Fluid Solver for Incompressible Flow in a Rectangular Domain with Arbitrarily-placed Inclusions

A Senior Honors Thesis

Presented to

The Department of Mathematics
Brandeis University

Department of Mathematics
Bachelor of Science in Applied Mathematics

Dr. Thomas Fai, Advisor

In Partial Fulfillment
of the Requirements for the Degree
BS in Applied Mathematics

by

Swaminathan Lamelas

May 2022

Fluid Solver for Incompressible Flow in a Rectangular Domain with Arbitrarily-placed Inclusions

A thesis presented to the Department of Mathematics, Brandeis University
Waltham, Massachusetts

By Swaminathan Lamelas

Microfluidic devices are used to process fluids at a small scale. Some of these devices contain physical obstacles that are used to manipulate the flow of fluid through the device. In our work, we implement a Python program for approximating the velocity of an incompressible fluid over a rectangular domain using the Navier-Stokes equations. Our strategy, which is built primarily on common finite difference methods, allows users to easily add obstacles into the domain and observe their effects on fluid flow. In the first half of the thesis, we describe the derivation, implementation, and benchmarking of our strategy. In the latter half, we describe some numerical experiments that we performed with our program to observe the effects of inclusions on both fluid flow and hydraulic resistance. We compare the hydraulic resistance reported by our program with an analytic approximation. Lastly, we briefly mention some possible expansions on our work.

This Senior Honors thesis, directed and approved by Swaminathan Lamelas' Committee, has been accepted

and

approved by

the Department of Mathematics at Brandeis University

in partial fulfillment of the requirements for the degree of: BS Applied Mathematics

---

An Huang, Undergraduate Advising Head

Dissertation Committee: Dr. Thomas Fai, Dr. An Huang

Printed Name	Signature
Printed Name	Signature

1 Introduction

Microfluidic devices are used in a variety of disciplines as a way to process particles in a fluid. The study of fluids at small volumes via these devices has been of interest in both academia and industry for quite some time and has recently grown in popularity. These devices have been primarily used in the research and development of commercial products in the life sciences as opposed to being stand alone products themselves [10]. "Hydrodynamic self-focusing in a parallel microfluidic device through cross-filtration" by Torino et. al. provides an example use of microfluidic devices [14]. In their work, Torino et. al. create a cross-filter by placing obstacles in a microfluidic device that manipulate the flow of a fluid in such a way that certain particles are separated from the fluid. A cross-filter is a type of filter that is less likely to become clogged when filtering out particles from a surrounding fluid.

Kelly and Fai study blood cell clogging in certain microfluidic devices [8]. They also are interested in a related problem that inspired this thesis: model the flow of an incompressible fluid through a rectangular domain with rectangular inclusions placed throughout. Inclusions are obstacles in the domain that the fluid must flow around. In this thesis, we implement a Python program that numerically approximates the fluid velocity using the Navier-Stokes equations for incompressible fluids [7]. Our primary focus in building this program is providing users with the ability to place inclusions in the domain as arbitrarily as possible.

In section 1.1, we introduce the particular boundary value problem we solve to obtain approximations for the velocity on the domain. In section 2, we discuss the implementation of our solver for Poisson's equation [11]. This is necessary because our strategy for approximating the solution to the Navier-Stokes equations involves approximating the solution to Poisson's equation. Our strategy for approximating the solution to Poisson's equation reduces to a solving a matrix system constructed using common finite difference techniques. In section 3, we describe and benchmark our fluid velocity solver which uses an iterative strategy to solve the Navier-Stokes equations in the presence of inclusions. This solver is built upon our Poisson solver. We also discuss evidence of unresolved inaccuracies with our velocity solver.

In section 4, we discuss a collection of numerical experiments that use our velocity solver to analyze the effects of inclusion arrangement on fluid flow. 2 of our experiments examine how the proximity and number of inclusions affect the fluid flow. Others such as Eluru et. al. look into the effects of these variations when studying particle clogging [3]. 2 of our other experiments examine the effects of placing inclusions into 2 columns and staggering them. Staggering refers to the horizontal offsetting of the inclusions in the 2 columns. To see an example of aligned and staggered inclusions, see Figures 15 and 16 in section 4.2 where these experiments are discussed in detail. Analysis of the impact of staggered inclusions on fluid flow is related to the work done by Torino et. al. [14] and Bacchin et. al [1]. In section 4, we also compare the

hydraulic resistance output by our solver with an analytic approximation that treats fluid flow channels like resistors. By channels, we mean the paths defined by inclusions in the domain that the fluid is forced to travel through. We find that for some inclusion set ups the analytic approximation fails to capture some impacts of the inclusions while our solver is able to. This demonstrates the importance of such a solver beyond providing us with the ability to analyze fluid flow using streamline plots derived from velocity approximations. We end the thesis describing some possible expansions on our work. This includes directions for improving our solver in accuracy and efficiency as well as incorporating our solver with the Immersed Boundary Method [12] to model the movement of blood cells in a fluid.

1.1 Problem Formulation

The incompressible Navier-Stokes equations are partial differential equations that describe the flow of incompressible fluids. As stated in "Finite Difference Schemes for Incompressible Flow Based on Local Pressure Boundary Conditions" by Johnston and Liu, the incompressible Navier-Stokes equations may be written as follows [7]. Incompressibility is enforced by the pressure Poisson equation.

$$\rho\left(\frac{d\mathbf{u}}{dt} + (\mathbf{u}\cdot\nabla)\mathbf{u}\right) + \nabla p = \mu\nabla\mathbf{u} \quad \text{(momentum equation)}$$
$$\Delta p = 2\rho\left(\frac{du}{dx}\frac{dv}{dy} - \frac{du}{dy}\frac{dv}{dx}\right) \quad \text{(pressure Poisson equation)}$$

ρ is the density and μ is the dynamic viscosity. $\mathbf{u} = \mathbf{u}(x, y, t)$ is the velocity and has a horizontal component $u = u(x, y, t)$ and a vertical component $v = v(x, y, t)$. $p = p(x, y, t)$ denotes the pressure. As mentioned previously, we are interested in fluid flow on a 2-dimensional rectangular domain with certain rectangular regions of the domain removed. Let the domain be defined as $D = [x_l, x_h] \times [y_l, y_h]$. The regions of D that are removed are known as inclusions. At time t and point $(x, y) \in D$ that lies outside of an inclusion, $u(x, y, t)$ denotes the horizontal velocity, $v(x, y, t)$ the vertical velocity, and $p(x, y, t)$ the pressure. The goal of our fluid velocity solver is to approximate the solution $\mathbf{u}(x, y, t)$ to the above equations under the following boundary conditions. There are no slip boundary conditions on the low and high y boundaries:

$$u(x, y_l, t) = 0 \qquad v(x, y_l, t) = 0 \tag{1}$$
$$u(x, y_h, t) = 0 \qquad v(x, y_h, t) = 0 \tag{2}$$

Since the fluid of study is incompressible, the divergence of the velocity $\nabla \cdot \mathbf{u}(x, y, t)$ is 0 over all of D and t. Hence we can add the boundary conditions on the left and right domain boundaries:

$$p(x_l, y, t) = p_l \quad v(x_l, y, t) = 0 \quad \nabla \cdot \mathbf{u}(x_l, y, t) = 0$$
$$p(x_h, y, t) = p_r \quad v(x_h, y, t) = 0 \quad \nabla \cdot \mathbf{u}(x_h, y, t) = 0$$

p_l and p_r, the pressure on the left and right boundaries, are included to enforce a pressure drop from left to right on D. Since $v(x_l, y, t) = v(x_h, y, t) = 0$ for all $y \in [y_l, y_h]$, we know $\frac{dv}{dy}(x_l, y, t) = \frac{dv}{dy}(x_h, y, t) = 0$ on the same y-range. Therefore, the boundary conditions on the low and high x boundaries can be simplified to:

$$p(x_l, y, t) = p_l \quad v(x_l, y, t) = 0 \quad \frac{du}{dx}(x_l, y, t) = 0 \tag{3}$$
$$p(x_h, y, t) = p_r \quad v(x_h, y, t) = 0 \quad \frac{du}{dx}(x_h, y, t) = 0 \tag{4}$$

In the event we have inclusions in the domain, we have no slip boundary conditions on all the boundaries of the inclusions. That is, for an inclusion spanning $[x'_l, x'_h] \times [y'_l, y'_h]$ we have:

$$\mathbf{u}(x'_l, y, t) = 0 \qquad \mathbf{u}(x'_h, y, t) = 0 \quad \text{for } y \in [y'_l, y'_h]$$
$$\mathbf{u}(x, y'_l, t) = 0 \qquad \mathbf{u}(x, y'_h, t) = 0 \quad \text{for } x \in [x'_l, x'_h]$$

In section 3.1, we formally describe an iterative strategy for approximating the solution to this boundary value problem. This strategy requires the ability to approximate a solution to Poisson's equation $\Delta p(x, y) = f(x, y)$ on a rectangular domain. For now, we take $f(x, y)$ to be a general function from $\mathbb{R}^2$ to $\mathbb{R}$. In section 3 we describe what $f(x, y)$ is in the context of our velocity solver.

2 Poisson Solver

In this section we describe the implementation of our Poisson solver which approximates a discrete solution to $\Delta p(x, y) = f(x, y)$ on a rectangular domain with rectangular inclusions. In this section, we explain how such an approximation is derived, implemented, and benchmarked. In sections 2.1 and 2.3 we take $f(x, y)$ to be a general function as noted above. In sections 2.2 and 2.4, $f(x, y)$ is specified in our benchmarks.

