2D Stochastic Modeling of Porous Media Flow: Effect of Membrane Morphology on Filter Performance (revisited)

Justin Lee, Siddarth Kunisetty, Anay Badlani, Heer Patel

With guidance from Professor Kondic, Professor Cummings, and Mr. Binan Gu from NJIT

October 5, 2021

Abstract

Filtration systems are an abundant and necessary component of a multitude of applications. Often used in the food and chemical industries, efficient filtration of media is undoubtedly beneficial. However, a current issue with filtration systems is the build-up of a “cake layer” over time, a barrier of particles, which impedes the movement of fluid through the filter. The current solution is to remove the cake build-up through a backwash of fluid, but this takes time and reduces productivity. Varying the membrane morphology (shape of membrane) affects the performance of the filter, and this paper attempts to find the optimal membrane shape.

Introduction

The research specified in this paper uses a 2 dimensional simulated pore and random walk simulation with varying probabilities to simulate the movement of fluid through a pore. We build a computational model with detailed definitions and algorithmic processes. We vary the probabilities of movement based on the flow patterns of the fluid, allowing the simulation of several cases. We have also varied the probability that particles stick to each other and the walls of the pore, known as sticking probability, to see the change in the optimal angle. Finally, we have also varied the shape of the membrane itself by varying the angle of the walls of the pore and the area of the pore.

Simulation Methodology

The goal of the computational model is to (1) capture particle aggregate formations along the inner walls of defined membrane surfaces while (2) ignoring surface cake formation and (3) collecting appropriate data to represent pore performance. The model itself is written in C++11 and is implemented with adjacency matrices, vectors, and objects that can be easily manipulated to change various simulation variables. The relative scale of the model, although limited by the IDE stack size, can be altered as well.

We define each 2D membrane geometry by it's horizontal pore width, vertical pore height, and computational domain size: the ordered triplet (X_n, Y_n, D_n)

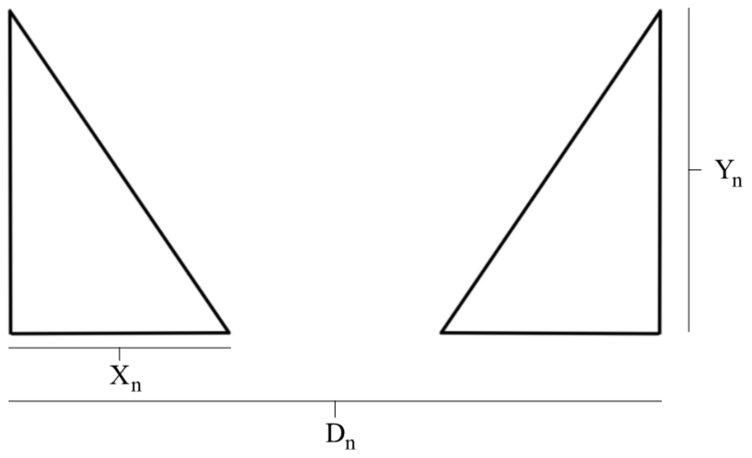


Figure 1. generalized pore geometry

Our methodology examines pore performance from all pore geometries satisfying a **constant** vertical pore height and a **constant** pore area, defined as the area of a regular trapezoid:

$$Area = (Y_n) * (D_n - X_n)$$

The maximum computational domain size is 100 units and the vertical range is 200 units, implemented through a 100 by 200 2D array. To generate pore with profile (X_1, Y_1, D_1), simply iterate through the computational domain following the linear equations below.

$$y < -(\frac{Y_1}{X_1}) * x + Y_1$$

$$y < (\frac{Y_1}{X_1}) * (x - (D_1 - 1 - X_1))$$

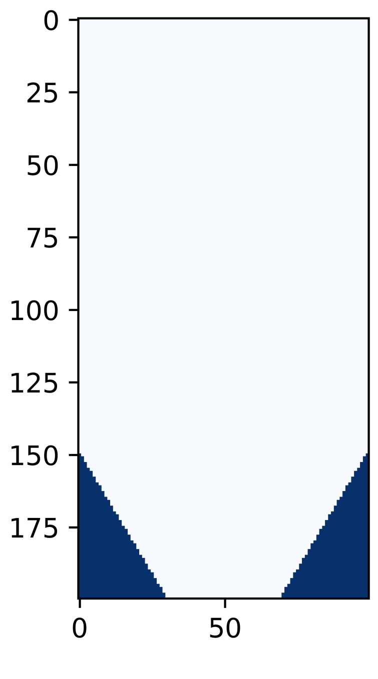


Figure 2. (30, 50, 100) pore generation

With no flat, horizontal surfaces, the pore effectively removes the cake-build up problem outside the pore.

