellj].y();
                        }
                }
                rc[cellh]=((ymax[cellh]-0.0032)/(0.75e-6)*2+1)/2*(0.75e-6)*(1.6e+17);
        }
}
forAll (mesh.C(),celli)
{
          if (mesh.C()[celli].x()>=(0.02)  && mesh.C()[celli].x()<=(0.07) &&
mesh.C()[celli].y()>(0.002) && mesh.C()[celli].y()<(0.0032))
          {
                fg2=1;
                srcm[celli]=fg2*nu2[celli]*Rm/thi;
//if with same x position, there is a cake
                forAll(mesh.C(),cellm)
                {
                        if ( abs(mesh.C()[cellm].x()-mesh.C()[celli].x())<1e-8 &&
vf[cellm]>0.63)
                        {

        srcm[celli]=fg2*nu2[celli]*Rm/thi+fg2*nu2[celli]*rc[cellm]/(thi+ymax[cellm]-
0.0032);
                        }
                }
          }

}
3)
// define the source term for U
const volScalarField nu1 = turbulence->nu();
int fg=0;
forAll(mesh.C(),cellh)
{
        if(vf[cellh]>0.63)
        {
                ymax[cellh]=mesh.C()[cellh].y();
                forAll(mesh.C(),cellj)
                {
                        if ( abs(mesh.C()[cellj].x()-mesh.C()[cellh].x())<1e-8 &&
vf[cellj]>0.63 && mesh.C()[cellj].y()>ymax[cellh])
                        {
                                ymax[cellh]=mesh.C()[cellj].y();
                        }
                }
                rc[cellh]=((ymax[cellh]-0.0032)/(0.75e-6)*2+1)/2*(0.75e-6)*(1.6e+17);
```

```
                srcu[cellh]=nu1[cellh]*rc[cellh]/(0.0012+ymax[cellh]-0.0032);
        }
}
4)
// Solve the Momentum equation

MRF.correctBoundaryVelocity(U);

tmp<fvVectorMatrix> tUEqn
(
    fvm::ddt(U) + fvm::div(phi, U)
  + MRF.DDt(U)
  + turbulence->divDevReff(U)
 ==
    fvOptions(U)-fvm::Sp(srcm, U)-fvm::Sp(srcu, U)
);
fvVectorMatrix& UEqn = tUEqn.ref();

UEqn.relax();

fvOptions.constrain(UEqn);

if (pimple.momentumPredictor())
{
    solve(UEqn == -fvc::grad(p));

    fvOptions.correct(U);
}
int num=0;
int num1=0;
int num2=0;
double sum=0.0;
double sum1=0.0;
double ave=0.0;
double ave1=0.0;
double sum2=0;
double ave2=0;
double sum3=0;
double ave3=0;
const labelUList& owner = mesh.owner();
const labelUList& neighbour = mesh.neighbour();

forAll(owner, facei)
        {
```

```
        if  (srcm[owner[facei]]!=0 && srcm[neighbour[facei]]==0) //||
(srcm[owner[facei]]==0 && srcm[neighbour[facei]]!=0))
                {
// calclate permeate flux
                        if (mesh.C()[neighbour[facei]].y() > 0.0032 )
                        {
                                sum=sum+U[neighbour[facei]].y();
                                sum1=sum1+U[owner[facei]].y();
                                num=num+1;
                        }
//flux inside cake
                        if(vf[neighbour[facei]]>=0.63)
                        {
                                        num2=num2+1;
                                        sum2=sum2+U[neighbour[facei]].y();
                                        //sum3=sum3+U[owner[facei]].y();
                        }
                }
            }
```

5) viscosity for nanofiltration

```
namespace Foam
{
namespace viscosityModels
{
   defineTypeNameAndDebug(changenunf, 0);
   addToRunTimeSelectionTable(viscosityModel, changenunf, dictionary);
}
}

//calcNu

Foam::volScalarField
Foam::viscosityModels::changenunf::calcNu() const
{

        const fvMesh& mesh_ =  U_.mesh();

        const volScalarField& vf =  mesh_.lookupObject<volScalarField>("vf");
        volScalarField newNu=nu0_*vf;