2.1 Derivation of Solver for a Rectangular Domain

To start, suppose we want to solve for $p(x, y)$ such that $\Delta p(x, y) = f(x, y)$ on the interior of a domain $D = [x_l, x_h] \times [y_l, y_h]$. The boundary conditions along the boundaries of D are allowed be Neumann or

Dirichlet. For the moment, we assume there are no inclusions in the domain. We discuss inclusions in section 2.3. Formally, we want to approximate the solution $p(x, y)$ such that:

$$\Delta p(x, y) = f(x, y) \tag{5}$$

$$(x, y) \in D = [x_l, x_h] \times [y_l, y_h] \tag{6}$$

$$p(x, y_l) = p_0(x, y_l) \quad \left(\text{or } \frac{\partial p}{\partial y}(x, y_l) = p_0(x, y_l)\right) \tag{7}$$

$$p(x, y_h) = p_1(x, y_h) \quad \left(\text{or } \frac{\partial p}{\partial y}(x, y_h) = p_1(x, y_h)\right) \tag{8}$$

$$p(x_l, y) = p_2(x_l, y) \quad \left(\text{or } \frac{\partial p}{\partial x}(x_l, y) = p_2(x_l, y)\right) \tag{9}$$

$$p(x_h, y) = p_3(x_h, y) \quad \left(\text{or } \frac{\partial p}{\partial x}(x_h, y) = p_3(x_h, y)\right) \tag{10}$$

In equations (7) through (10), p_0, p_1, p_2 and p_3 are known functions that can be evaluated at points (x, y) on the boundaries they correspond to. The first step to deriving a discrete approximation to $p(x, y)$ subject to these conditions is to discretize D into a non-staggered rectangular grid. Let $h \in \mathbb{R}_{>0}$ define the space between points in the grid in both the x and y directions. h should evenly divide both $y_h - y_l$ and $x_h - x_l$. The discretization of D is given as the following set:

$$\{\big(x_l + (j+1)h, y_l + (i+1)h\big) : i = -1, 0, 1, \ldots, N_r \text{ and } j = -1, 0, 1, \ldots, N_c\} \tag{11}$$

N_r gives the number of interior grid rows and N_c gives the number of interior grid columns. Restricting $i = 0, 1, \ldots, N_r - 1$ and $j = 0, 1, \ldots, N_c - 1$ yields a subset of the grid points which form the interior grid: a discretization of the interior of D. N_r and N_c are defined in terms of h and the bounds of D:

$$N_r = \frac{y_h - y_l}{h} - 1 \qquad N_c = \frac{x_h - x_l}{h} - 1$$

Thus, by increasing h, the number of grid points increases, meaning $p(x, y)$ is approximated for more points in D yielding a better approximation of the true behavior of $p(x, y)$. Let $p_{i,j}$ denote the approximation of $p(x_l + (j+1)h, y_l + (i+1)h)$. The solver calculates $p_{i,j}$ for grid points $\{(i, j) : i = 0, 1, \ldots, N_r - 1 \text{ and } j = 0, 1, \ldots, N_c - 1\}$.

The second step to deriving a discrete approximation to $p(x, y)$ is to approximate the constraints on it. Most importantly, $\Delta p(x, y) = f(x, y)$. To do this, we need to use an approximation of the Laplacian $\Delta(\cdot)$ that makes use of the grid constructed above to approximate $\Delta p(x, y)$ at an (x, y) that lies on the grid. For

$g(x,y)$, $\frac{\partial g}{\partial x}$ can be approximated using a second order finite centered difference method as follows:

$$\frac{\partial g}{\partial x} \approx \frac{g(x+\delta x, y) - g(x-\delta x, y)}{2\delta x} \tag{12}$$

δx represents a step size that discretizes the x-axis. Here we apply the single variable centered difference method [5] to approximate a partial derivative. We can similarly approximate $\frac{\partial^2 g}{\partial x^2}$ with second order accuracy based off the centered difference method for approximating the second derivative for a single variable [5]:

$$\frac{\partial^2 g}{\partial x^2}(x,y) \approx \frac{g(x+\delta x, y) - 2g(x,y) + g(x-\delta x, y)}{\delta x^2} \tag{13}$$

(13) can be applied to approximate Δg with second order accuracy assuming that δx is our step size on both x and y. This approximation is referred to as the 5-point stencil approximation for the Laplacian of $p(x,y)$ [11].

$$\Delta g(x,y) = \frac{\partial^2 g}{\partial x^2} + \frac{\partial^2 g}{\partial y^2} \approx \frac{g(x+\delta x, y) + g(x-\delta x, y) - 4g(x,y) + g(x, y+\delta x) + g(x, y-\delta x)}{\delta x^2} \tag{14}$$

Recall that our task is to approximate $p(x,y)$ for interior grid point (x,y) for the grid defined by (11) such that $\Delta p(x,y) = f(x,y)$. If (x,y) lies on the interior of the grid, we have that $(x+h,y), (x-h,y), (x,y+h)$ and $(x,y-h)$ all fall in the set defined by (11). In particular, if $p_{i,j}$ is the approximation for $p(x,y)$, then we have the following approximations for $p(x+h,y), p(x-h,y), p(x,y+h)$ and $p(x,y-h)$:

$$p_{i,j+1} \approx p(x+h,y) \qquad p_{i,j-1} \approx p(x-h,y) \qquad p_{i+1,j} \approx p(x,y+h) \qquad p_{i-1,j} \approx p(x,y-h) \tag{15}$$

We refer to $(i-1,j), (i+1,j), (i,j-1)$ and $(i,j+1)$ as the neighbors of (i,j). Applying equation (14) to interior grid position (x,y), with $\delta x = h$ as defined above, we have the following approximation stencil using the definitions in (15):

$$\Delta p(x,y) \approx \frac{p_{i-1,j} + p_{i+1,j} - 4p_{i,j} + p_{i,j-1} + p_{i,j+1}}{h^2} \tag{16}$$

The problem of approximating $p(x,y)$ that satisfies equation (5) has been reduced to finding $p_{i,j}$ for $i = 0, 1, \ldots, N_r - 1$ and $j = 0, 1, \ldots, N_c - 1$ that satisfy:

$$p_{i-1,j} + p_{i+1,j} - 4p_{i,j} + p_{i,j-1} + p_{i,j+1} = h^2 f(x_l + (j+1)h, y_l + (i+1)h) \tag{17}$$

It is important to note here that while $p_{i,j}$ is an unknown we are trying to solve for, this is not necessarily true for p_{n_i,n_j} for the neighbors (n_i,n_j) of (i,j). Each case where (n_i,n_j) is not an unknown is described

below. First, we introduce 2 more approximations of $\frac{\partial g}{\partial x}$ for $g(x, y)$:

$$\frac{\partial g}{\partial x}(x, y) \approx \frac{g(x, y) - g(x - \delta x, y)}{\delta x} \tag{18}$$

$$\frac{\partial g}{\partial x}(x, y) \approx \frac{g(x + \delta x, y) - g(x, y)}{\delta x} \tag{19}$$

These are modifications of the first order backward and forward finite difference approximations of a derivative for a single variable respectively [5]. These equations are used shortly. First, consider equation (17) when $(i, j) = (0, 0)$. Equation (17) becomes:

$$p_{-1,0} + p_{1,0} - 4p_{0,0} + p_{0,-1} + p_{0,1} = h^2 f(x_l + h, y_l + h)$$

$p_{-1,0}$ and $p_{0,-1}$ do not correspond to interior grid points. That is, they are not values that are calculated by our Poisson solver. Hence, they need to be replaced by known values in some way. If the boundary condition given by equation (7) is Dirichlet, then $p_{-1,0}$ can be set exactly to $p_0(x_l + h, y_l)$. Similarly, if the boundary condition given by (9) is also Dirichlet, $p_{0,-1}$ can be set to $p_2(x_l, y_l + h)$. Equations (7) and (9) could alternatively be Neumann conditions. We cannot set $p_{-1,0}$ and $p_{0,-1}$ as above because $p(x_l + h, y_l)$ and $p(x_l, y_l + h)$ are not known in this case. Observe that the following approximations can be made using (19):

$$\begin{aligned}
\frac{\partial p}{\partial y}(x_l + h, y_l) = p_0(x_l + h, y_l) &\approx \frac{p(x_l + h, y_l + h) - p(x_l + h, y_l)}{h} \approx \frac{p_{0,0} - p_{-1,0}}{h} \\
\implies p_{-1,0} - p_{0,0} &\approx -hp_0(x_l + h, y_l) \\
\frac{\partial p}{\partial x}(x_l, y_l + h) = p_2(x_l, y_l + h) &\approx \frac{p(x_l + h, y_l + h) - p(x_l, y_l + h)}{h} \approx \frac{p_{0,0} - p_{0,-1}}{h} \\
\implies p_{0,-1} - p_{0,0} &\approx -hp_2(x_l, y_l + h)
\end{aligned}$$

Hence, when $(i, j) = (0, 0)$ we have 4 possible equations:

$$\begin{aligned}
p_0(x_l + h, y_l) + p_{1,0} - 4p_{0,0} + p_2(x_l, y_l + h) + p_{0,1} &= h^2 f(x_l + h, y_l + h) \\
p_0(x_l + h, y_l) + p_{1,0} - 3p_{0,0} - hp_2(x_l, y_l + h) + p_{0,1} &= h^2 f(x_l + h, y_l + h) \\
-hp_0(x_l + h, y_l) + p_{1,0} - 3p_{0,0} + p_2(x_l, y_l + h) + p_{0,1} &= h^2 f(x_l + h, y_l + h) \\
-hp_0(x_l + h, y_l) + p_{1,0} - 2p_{0,0} - hp_2(x_l, y_l + h) + p_{0,1} &= h^2 f(x_l + h, y_l + h)
\end{aligned}$$