Particle generation follows Brownian Motion and implements a random-walk algorithm to simulate particle movement. To start, a seed defined by $(X,\ Y)$ is selected randomly following these conditions:

$$200 > Y >= 185$$

$$D_n > X >= 0$$

The particle then follows a quinti-directional random-walk defined by a probability density function (§ 3), which produces 5 individual probabilities: leftwards, diagonal left-down, downwards, diagonal right-down, and rightwards. We call these 5 probabilities $P_1, P_2, ...P_5$ respectively.

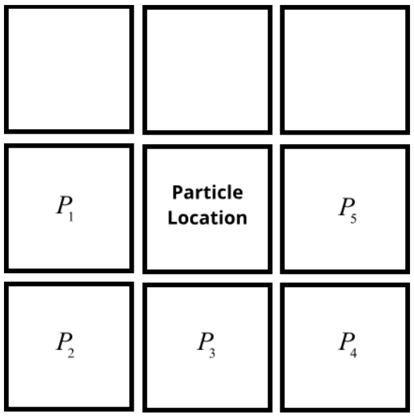


Figure 3. probability directions

The model has a wrap-around effect where the particle will emerge on the opposite side if it travels into either wall of the computational domain. The particle continues to walk until it either (1) becomes permanently stuck against another particle or the membrane surface or (2) it exits the pore opening at $y = 0$.

The simulation concludes when a pore has been completely sealed; that is, when there exists a connected path of particles completely covering the pore opening (diagonal connections are allowed).

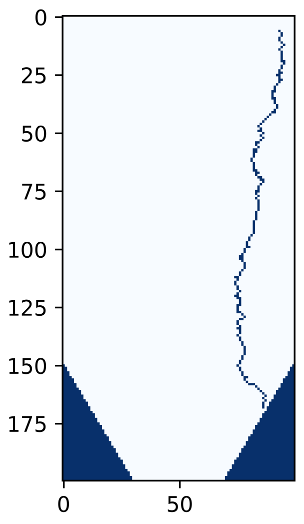


Figure 4. a single particle's random-walk path

Closure detection and percolation algorithms have been previously developed using path finding methodologies (such as A* heuristic search[1]), but this approach drastically increases the time complexity of the simulation. For each stuck particle, a new search must take place that exhausts all possible paths before returning.

Instead, by separately labeling each particle left/right, each new search only needs to test the 8 adjacent spaces[2] around a location for a particle of opposite orientation. In the case that separately labeled particles cannot be implemented, another developed method included in our model would be a contour tracing algorithm[3], which outlines the particle

1. mentions the search algorithm utilized in Paz, Sidhu, Sousa "2D Stochastic Modeling of Porous Media Flow; Effect on Membrane Morphology on Filter Performance"

2. at most 8 spaces, there are exceptions for spaces bordering the computational space

3. in GitHub link at (§ 6)

aggregate, and determines closure based on the final resting position of the tracing point. Both methods work and are fast compared with path-finding approaches.

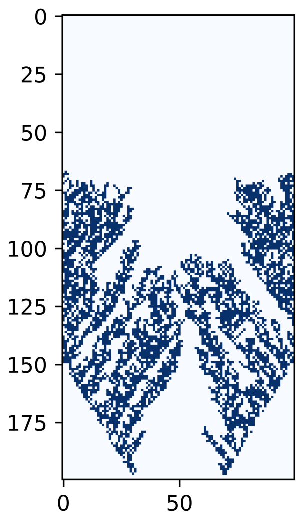


Figure 5. particle aggregate of one simulation; pore geometry (30, 50, 100)

The model collects the total particles stuck, total particles spawned, and total particle aggregate structure of each simulation. Compiling over multiple simulations, we can produce a particle aggregate density image. Darker regions indicate higher density of stuck particles.