//Foam::volScalarField nu2_;


        forAll (mesh_.C(),celli)
```

```
        {
                if (vf()[celli]<0.625)
                {
                        newNu[celli]= nu0_.value()*pow(1+1.5*vf()[celli]/(1-
vf()[celli]/0.63),2);
                }
                else
                {
                        newNu[celli]= nu0_.value()*1;
                }
        }


        forAll (mesh_.boundaryMesh(), patchI)
        {
          forAll(vf.boundaryField()[patchI], faceI)
      {
                if (vf.boundaryField()[patchI][faceI]<0.625)
                {

        newNu.boundaryFieldRef()[patchI][faceI]=nu0_.value()*pow(1+1.5*vf.boundary
Field()[patchI][faceI]/(1-vf.boundaryField()[patchI][faceI]/0.63),2);
                }
                else
                {
                        newNu.boundaryFieldRef()[patchI][faceI]= nu0_.value()*1;
                }
      }
      }
        return newNu;
}
```

6) main function

```
#include "fvCFD.H"
#include "singlePhaseTransportModel.H"
#include "turbulentTransportModel.H"
#include "pimpleControl.H"
#include "fvOptions.H"

// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

int main(int argc, char *argv[])
{
   #include "postProcess.H"
```

```
#include "setRootCase.H"
#include "createTime.H"
#include "createMesh.H"
#include "createControl.H"
#include "createTimeControls.H"
#include "createFields.H"
#include "createFvOptions.H"
#include "initContinuityErrs.H"

turbulence->validate();

// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

Info<< "\nStarting time loop\n" << endl;
//dimensionSet::debug = 0;
while (runTime.run())
{
    #include "readTimeControls.H"
    #include "CourantNo.H"
    #include "setDeltaT.H"

    runTime++;

    Info<< "Time = " << runTime.timeName() << nl << endl;
        const labelUList& owner6 = mesh.owner();
        const labelUList& neighbour6 = mesh.neighbour();
    // --- Pressure-velocity PIMPLE corrector loop
    while (pimple.loop())
    {


        #include "sourcemem.H"
        #include "UEqn.H"

        // --- Pressure corrector loop
        while (pimple.correct())
        {
            #include "pEqn.H"
        }

        if (pimple.turbCorr())
        {
            laminarTransport.correct();
            turbulence->correct();
```

```
        }
      }

//wait for fullt developed
        if (mesh.time().value() > 10)
        {
          abc=abc+1;
       if (abc==1)
          {
//interfield
                forAll (mesh.C(),celli)
        {
                  if (mesh.C()[celli].y()>0.0032)
                  {
                        vf[celli]=4e-5;

                  }
                  if (mesh.C()[celli].y()<=0.0032)
                  {
                        vf[celli]=0;

                  }
                }
          }

//convection diffusion equation
            #include "dVfcal.H"
             double vf6=0;
             surfaceScalarField dVff = fvc::interpolate(dVf);
//all faces inside membrane including top and bottom
             forAll(neighbour6, facen)
             {
         if  (srcm[neighbour6[facen]]!=0 || srcm[owner6[facen]]!=0 ||
srcu[neighbour6[facen]]!=0 || srcu[owner6[facen]]!=0)
                {
                        phi[facen]=0;
                        dVff[facen]=0;

                }
             }

        fvScalarMatrix vfEqn
        (
           fvm::ddt(vf)
         + fvm::div(phi, vf)
```

```
            - fvm::laplacian(dVff, vf)
            ==
              fvOptions(vf)
          );

          vfEqn.relax();
          fvOptions.constrain(vfEqn);
          vfEqn.solve();
          fvOptions.correct(vf);
             #include "sourceU.H"
             forAll(mesh.C(),celli)
             {
                    if (vf[celli]>=vf6)
                    {
                           vf6=vf[celli];
                    }
             }
             Info<<vf6<<endl;

}


       runTime.write();

       Info<< "ExecutionTime = " << runTime.elapsedCpuTime() << " s"
          << "  ClockTime = " << runTime.elapsedClockTime() << " s"
          << nl << endl;

    }

    Info<< "End\n" << endl;

    return 0;
}
```