To match the pattern of (17), it is more convenient to write these in such a way that the left hand side of

these equations are solely made up of $p_{i,j}$ for interior grid positions (i, j).

$$p_{1,0} - 4p_{0,0} + p_{0,1} = h^2 f(x_l + h, y_l + h) - p_0(x_l + h, y_l) - p_2(x_l, y_l + h) \tag{20}$$

$$p_{1,0} - 3p_{0,0} + p_{0,1} = h^2 f(x_l + h, y_l + h) - p_0(x_l + h, y_l) + hp_2(x_l, y_l + h) \tag{21}$$

$$p_{1,0} - 3p_{0,0} + p_{0,1} = h^2 f(x_l + h, y_l + h) + hp_0(x_l + h, y_l) - p_2(x_l, y_l + h) \tag{22}$$

$$p_{1,0} - 2p_{0,0} + p_{0,1} = h^2 f(x_l + h, y_l + h) + hp_0(x_l + h, y_l) + hp_2(x_l, y_l + h) \tag{23}$$

The next grid position we consider is $(N_r - 1, N_c - 1)$. Equation (17) becomes:

$$p_{N_r-2,N_c-1} + p_{N_r,N_c-1} - 4p_{N_r-1,N_c-1} + p_{N_r-1,N_c-2} + p_{N_r-1,N_c} = h^2 f(x_l + N_c h, y_l + N_r h)$$

p_{N_r,N_c-1} and p_{N_r-1,N_c} do not correspond to interior grid points. Similar to the derivation for the equation for $p_{0,0}$, if equations (8) and (10) are Dirichlet boundary conditions we set $p_{N_r,N_c-1} = p_1(x_l + N_c h, y_h)$ and $p_{N_r-1,N_c} = p_3(x_h, y_l + N_r h)$. In the case of equations (8) and (10) being Neumann conditions, we cannot set p_{N_r,N_c-1} and p_{N_r-1,N_c} because $p(x_l + N_c h, y_h)$ and $p(x_h, y_l + N_r h)$ are not known. The following observations can be made using (18):

$$\frac{\partial p}{\partial y}(x_l + N_c h, y_h) = p_1(x_l + N_c h, y_h) \approx \frac{p(x_l + N_c h, y_h) - p(x_l + N_c h, y_l + N_r h)}{h} \approx \frac{p_{N_r,N_c-1} - p_{N_r-1,N_c-1}}{h}$$

$$\implies p_{N_r,N_c-1} - p_{N_r-1,N_c-1} \approx hp_1(x_l + N_c h, y_h)$$

$$\frac{\partial p}{\partial x}(x_h, y_l + N_r h) = p_3(x_h, y_l + N_r h) \approx \frac{p(x_h, y_l + N_r h) - p(x_l + N_c h, y_l + N_r h)}{h} \approx \frac{p_{N_r-1,N_c} - p_{N_r-1,N_c-1}}{h}$$

$$\implies p_{N_r-1,N_c} - p_{N_r-1,N_c-1} \approx hp_3(x_h, y_l + N_r h)$$

Thus, when $(i, j) = (N_r - 1, N_c - 1)$ we have 4 possible equations arranged to match the format of (17):

$$p_{N_r-2,N_c-1} - 4p_{N_r-1,N_c-1} + p_{N_r-1,N_c-2} = h^2 f(x_l + N_c h, y_l + N_r h) - p_1(x_l + N_c h, y_h) - p_3(x_h, y_l + N_r h) \tag{24}$$

$$p_{N_r-2,N_c-1} - 3p_{N_r-1,N_c-1} + p_{N_r-1,N_c-2} = h^2 f(x_l + N_c h, y_l + N_r h) - p_1(x_l + N_c h, y_h) - hp_3(x_h, y_l + N_r h) \tag{25}$$

$$p_{N_r-2,N_c-1} - 3p_{N_r-1,N_c-1} + p_{N_r-1,N_c-2} = h^2 f(x_l + N_c h, y_l + N_r h) - hp_1(x_l + N_c h, y_h) - p_3(x_h, y_l + N_r h) \tag{26}$$

$$p_{N_r-2,N_c-1} - 2p_{N_r-1,N_c-1} + p_{N_r-1,N_c-2}$$
$$= h^2 f(x_l + N_c h, y_l + N_r h) - h p_1(x_l + N_c h, y_h) - h p_3(x_h, y_l + N_r h) \tag{27}$$

The strategies used to derive the equations for $(i,j) = (0,0)$ and $(i,j) = (N_r - 1, N_c - 1)$ can be used to derive the equations for any (i,j) that has neighbors who lie on a boundary. The strategy for constructing the equation for $p_{i,j}$ can be summarized as follows:

1. Start with equation (17).
2. In the case that (i,j) has a neighbor (n_i, n_j) that lies on a boundary with a Dirichlet condition, replace p_{n_i,n_j} in the stencil with the boundary value at $(x_l + (n_j+1)h, y_l + (n_i+1)h)$ pertaining to (i,j) .
3. If (n_i, n_j) lies on a boundary with a Neumann condition there are 2 cases. If $n_i > i$ or $n_j > j$, then approximate $p_{n_i,n_j} - p_{i,j}$ to be h multiplied with the boundary value at $(x_l + (n_j+1)h, y_l + (n_i+1)h)$ pertaining to (i,j) using the backward difference method given by (18). If $n_i < i$ or $n_j < j$, then approximate $p_{i,j} - p_{n_i,n_j}$ to be h multiplied with the boundary value at $(x_l + (n_j+1)h, y_l + (n_i+1)h)$ pertaining to (i,j) using the forward difference method given by (19).
4. Lastly, arrange the equation for $p_{i,j}$ so that only the unknowns are on the left hand side.

Now, let us consider the other 2 corner interior grid points. There are 4 possible equations for $(0, N_c - 1)$. Following the strategy above, we use (19) in the case that equation (7) is a Neumann condition and (18) in the case that equation (10) is a Neumann condition.

$$p_{1,N_c-1} - 4p_{0,N_c-1} + p_{0,N_c-2} = h^2 f(x_l + N_c h, y_l + h) - p_0(x_l + N_c h, y_l) - p_3(x_h, y_l + h) \tag{28}$$
$$p_{1,N_c-1} - 3p_{0,N_c-1} + p_{0,N_c-2} = h^2 f(x_l + N_c h, y_l + h) - p_0(x_l + N_c h, y_l) - h p_3(x_h, y_l + h) \tag{29}$$
$$p_{1,N_c-1} - 3p_{0,N_c-1} + p_{0,N_c-2} = h^2 f(x_l + N_c h, y_l + h) + h p_0(x_l + N_c h, y_l) - p_3(x_h, y_l + h) \tag{30}$$
$$p_{1,N_c-1} - 2p_{0,N_c-1} + p_{0,N_c-2} = h^2 f(x_l + N_c h, y_l + h) + h p_0(x_l + N_c h, y_l) - h p_3(x_h, y_l + h) \tag{31}$$

Similarly for $(N_r - 1, 0)$, we use (19) in the case that equation (9) is a Neumann condition and (18) in the case that equation (8) is a Neumann condition.

$$p_{N_r-2,0} - 4p_{N_r-1,0} + p_{N_r-1,1} = h^2 f(x_l + h, y_l + N_r h) - p_1(x_l + h, y_h) - p_2(x_l, y_l + N_r h) \tag{32}$$
$$p_{N_r-2,0} - 3p_{N_r-1,0} + p_{N_r-1,1} = h^2 f(x_l + h, y_l + N_r h) - p_1(x_l + h, y_h) + h p_2(x_l, y_l + N_r h) \tag{33}$$
$$p_{N_r-2,0} - 3p_{N_r-1,0} + p_{N_r-1,1} = h^2 f(x_l + h, y_l + N_r h) - h p_1(x_l + h, y_h) - p_2(x_l, y_l + N_r h) \tag{34}$$

$$p_{N_r-2,0} - 2p_{N_r-1,0} + p_{N_r-1,1} = h^2 f(x_l + h, y_l + N_r h) - hp_1(x_l + h, y_h) + hp_2(x_l, y_l + N_r h) \tag{35}$$

For interior points $(0, j)$ for $j = 1, \ldots, N_c - 2$, only one of the stencil points, $(-1, j)$, lies on the boundary. Thus, there are 2 possible equations for these points. We use (19) in the case that (7) is a Neumann condition.

$$p_{1,j} - 4p_{0,j} + p_{0,j-1} + p_{0,j+1} = h^2 f(x_l + (j+1)h, y_l + h) - p_0(x_l + (j+1)h, y_l) \tag{36}$$

$$p_{1,j} - 3p_{0,j} + p_{0,j-1} + p_{0,j+1} = h^2 f(x_l + (j+1)h, y_l + h) + p_0(x_l + (j+1)h, y_l) \tag{37}$$

For interior points $(N_r - 1, j)$ for $j = 1, \ldots, N_c - 2$, only one of the stencil points, (N_r, j), lies on the boundary. Thus, there are 2 possible equations for these points. We use (18) in the case that (8) is a Neumann condition.