The simulation averages 5.3 full simulations per second, or 5.6 simulations per second if disregarding data collection and storage.

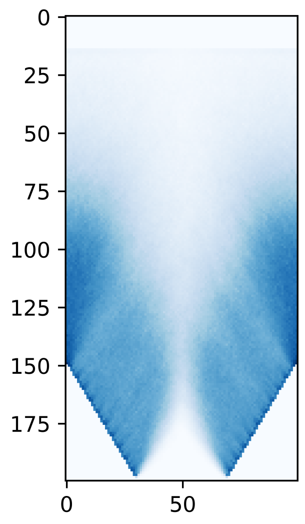


Figure 6. particle aggregate over 1000 simulations

Normal Distribution Particle Flow

The random-walk of the particle is governed by a probability density function (PDF), specifically that of a normal distribution with standard deviation σ. Given a random number machine, it's PDF is defined as an integral whose value between two numbers A and B represent the relative probability of the random number machine producing a value between A and B. We wanted our particle to behave according to that of a normal distribution, whose PDF is defined as following:

$$f(x) = \frac{1}{\sigma\sqrt{2\pi}} \exp\left(-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2\right)$$

To equate this to our model, we need to divide the normal distribution into five sections of equal width to represent the five directions of movement, centered at $x = 0$. The normal distribution has domain all real numbers, which cannot be evenly divided by five.

Therefore, we require a truncated normal distribution, which is a normal distribution that is restricted to lie within a finite domain. The formula for a truncated normal distribution in terms of the normal distribution is as follows:

$$f_t(x) = \frac{\phi(\frac{x-\mu}{\sigma})}{\sigma(\Theta(\frac{b-\mu}{\sigma}) - \Theta(\frac{a-\mu}{\sigma}))}$$

Setting width equal to 50, the truncated normal distribution will have domain from -125 to 125, with each interval of 50 representing the probabilities of moving leftwards, diagonal left-down, downwards, diagonal right-down, and rightwards respectively. In essence, this will represent the flow intensity of the simulation - a larger standard deviation will increase the probabilities for diagonal and sideways travel, and vice versa.

C++ has native support for a normal distribution in the "random" library. Without actually designing a truncated normal distribution, we can simply use the preceding normal distribution from "random" and discard any values outside the desired domain. The resultant probability density function accurately simulates that of a real truncated normal distribution, because the relative probability of each state remains the same. However, this approach grows computationally inefficient as the standard deviation increases.

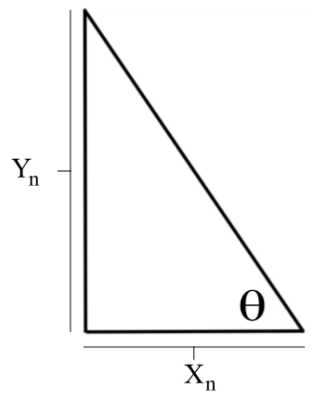
This problem is solved through a repetition-based procedure where we generate 100,000 values from the normal distribution (continuing to discard values outside the desired domain) and track the probabilities of each of the five states. The symmetric states' probabilities are then averaged, and particle movement is determined accordingly with pseudo-random number generator rand()[4].

This approach decreases simulation time by 3.35 times, while maintaining over 99.7% accuracy compared with C++'s native distribution over ten million particle moves.

Findings

Given a fixed membrane height of 50 units and a fixed pore area of 3500 units^2, there are 30 possible membrane geometries that satisfy the above requirements. Following from equation **(1)**, the possible pore geometries are (30, 50, 100), (29, 50, 99), … (1, 50, 71).

Our pore performance optimizing factor Θ, the angle between the horizontal and the slanted membrane wall, is then simply



$$tan(\Theta) = \frac{X_n}{Y_n}$$

$$\Theta = arctan(\frac{X_n}{Y_n})$$

The model measures pore performance in terms of lifetime efficiency β. Let E_x represent a pore's lifetime, the average number of particles spawned before pore closure. Let F_x represent pore efficiency, the percentage of particles trapped before pore closure. Thus,

4. Beware of modulo bias with any pseudo-random generator. Make sure to choose a small modulo m or select m such that $m \mid$ RAND_MAX

$$\beta_x = E_x * F_x$$

Simplifying, this is just proportional to the number of particles trapped by the pore before closure.