$$p_{N_r-2,j} - 4p_{N_r-1,j} + p_{N_r-1,j-1} + p_{N_r-1,j+1} = h^2 f(x_l + (j+1)h, y_l + N_r h) - p_1(x_l + (j+1)h, y_h) \tag{38}$$

$$p_{N_r-2,j} - 3p_{N_r-1,j} + p_{N_r-1,j-1} + p_{N_r-1,j+1} = h^2 f(x_l + (j+1)h, y_l + N_r h) - hp_1(x_l + (j+1)h, y_h) \tag{39}$$

For interior points $(i, 0)$ for $i = 1, \ldots, N_r - 2$, only one of the stencil points, $(i, -1)$, lies on the boundary. Thus, there are 2 possible equations for these points. We use (19) in the case that (9) is a Neumann condition.

$$p_{i-1,0} + p_{i+1,0} - 4p_{i,0} + p_{i,1} = h^2 f(x_l + h, y_l + (i+1)h) - p_2(x_l, y_l + (i+1)h) \tag{40}$$

$$p_{i-1,0} + p_{i+1,0} - 3p_{i,0} + p_{i,1} = h^2 f(x_l + h, y_l + (i+1)h) + hp_2(x_l, y_l + (i+1)h) \tag{41}$$

For interior points $(i, N_c - 1)$ for $i = 1, \ldots, N_r - 2$, only one of the stencil points, (i, N_c), lies on the boundary. Thus, there are 2 possible equations for these points. We use (18) in the case that (10) is a Neumann condition.

$$p_{i-1,N_c-1} + p_{i+1,N_c-1} - 4p_{i,N_c-1} + p_{i,N_c-2} = h^2 f(x_l + N_c h, y_l + (i+1)h) - p_3(x_h, y_l + (i+1)h) \tag{42}$$

$$p_{i-1,N_c-1} + p_{i+1,N_c-1} - 3p_{i,N_c-1} + p_{i,N_c-2} = h^2 f(x_l + N_c h, y_l + (i+1)h) - hp_3(x_h, y_l + (i+1)h) \tag{43}$$

Lastly, we have the interior grid points who have no neighbors on the boundary. For these points, we can just use equation (17).

Let equation (17) and equations (20) through (43) be known as the equations for $p_{i,j}$ which are used to approximate $\Delta p(x, y) = f(x, y)$. Observe that in these equations there is a linear relationship between our $N_r N_c$ unknowns $\{p_{i,j} : i = 0, 1, \ldots, N_r - 1 \text{ and } j = 0, 1, \ldots, N_c - 1\}$. Since we have one of these equations for each $p_{i,j}$ we have $N_r N_c$ linear equations for the same number unknowns. Thus, our approximation for $p(x, y)$ that satisfies equations (5) through (10) has been reduced to solving a linear system. The next step

is to set up this system as a matrix equation $A\mathbf{p} = \mathbf{b}$. This allows for the use of various built in solvers to calculate $p_{i,j}$.

$\mathbf{p}$ is a vector of $p_{i,j}$ stacked columnwise. That is, $p_{i,j}$ is stored in $\mathbf{p}_{k(i,j)}$ for $k(i,j)$ given by:

$$k(i,j) = jN_r + i \tag{44}$$

The left hand side for any one of the equations for $p_{i,j}$ can be expressed as a product $\mathbf{a^{k(i,j)}}^T\mathbf{p}$. $\mathbf{a^{k(i,j)}}_{k(i',j')}$ is set to the coefficient of $p_{i',j'}$ in the equation for $p_{i,j}$ for all interior grid points (i',j'). For example, the left hand side of equation (17) can be expressed as $\mathbf{a^{k(i,j)}}^T\mathbf{p}$ where $\mathbf{a^{k(i,j)}}$ is 0 except at the following indices:

$$\mathbf{a^{k(i,j)}}_{k(i-1,j)} = 1 \qquad \mathbf{a^{k(i,j)}}_{k(i+1,j)} = 1 \qquad \mathbf{a^{k(i,j)}}_{k(i,j)} = -4 \qquad \mathbf{a^{k(i,j)}}_{k(i,j-1)} = 1 \qquad \mathbf{a^{k(i,j)}}_{k(i,j+1)} = 1$$

The left hand side of equations (20) through (43) can be expressed as dot products following a similar strategy. A is constructed as:

$$A = \begin{pmatrix} \mathbf{a^{k(0,0)}}^T \\ \mathbf{a^{k(1,0)}}^T \\ \vdots \\ \mathbf{a^{k(0,1)}}^T \\ \mathbf{a^{k(1,1)}}^T \\ \vdots \\ \mathbf{a^{k(N_r-1,N_c-1)}}^T \end{pmatrix} \in \mathbb{Z}^{N_rN_c \times N_rN_c} \tag{45}$$

Thus, $A\mathbf{p}$ holds the left hand side of the equations for all interior $p_{i,j}$. To complete the matrix equation, collect the right hand side of the equations for $p_{i,j}$ into $\mathbf{b}$. $\mathbf{b}_{k(i,j)}$ holds the right hand side of the equation for $p_{i,j}$. For example, if the equation for $p_{i,j}$ is of the form of equation (43), then $\mathbf{b}_{k(i,j)} = h^2 f(x_l + N_c h, y_l + (i+1)h) - hp_3(x_h, y_l + (i+1)h)$.

$A\mathbf{p} = \mathbf{b}$ can be solved as follows:

1. Perform an LU factorization with complete pivoting [15] on A to get $PAQ = LU$. P and Q are permutation matrices, L is a lower triangular matrix, and U is an upper triangular matrix.

2. Rearrange the above equation making use of the fact that permutation matrix Q is orthogonal and right multiply both sides by $\mathbf{p}$ to get $P\mathbf{b} = LUQ^T\mathbf{p}$.

3. Define $\mathbf{z} := Q^T\mathbf{p}$ and $\mathbf{y} := U\mathbf{z}$.

4. Solve $L\mathbf{y} = P\mathbf{b}$ for $\mathbf{y}$ using forward substitution.

5. Solve $U\mathbf{z} = \mathbf{y}$ for $\mathbf{z}$ using backward substitution.

6. Compute $\mathbf{p} = Q\mathbf{z}$.

The reason why $A\mathbf{p} = \mathbf{b}$ is solved using an LU decomposition is directly tailored to our Poisson solver being used in our velocity solver. This is explained in section 3.1.

Note that in equations (17) and (20) through (43) that the majority of $p_{i,j}$ have a coefficient of 0. This results in A being a very sparse matrix with its nonzero values concentrated near the main diagonal. Figure 3 in section 2.2 shows a particular construction of A. SciPy [16] provides the capability, via the SuperLU [9] library, to solve $A\mathbf{p} = \mathbf{b}$ using an LU factorization with complete pivoting when A is represented as a SciPy sparse matrix. The SuperLU library is quite easy to use in Python. A SuperLU object can be constructed from A represented as a `scipy.sparse.csc_matrix` via the `scipy.sparse.linalg.splu` function. `scipy.sparse.csc_matrix` is one of numerous SciPy classes that represents a sparse matrix. Calling `solve` on a SuperLU object provided $\mathbf{b}$ solves for $\mathbf{p}$ in a manner similar to the steps 3 through 6 described above.

However, there remains the issue of representing A as a `scipy.sparse.csc_matrix`. A is constructed as follows. For each interior grid point (i, j), the $k(i, j)$th row of A is set according to one of equations (17) and (20) through (43) as described on page 10. We test 5 methods that could be used to construct A as a `scipy.sparse.csc_matrix`. We test these 5 methods on constructing the identity matrix with sizes 10,000, 20,000 ... 100,000. That is, we construct the identity matrix setting each nonzero element for each row one at a time to simulate how A would be constructed. We choose the identity matrix because, like A, it is very sparse with its nonzero values near the main diagonal. The 5 methods are:

1. Construct a `numpy.ndarray` of 0s and set the diagonal element of each row to 1. `numpy.ndarray` stores a dense matrix and can be converted into a `scipy.sparse.csc_matrix`. We expect this to perform poorly, especially the conversion.

2. Construct an empty `scipy.sparse.csc_matrix` and set the diagonal element of each row to 1. It is specifically stated on the SciPy documentation of `scipy.sparse.csc_matrix` that one should not change its sparsity in any manner. Thus, we also expect this to be a slow approach.

3. Construct an empty `scipy.sparse.lil_matrix` and set the diagonal element of each row to 1. `scipy.sparse.lil_matrix` is another sparse matrix type offered by SciPy that handles sparsity changes efficiently. `scipy.sparse.lil_matrix` can be converted into a `scipy.sparse.csc_matrix`. We expect this approach to perform well.

4. Represent the identity matrix as 3 `lists`: `rows`, `cols`, and `vals`. For each row, add its index to `rows`, add the row's index on the diagonal to `cols`, and 1 to `vals`. `scipy.sparse.csc_matrix` can be constructed from these lists under the assumption that the value at position (`rows[n]`,`cols[n]`) in the resulting matrix is `vals[n]`. We expect this approach to perform well.

5. Use the same strategy as 4 except have `rows`, `cols`, and `vals` be pre-allocated `numpy.ndarrays`. We expect this approach to be faster than 4 because we expect our 3 `lists` will need to internally resize as we append to them.

Figure 1 shows the runtime comparisons for these 5 strategies. From it we see that strategies 1 and 2 perform very poorly compared to strategies 3, 4, and 5. Figure 2 provides a closer look at the 3 best strategies. From it we can see that the runtime of strategy 3 is $O(\text{size})$ while the runtime of strategies 4 and 5 increases far less for larger sizes. Also, strategy 5 does not noticeably outperform strategy 4 despite our expectations. It is important to note that it is easy to use strategy 5 to construct the identity matrix because the size to pre-allocate the 3 arrays to is known. However, when constructing A, the pre-allocation size is not as easy to derive because the number of nonzero elements is directly dependent on the nature of the boundary conditions. We adopt strategy 4 to initialize A as a `scipy.sparse.csc_matrix` in our solver.

2.2 Benchmarks Against an Existing Solver

We implement the strategy we describe in section 2.1 in a Python class `PoissonSolver`. We initialize a `PoissonSolver` object with x_l, x_h, y_l, y_h, h, and the boundary conditions. The class also provides methods to construct A and $\mathbf{b}$ and solve for $\mathbf{p}$. We benchmark this solver with benchmark problems used by the existing Medusa library [13]. The first benchmark is done on the following problem.

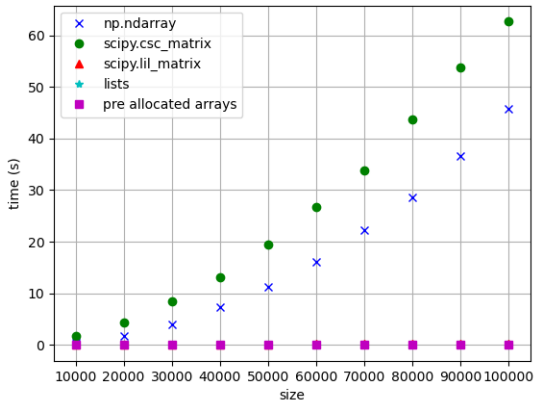


Figure 1: `csc_matrix` Initialization Runtimes

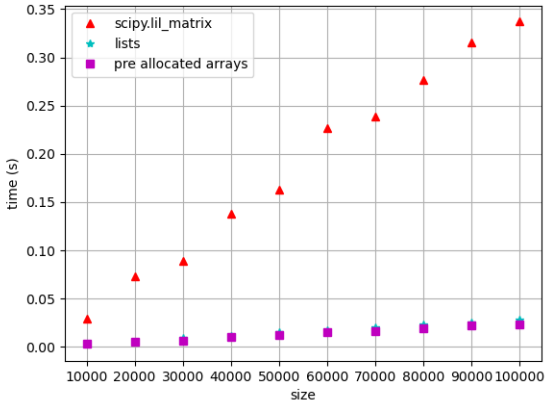


Figure 2: Top 3 `csc_matrix` Initialization Runtimes

Find $p(x, y)$ that satisfies:

$$\Delta p(x, y) = -2\pi^2 \sin(\pi x) \sin(\pi y)$$
$$(x, y) \in D = [0, 1] \times [0, 1]$$
$$p(x, 0) = p(x, 1) = p(0, y) = p(1, y) = 0$$

The solution to this problem is given by $p(x, y) = \sin(\pi x) \sin(\pi y)$. This problem and its solution are taken from the Medusa example page on solving Poisson's equation with their library [6]. Figure 3 shows the appearance of the matrix A that our solver constructs for $h = 0.01$. For $h = 0.01$, D is discretized such that $N_r = N_c = 99$ which means that A is $99^2 \times 99^2$. In the figure, black represents the matrix entries that are nonzero while white represents entries that are zero. We can see that the matrix is very sparse with its nonzero entries concentrated on the main diagonal. Our solver's output matches the solution visually quite well, but we want to ensure that the solver improves its approximation of $p(x, y)$ as h is decreased. We test the solver for $h = 0.005, 0.01, 0.025, 0.05$, and 0.1. That corresponds to D being discretized such that $N_r = N_c =$ 199,99,39,19, and 9. We calculate 2 error metrics for each of these: (i) the mean absolute error of our solver's approximation of the solution $p(x, y)$ over the interior grid points and (ii) the maximum absolute error of our solver's approximation of $p(x, y)$ over the interior grid points. Figures 4 (a) and 4 (b) are log-log plots of the tested values of h versus (i) and (ii) respectively.

The order 1 line in Figure 4 (a) portrays the relationship between h and the mean absolute error for an order 1 method. The order 2 line is the same except for an order 2 method. In Figure 4 (b), the order lines represent the relationship between h and the maximum absolute error for order 1 and 2 methods. Thus, we can see that our solver is close to being an order 2 method for both of our error metrics when applied to

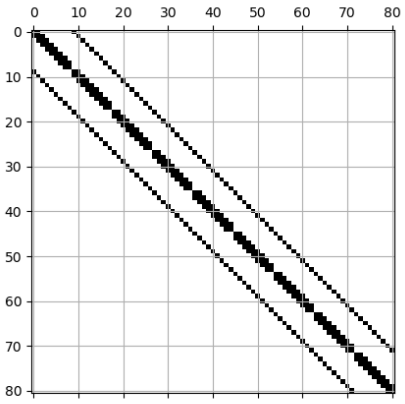
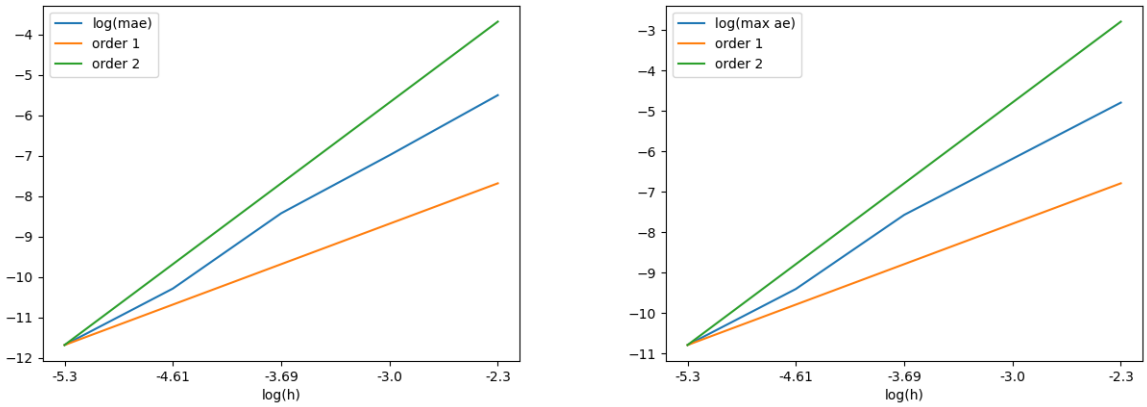


Figure 3: Visualization of an Example A Matrix Constructed by our Poisson Solver



(a) Mean Absolute Error

(b) Maximum Absolute Error

Figure 4: Log-log Plots of Error Metrics for First Medusa Example

this problem. This is expected since the approximations made in the construction of A to solve this particular problem are all second order accurate. The only approximations that are first order in equations (17) and (20) through (43) appear in the equations incorporating Neumann boundary conditions. None of these equations are used in the solver for this particular problem. The second benchmark is done on the second Medusa library Poisson example [6].

Find $p(x, y)$ that satisfies:

$$\begin{aligned} \Delta p(x,y) &= -2\pi^2 \sin(\pi x)\sin(\pi y) \\ (x,y) &\in D = [0,0.5]\times[0,0.5] \\ p(x,0) &= p(0,y) = 0 \\ \frac{\partial p}{\partial y}(x,0.5) &= \frac{\partial p}{\partial x}(0.5,y) = 0 \end{aligned}$$

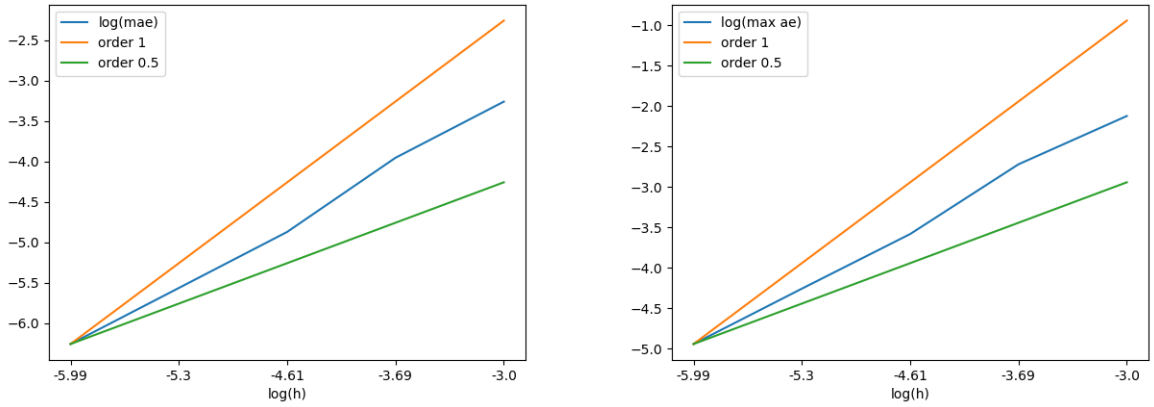
$p(x, y) = \sin(\pi x)\sin(\pi y)$ satisfies this as well as noted in the Medusa example. Our solver's output for $h = 0.01$ ($N_r = N_c = 49$) visually matches the Medusa solution given in the "Mixed boundary conditions" section of the Medusa Poisson examples well [6]. Again, we test how the solver behaves for different h. We calculate the error metrics (i) and (ii) described on the previous page on the solver's approximation for $h = 0.0025, 0.005, 0.01, 0.025$, and 0.05. These step sizes correspond to D being discretized such that $N_r = N_c$ =199,99,49,19, and 9. We plot these metrics against h in Figure 5. Figures 5 (a) and 5 (b) include lines to represent the behavior of order 1 and order 0.5 methods. For both error metrics, we can see that our solver is close to being an order 1 method when applied to this problem. This is expected because, in contrast

to the first example, we have to incorporate Neumann boundary conditions in our derivation of the equations for $p_{i,j}$. As described previously, this involves first order derivative approximations on the boundaries.