Running 500 simulations per each of the 30 possible pore geometries (15000 total simulations), the following graphs show β as a function of Θ.

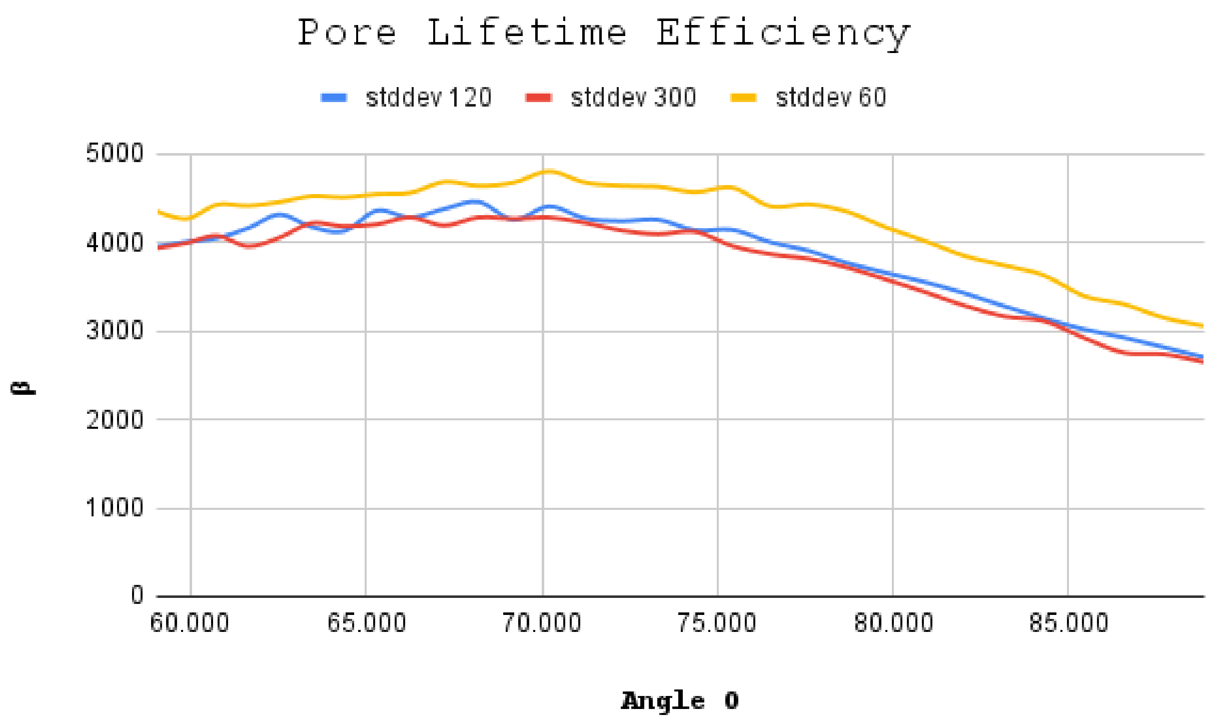


Figure 7. β as Θ increases

Given possible data outliers and that perfectly straight slanted membrane walls are not possible in our current model, the curves are not well defined - a maximum is not always clearly observable. To account for this, β is smoothed using a 3-point moving average function defined as:

$$\beta_x = \frac{(\beta_{x-1} + \beta_x + \beta_{x+1})}{3}$$

The results are shown below.