2.3 Addition of Inclusions

An inclusion is a rectangular region that is removed from the problem domain. For each inclusion that is added to the problem, 4 new boundaries are created. Thus, the user must specify 4 new boundary conditions when they add an inclusion. Our solver requires that inclusions must have bounds that fall on valid interior grid points, not in between. Furthermore, inclusions cannot be too close to each other or to the boundary of the domain. This is explained more in section 2.4. To account for inclusions, we build upon our strategy from section 2.1. We discretize domain D in the same manner and use equations (17) and (20) through (43) as long as $p_{i,j}$ does not have a neighbor that is on the boundary of an inclusion.

As mentioned previously, some interior points have neighbors that lie on the boundary of an inclusion. We use the general strategy outlined on page 8 for deriving an equation for $p_{i,j}$ to substitute in inclusion boundary values as necessary. The derivation of the rows of A is the same as in section 2.1 for interior grid points that lie outside of inclusions. Recall that A has a row for each interior grid point in our discretization of the domain. For (i, j) that lie in an inclusion, including the boundary of an inclusion, set $\mathbf{a^{k(i,j)}}^T_{k(i,j)} = 1$ and all other elements of $\mathbf{a^{k(i,j)}}^T$ to 0. $\mathbf{b}_{k(i,j)}$ for (i, j) can be set to some constant. In Python, it is convenient to set it to `numpy.nan` for the purposes of plotting the output of our solver. Despite having equations represented in $A\mathbf{p} = \mathbf{b}$ for interior grid points in and outside of inclusions, the equations are independent of each other. The equations for interior grid points not in an inclusion are only dependent on other interior grid points that are not in an inclusion and known values expressed in terms of f and boundary values. Therefore, the



(a) Mean Absolute Error

(b) Maximum Absolute Error

Figure 5: Log-log Plots of Error Metrics for Second Medusa Example

equations for interior grid points in inclusions are independent in this setup.

More boundary conditions have to be managed given that each inclusion has 4 of them. As mentioned previously, this is one of the primary reasons we went with strategy 4 for initializing the `scipy.sparse.csc_matrix` representation of A. Building up 3 lists allows for a more easily understandable implementation. Since we want to maintain the flexibility to incorporate different arrangements of inclusions, we construct A by iterating over each interior grid point. With the addition of inclusions, the set of points $\{p_{i,j} : i = 0, 1, \ldots, N_r - 1 \text{ and } j = 0, 1, \ldots, N_c - 1\}$ produced by our solver is made up of 2 different types of points. For interior grid points (i, j) that do not lie on an inclusion, $p_{i,j}$ is the approximation for $p(x_l + (j+1)h, y_l + (i+1)h)$. For interior grid points (i, j) that lie on an inclusion, $p_{i,j}$ is `numpy.nan` to signify that the p is not calculated at those points.

2.4 Benchmark for a Single Inclusion

Our class `PoissonSolver` is adapted to include a method that allows for the addition of an inclusion given the range of the inclusion and its boundary conditions. We benchmark the updated `PoissonSolver` against the following problem taken from "Discontinuous Galerkin approximations in computational mechanics: hybridization, exact geometry and degree adaptivity" by Giacomini and Sevilla [4].

$$\begin{aligned}
\Delta p(x, y) &= 0 \\
(x, y) &\in D = [-75, 75] \times [-100, 100] - [-50, 50] \times [-50, 50] \\
p(x, -100) = p(x, 100) &= p(-75, y) = p(75, y) = 1 \\
p(x, -50) = p(x, 50) &= 0 \quad \text{for } x \in [-50, 50] \\
p(-50, y) = p(50, y) &= 0 \quad \text{for } y \in [-50, 50]
\end{aligned}$$

In this case, our domain has a single inclusion spanning $[-50, 50] \times [-50, 50]$. When applying our solver to this problem, we need to choose h such that $[-75, 75] \times [-100, 100]$ is discretized in such a way that $x = -50$ and $x = 50$ lie on grid columns and $y = -50$ and $y = 50$ lie on grid rows. Figure 1 in Giacomini and Sevilla's work [4] includes 3 approximations of $\left\|\nabla p(x, y)\right\|_2$. We test our solver on this problem with 3 values of h. However, to compare our solver results with that figure, we need to have a way of approximating $\nabla p(x, y)$ given our solver's approximation of $p(x, y)$.

Recall the form of the set of points $p_{i,j}$ our solver returns as described at the end of the previous section. Consider a set of points $g_{i,j}$ that takes on the same form. That is, it serves as an approximation of a function $g(x, y)$ over a matrix of grid points on the interior of a domain except at inclusions. We want to use $g_{i,j}$ to

approximate $\frac{\partial g}{\partial x}$ and $\frac{\partial g}{\partial y}$ at the interior grid points. Our velocity solver will also need to be able to approximate the gradient of a function using an approximation of the form of $g_{i,j}$.

Assuming that $g_{i,j}$ are spaced out by some spacing h, applying (12) we get:

$$\frac{\partial g}{\partial x}(x_l + (j+1)h, y_l + (i+1)h) \approx \frac{g_{i,j+1} - g_{i,j-1}}{2h} \tag{46}$$

$$\frac{\partial g}{\partial y}(x_l + (j+1)h, y_l + (i+1)h) \approx \frac{g_{i+1,j} - g_{i-1,j}}{2h} \tag{47}$$

However, (46) and (47) cannot always be used. For example, suppose $(i, j+1), (i, j-1), (i+1, j)$, or $(i-1, j)$ lies on a boundary, either of the domain or of an inclusion. We need to derive modified versions of these equations similar to how we derived equations (20) through (43) from equation (17). That is, we must incorporate boundary values. Therefore, to approximate $\frac{\partial g}{\partial x}$ and $\frac{\partial g}{\partial y}$ on a set of interior grid points, we not only need $g_{i,j}$ but also the boundary conditions of the domain and of any contained inclusions.

First, consider the case where $(i, j-1)$ lies on a boundary with boundary value specified $g_1(x, y)$. If there is a Dirichlet condition on the boundary, then replace $g_{i,j-1}$ with $g_1(x_l + jh, y_l + (i+1)h)$ in (46). Thus, we get:

$$\frac{\partial g}{\partial x}(x_l + (j+1)h, y_l + (i+1)h) \approx \frac{g_{i,j+1} - g_1(x_l + jh, y_l + (i+1)h)}{2h} \tag{48}$$

Suppose there is a Neumann condition on the boundary with boundary value of the form $\frac{\partial g}{\partial x}(x, y) = g_1(x, y)$. We approximate $\frac{\partial g}{\partial x}(x_l + (j+1)h, y_l + (i+1)h)$ as an average of the derivatives of the left and right:

$$\begin{aligned}\frac{\partial g}{\partial x}(x_l + (j+1)h, y_l + (i+1)h) &\approx \frac{1}{2}\left(\frac{\partial g}{\partial x}(x_l + (j+2)h, y_l + (i+1)h) + \frac{\partial g}{\partial x}(x_l + jh, y_l + (i+1)h)\right) \\ &\approx \frac{1}{2}\left(\frac{g_{i,j+2} - g_{i,j}}{2h} + g_1(x_l + jh, y_l + (i+1)h)\right)\end{aligned} \tag{49}$$

Here, we use a second order centered finite difference for the derivative to the right in the average. Note that equation (49) assumes that $(i, j+2)$ is an interior grid point. This restricts how close inclusions can be to each other or to the domain boundary. In particular, no interior grid point in an inclusion can be less than 4 rows or columns away from an interior grid point in another inclusion or a grid point on the domain boundary. For a similar reason, we assume that the grid is not so small that $(i, j+2)$ is not a valid interior grid point. In particular, we assume that h is small enough that $N_r, N_c \geq 3$. Following the idea behind equations (48) and (49) we can derive the following derivative approximations.

If $(i, j+1)$ lies on a boundary with boundary value $g_2(x, y)$, equation (50) gives the approximation in the case that the boundary has a Dirichlet condition and equation (51) gives the approximation in the case of a

Neumann condition of the form $\frac{\partial g}{\partial x}(x,y) = g_2(x,y)$.

$$\frac{\partial g}{\partial x}(x_l + (j+1)h, y_l + (i+1)h) \approx \frac{g_2(x_l + (j+2)h, y_l + (i+1)h) - g_{i,j-1}}{2h} \tag{50}$$

$$\frac{\partial g}{\partial x}(x_l + (j+1)h, y_l + (i+1)h) \approx \frac{1}{2}\left(\frac{g_{i,j} - g_{i,j-2}}{2h} + g_2(x_l + (j+2)h, y_l + (i+1)h)\right) \tag{51}$$

The next 4 equations are modifications of (47) instead of (46). If $(i-1, j)$ lies on a boundary with boundary value $g_3(x,y)$, equation (52) gives the approximation in the case that the boundary has a Dirichlet condition and equation (53) gives the approximation in the case of a Neumann condition of the form $\frac{\partial g}{\partial y}(x,y) = g_3(x,y)$.

$$\frac{\partial g}{\partial y}(x_l + (j+1)h, y_l + (i+1)h) \approx \frac{g_{i+1,j} - g_3(x_l + (j+1)h, y_l + ih)}{2h} \tag{52}$$

$$\frac{\partial g}{\partial y}(x_l + (j+1)h, y_l + (i+1)h) \approx \frac{1}{2}\left(\frac{g_{i+2,j} - g_{i,j}}{2h} + g_3(x_l + (j+1)h, y_l + ih)\right) \tag{53}$$