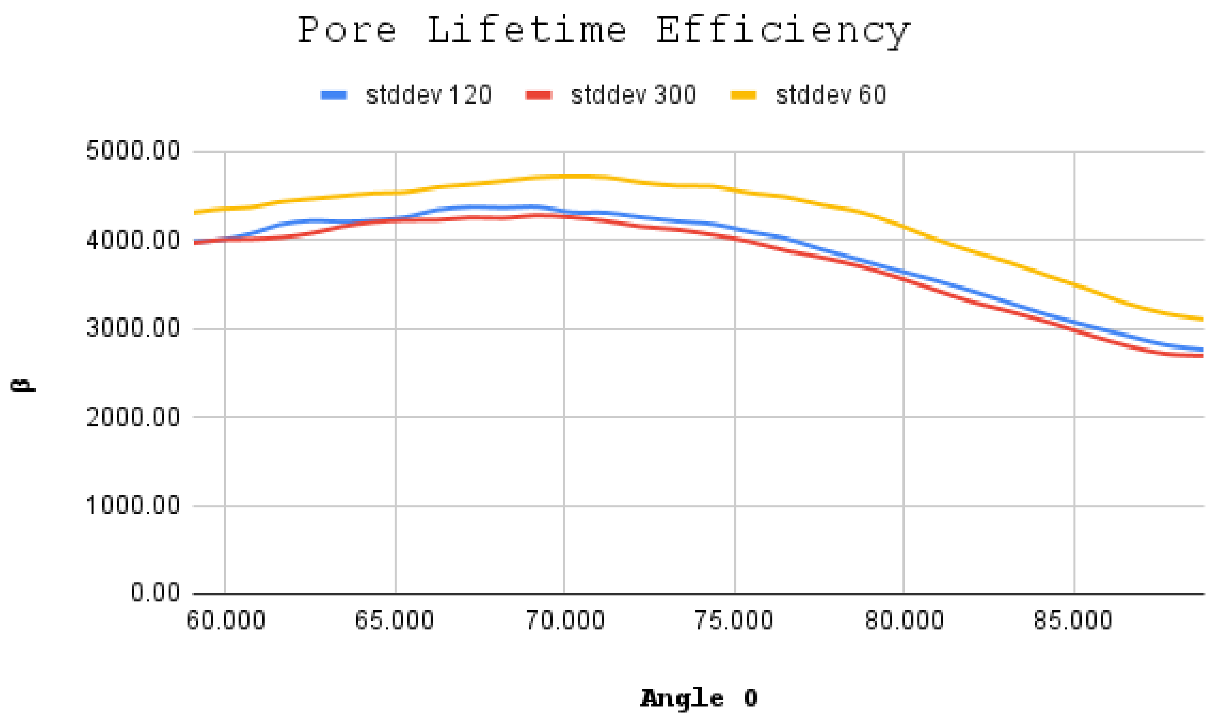


Figure 8. β as Θ increases

Given a fixed membrane height of 50 units and a fixed pore area of 3500 units^2, our model suggests a relatively constant optimum angle for varying flow intensities; on average, with standard deviations ranging from 40 to 300, the optimum angle is **69.193 degrees**. The optimal angle does fluctuate slightly, with a range from **67.218** degrees to **71.222**.

Our model also examined the effects of sticking probabilities on optimal angles. Sticking probability P is defined as the percent chance a moving particle will become stuck after a collision with another stuck particle or the membrane surface. Given the same fixed area and height conditions as above with a standard deviation of 50 for particle movement, the model shows that the optimum angle Θ increases as P decreases.

With P equal to 100%, Θ is 69.193 degrees.

With P equal to 75%, Θ is 71.222 degrees.

With Pequal to 50%, Θ is 74.358 degrees.

With Pequal to 25%, Θ is 75.426 degrees.

Current Model Limitations

The current model is solely two-dimensional, so there is necessary work to be done with three-dimensional computer based simulations. Three dimensional analysis will require a different particle closure detection algorithm than the one implemented in our model, however. Our strict definition of closure (§ 2) may also need to be broadened; to satisfy that no possible path exists from outside the pore to below the pore, the simulation time and particle aggregate may be too large. Instead, a relaxed definition of closure could be related to particle density within a specified volume, or some measurement of fluid flux through the pore itself.

The current model also assumes a perfectly stable particle flow; each particle is only generated after the previous particle has finished its path. To better realize particle filtration, the model should be able to handle multiple moving particles at once. One perspective to address this would be to clock and release particles according to a clock. For example, given that each particle(s) move takes 1 unit of time, particles are released every 100 units of time.

Conclusion

This paper outlines the stochastic simulation methodology and the model's findings within the context of particle filtration. While the research goals have specified a constant pore area and constant pore height, the written computational model can be easily adapted to analyze the performance of all possible 2D geometries. The complete code and algorithms can be found here