If $(i+1, j)$ lies on a boundary with boundary value $g_4(x,y)$, equation (54) gives the approximation in the case that the boundary has a Dirichlet condition and equation (55) gives the approximation in the case of a Neumann condition of the form $\frac{\partial g}{\partial y}(x,y) = g_4(x,y)$.

$$\frac{\partial g}{\partial y}(x_l + (j+1)h, y_l + (i+1)h) \approx \frac{g_4(x_l + (j+1)h, y_l + (i+2)h) - g_{i-1,j}}{2h} \tag{54}$$

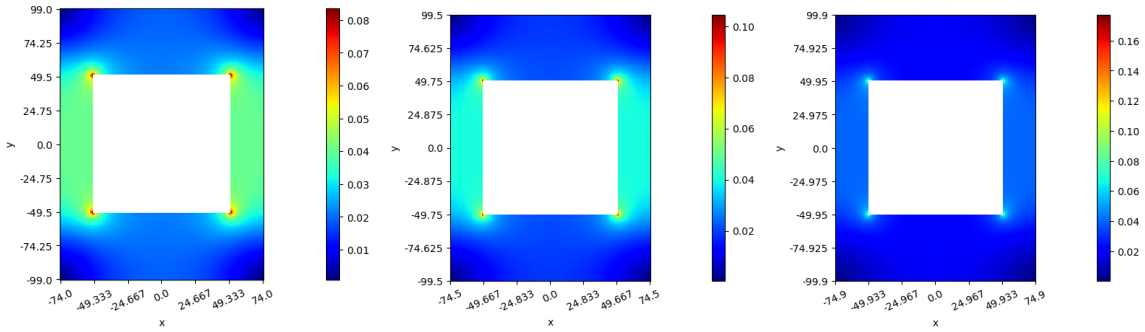
$$\frac{\partial g}{\partial y}(x_l + (j+1)h, y_l + (i+1)h) \approx \frac{1}{2}\left(\frac{g_{i,j} - g_{i-2,j}}{2h} + g_4(x_l + (j+1)h, y_l + (i+2)h)\right) \tag{55}$$

By applying equations (46) through (55) appropriately, we can approximate $\frac{\partial g}{\partial x}$ and $\frac{\partial g}{\partial y}$ at all of the interior grid points that are not in inclusions.

We first implement this strategy in Python using a pair of nested loops. That is, we visit each interior grid point (i,j) and apply the appropriate equations from (46) through (55). However, this strategy is inefficient assuming that the majority of interior grid points do not have neighbors who lie on a boundary. Thus, we consider a different approach. We store $\{g_{i,j}\}$ in a `numpy.ndarray`. For any (i,j) that is in an inclusion, the element in the array is set to `numpy.nan`. We then apply the following strategy.

1. Approximate $\frac{\partial g}{\partial x}$ at all interior grid points besides those in column $j = 0$ or $j = N_c - 1$ using (46).
2. Approximate $\frac{\partial g}{\partial y}$ at all interior grid points besides those in row $i = 0$ or $i = N_r - 1$ using (47).
3. Go back and revisit all interior grid points who have neighbors on a boundary and approximate $\frac{\partial g}{\partial x}$ and $\frac{\partial g}{\partial y}$ appropriately at these points using (48) through (55).

We know ahead of time that equation (46) cannot be used to approximate $\frac{\partial g}{\partial x}$ on the first and last interior grid columns because we know those columns have horizontal neighbors that lie on the boundary of the domain.



(a) $h = 1$ ($N_r = 199, N_c = 149$) (b) $h = 0.5$ ($N_r = 399, N_c = 299$) (c) $h = 0.1$ ($N_r = 1999, N_c = 1499$)

Figure 6: Approximations for $\left\|\nabla p(x,y)\right\|_2$ for Single Inclusion Benchmark

For a similar reason we know ahead of time that equation (47) cannot be used to approximate $\frac{\partial g}{\partial y}$ on the first and last interior grid rows. This is why these interior grid regions are omitted in steps 1 and 2. Steps 1 and 2 can be implemented quite efficiently on a `numpy.ndarray` by using slicing. Step 3 is necessary as the slicing in steps 1 and 2 will result in (incorrect) gradient approximations of `numpy.nan` for interior points who have neighbors on inclusion boundaries. Assuming that the majority of interior grid points do not have neighbors on an inclusion boundary, we do not approximate the derivative twice for too many points and take advantage of the efficiency of slicing over nested loops on the majority of points.

Now, we return to the example problem described on page 16. Once we get $p_{i,j}$ from our solver, using the boundary conditions provided in the problem, we can approximate $\left\|\nabla p(x,y)\right\|_2$ using the strategy above. Figure 6 shows our approximations of $\left\|\nabla p(x,y)\right\|_2$ for grids constructed for $h = 1, 0.5$, and 0.1. (N_r, N_c) for each grid are included in the figure captions. These match the plots in Figure 1 of Giacomini and Sevilla's work [4].

3 Fluid Velocity Solver

In this section, we describe our fluid velocity solver which approximates the solution to the problem outlined in section 1.1. In particular, we calculate what $\mathbf{u}(x,y,t)$ converges to over time. We discuss how we implement our fluid velocity solver, how we test it, and some unresolved issues with it.

3.1 Iterative Method

Our method for approximating $u(x,y,t)$ and $v(x,y,t)$ is based off of the explicit Forward Euler strategy and follows the work of Johnston and Liu [7]. As in section 2.1, let us first consider a rectangular domain D without inclusions. Our update step is given below starting with $u(x,y,0) = v(x,y,0) = 0$ for all $(x,y) \in D$:

1. Find the solution $p(x, y, t_k)$ to the following problem:

$$\Delta p = 2\rho \left(\frac{\partial u}{\partial x}(x, y, t_k) \frac{\partial v}{\partial y}(x, y, t_k) - \frac{\partial u}{\partial y}(x, y, t_k) \frac{\partial v}{\partial x}(x, y, t_k) \right) \tag{56}$$

$$(x, y) \in D = [x_l, x_h] \times [y_l, y_h] \tag{57}$$

$$\frac{\partial p}{\partial y}(x, y_l, t_k) = \mu \frac{\partial^2 v}{\partial y^2}(x, y_l, t_k) \tag{58}$$

$$\frac{\partial p}{\partial y}(x, y_h, t_k) = \mu \frac{\partial^2 v}{\partial y^2}(x, y_h, t_k) \tag{59}$$

$$p(x_l, y, t_k) = p_l \qquad p(x_h, y, t_k) = p_r \tag{60}$$

2. Calculate $u(x, y, t_{k+1})$ and $v(x, y, t_{k+1})$ as follows:

$$\begin{aligned} u(x, y, t_{k+1}) = u(x, y, t_k) - \frac{\Delta t}{\rho} \frac{\partial p}{\partial x}(x, y, t_k) - \Delta t \left(u(x, y, t_k) \frac{\partial u}{\partial x}(x, y, t_k) + v(x, y, t_k) \frac{\partial u}{\partial y}(x, y, t_k) \right) \\ + \Delta u(x, y, t_k) \frac{\mu \Delta t}{\rho} \end{aligned} \tag{61}$$

$$\begin{aligned} v(x, y, t_{k+1}) = v(x, y, t_k) - \frac{\Delta t}{\rho} \frac{\partial p}{\partial y}(x, y, t_k) - \Delta t \left(u(x, y, t_k) \frac{\partial v}{\partial x}(x, y, t_k) + v(x, y, t_k) \frac{\partial v}{\partial y}(x, y, t_k) \right) \\ + \Delta v(x, y, t_k) \frac{\mu \Delta t}{\rho} \end{aligned} \tag{62}$$

In order to implement this strategy, we have to make a variety of approximations. D is discretized in the same manner as given in equation (11) from section 2.1 given some spacing h. We approximate $u(x, y, t_k)$ and $v(x, y, t_k)$ for a particular time step t_k at the values (x, y) that lie on the interior of this grid. That is, let $u_{i,j} \approx u(x_l + (j+1)h, y_l + (i+1)h, t_k)$ and $v_{i,j} \approx v(x_l + (j+1)h, y_l + (i+1)h, t_k)$.

As derived in equations (46) through (55) in section 2.4, we know how to approximate $\frac{\partial g}{\partial x}$ and $\frac{\partial g}{\partial y}$ over interior grid points $g_{i,j}$ given the boundary conditions on those points. By applying this strategy with $g_{i,j} = v_{i,j}$ for interior grid points (i, j) and boundary conditions $v(x, y_l) = v(x, y_h) = v(x_l, y) = v(x_h, y) = 0$, we can get approximations for $\frac{\partial v}{\partial x}(x, y, t_k)$ and $\frac{\partial v}{\partial y}(x, y, t_k)$. Before applying this strategy to approximate $\frac{\partial u}{\partial x}(x, y, t_k)$ and $\frac{\partial u}{\partial y}(x, y, t_k)$ we show that the Neumann conditions on $u(x, y, t_k)$ expressed in equations (3) and (4) in section 1.1 can be expressed as Dirichlet conditions.

Since the fluid being modelled is incompressible on D, we can derive the following using the finite centered difference method given in equation (12) from section 2.1:

$$\begin{aligned} \frac{\partial u}{\partial x}(x_l + h, y, t_k) + \frac{\partial v}{\partial y}(x_l + h, y, t_k) \approx \frac{u(x_l + 2h, y, t_k) - u(x_l, y, t_k)}{2h} \\ + \frac{v(x_l + h, y + h, t_k) - v(x_l + h, y - h, t_k)}{2h} = 0 \end{aligned}$$

$$\frac{\partial u}{\partial x}(x_l + N_c h, y, t_k) + \frac{\partial v}{\partial y}(x_l + N_c h, y, t_k) \approx \frac{u(x_l + (N_c+1)h, y, t_k) - u(x_l + (N_c-1)h, y, t_k)}{2h} + \frac{v(x_l + N_c h, y + h, t_k) - v(x_l + N_c h, y - h, t_k)}{2h} = 0$$

Therefore, we can more explicitly write our new boundary conditions on $u(x, y, t_k)$ as:

$$u(x_l, y_l + (i+1)h, t_k) = u(x_l + 2h, y_l + (i+1)h, t_k) + v(x_l + h, y_l + (i+2)h, t_k) - v(x_l + h, y_l + ih, t_k) \tag{63}$$

$$u(x_l + (N_c+1)h, y_l + (i+1)h, t_k) = u(x_l + (N_c-1)h, y_l + (i+1)h, t_k) - v(x_l + N_c h, y_l + (i+2)h, t_k) + v(x_l + N_c h, y_l + ih, t_k) \tag{64}$$

Since these boundary conditions are Dirichlet, when they are evaluated to approximate $\frac{\partial u}{\partial x}(x, y, t_k)$ over our discretized region, y will lie on a grid row. This can be seen in the right hand side of equations (48) and (50) in section 2.4 which give the approximation of the horizontal derivative at grid points who have neighbors on boundaries with Dirichlet conditions.

It is important to note that the boundary conditions given by (63) and (64) are dependent on interior values of $u(x, y, t_k)$ and $v(x, y, t_k)$ which are the values we will be approximating. We can remedy this as follows. We know from equations (48) and (50) that the grid row of $u(x_l, y_l+(i+1)h, t_k)$ and $u(x_l+(N_c+1)h, y_l+(i+1)h, t_k)$ will be identical to the grid row of the interior point at which we are trying to approximate the horizontal derivative. Hence, we know that $u(x_l+2h, y_l+(i+1)h, t_k)$ and $u(x_l+(N_c-1)h, y_l+(i+1)h, t_k)$ are interior values and can be replaced by their respective approximations $u_{i,1}$ and u_{i,N_c-2}. If $1 \leq i \leq N_r - 2$, we also know that $v(x_l+h, y_l+(i+2)h, t_k), v(x_l+h, y_l+ih, t_k), v(x_l+N_c h, y_l+(i+2)h, t_k)$ and $v(x_l+N_c h, y_l+ih, t_k)$ are interior values and can thus be respectively replaced by their approximations $v_{i+1,0}, v_{i-1,0}, v_{i+1,N_c-1}$ and v_{i-1,N_c-1}. In the case that $i = 0$ or $i = N_r-1$ we can substitute 0 as we know that $v(x_l+h, y_l+(N_r+1)h, t_k) = v(x_l + h, y_l, t_k) = v(x_l + N_c h, y_l + (N_r + 1)h, t_k) = v(x_l + N_c h, y_l, t_k) = 0$.

With our new formulation of the boundary conditions on $u(x, y, t_k)$ we can use our strategy from section 2.4 to approximate the gradients of $u(x, y, t_k)$ and $v(x, y, t_k)$ using our approximations $u_{i,j}$ and $v_{i,j}$. However, update steps (61) and (62) also require an approximation of $\Delta u(x, y, t_k)$ and $\Delta v(x, y, t_k)$. To make these approximations, we can use many of the ideas from section 2.1. Again, consider a set of points $g_{i,j}$ as in section 2.4. If $g_{i,j}$ has no neighbors that lie on the boundary of the domain or of an inclusion, then we can use the 5-point strategy from section 2.1 to approximate the Laplacian at this point:

$$\Delta g(x_l + (j+1)h, y_l + (i+1)h) \approx \frac{g_{i-1,j} + g_{i+1,j} - 4g_{i,j} + g_{i,j-1} + g_{i,j+1}}{h^2} \tag{65}$$

Suppose $(i-1,j)$ lies on a boundary with boundary value given by $g_1(x,y)$. If the boundary has a Dirichlet condition, then we replace $g_{i-1,j}$ in (65) with $g_1(x_l+(j+1)h, y_l+ih)$. If the boundary has a Neumann condition of the form $\frac{\partial g}{\partial y}(x,y) = g_1(x,y)$ then, applying the forward difference method given by equation (19) in section 2.1, we replace $\frac{g_{i-1,j}-g_{i,j}}{h}$ in (65) with $-g_1(x_l+(j+1)h, y_l+ih)$. Suppose $(i+1,j)$ lies on a boundary with boundary value given by $g_2(x,y)$. If the boundary has a Dirichlet condition, then we replace $g_{i+1,j}$ in (65) with $g_2(x_l+(j+1)h, y_l+(i+2)h)$. If the boundary has a Neumann condition of the form $\frac{\partial g}{\partial y}(x,y) = g_2(x,y)$ then, applying the backward difference method given by (18), we replace $\frac{g_{i+1,j}-g_{i,j}}{h}$ in (65) with $g_2(x_l+(j+1)h, y_l+(i+2)h)$. Suppose $(i,j-1)$ lies on a boundary with boundary value given by $g_3(x,y)$. If the boundary has a Dirichlet condition, then we replace $g_{i,j-1}$ in (65) with $g_3(x_l+jh, y_l+(i+1)h)$. If the boundary has a Neumann condition of the form $\frac{\partial g}{\partial x}(x,y) = g_3(x,y)$ then, applying the forward difference method given by (19), we replace $\frac{g_{i,j-1}-g_{i,j}}{h}$ in (65) with $-g_3(x_l+jh, y_l+(i+1)h$. Suppose $(i,j+1)$ lies on a boundary with boundary value given by $g_4(x,y)$. If the boundary has a Dirichlet condition, then we replace $g_{i,j+1}$ in (65) with $g_4(x_l+(j+2)h, y_l+(i+1)h)$. If the boundary has a Neumann condition of the form $\frac{\partial g}{\partial x}(x,y) = g_4(x,y)$ then, applying the backward difference method given by (18), we replace $\frac{g_{i,j+1}-g_{i,j}}{h}$ in (65) with $g_4(x_l+(j+2)h, y_l+(i+1)h)$.

Similar to our strategy described in section 2.4, if $\{g_{i,j}\}$ are stored in a `numpy.ndarray` we can use the following strategy to approximate $\Delta g(x,y)$.

1. Approximate $\Delta g(x,y)$ at all interior grid points besides those in column $j=0$ and $j=N_c-1$ and rows $i=0$ and $i=N_r-1$ with equation (65).

2. Go back and revisit all interior grid points who have neighbors on a boundary and approximate $\Delta g(x,y)$ appropriately using the strategy described in the previous paragraph.

Similarly to approximating the gradient, we implement step 1 efficiently using slicing. Step 2 goes back and corrects Laplacian approximations for interior points with neighbors on boundaries. Assuming that the majority of interior grid points do not have boundary neighbors, this strategy allows us to take advantage of slicing without computing the Laplacian multiple times at too many points.

The next step of implementing the update given on page 20 is solving the Poisson problem in step 1. As described previously, $\frac{\partial u}{\partial x}(x,y,t_k)$, $\frac{\partial v}{\partial y}(x,y,t_k)$, $\frac{\partial u}{\partial y}(x,y,t_k)$ and $\frac{\partial v}{\partial x}(x,y,t_k)$ are all approximated as discretized sets of points $g_{i,j}$. Hence, the right hand side of equation (56) is also a discretized set of points. As described in section 2.1 in the construction of $\mathbf{b}$, we must be able to access the right hand side of Poisson's equation for every point in the grid used to discretize the equation domain. Therefore, we must use the same grid spacing h in the discretization of $u(x,y,t_k)$ and $v(x,y,t_k)$ as in our Poisson solver.

We also must approximate the right hand side of the Neumann conditions on $p(x,y,t_k)$ given by equations

(58) and (59). As described by Johnston and Liu [7], we can approximate:

$$\frac{\partial^2 v}{\partial y^2}(x_l + (j+1)h, y_l) \approx \frac{2v_{0,j}}{h^2} \qquad \frac{\partial^2 v}{\partial y^2}(x_l + (j+1)h, y_h) \approx \frac{2v_{N_r-1,j}}{h^2}$$

Now that the Poisson problem given by (56) through (60) can be expressed explicitly in terms of our approximations $u_{i,j}$ and $v_{i,j}$, we can solve it using our Poisson solver to get our approximation $p_{i,j}$ for $p(x, y, t_k)$. The output of the solver can then be used along with our gradient approximation strategy from section 2.4 to approximate $\frac{\partial p}{\partial x}(x, y, t_k)$ and $\frac{\partial p}{\partial y}(x, y, t_k)$ in part 2 of the update step on page 20. We now present a more detailed version of our method for approximating the fluid velocity.

Initialization: Initialize $u_{i,j}$ and $v_{i,j}$ to be all 0. Let $\Delta t = \frac{1}{4h^2}\frac{\rho}{\mu}$ seconds. Our method will simulate a total time of $T = \frac{\rho(x_h - x_l)^2}{\mu}$ seconds.

Iteration (or **Time step**)**:** For $k = 1, \ldots, \frac{T}{\Delta t}$ do the following:

1. Approximate $\frac{\partial u}{\partial x}(x, y, t_k)$,$\frac{\partial u}{\partial y}(x, y, t_k)$,$\Delta u(x, y, t_k)$,$\frac{\partial v}{\partial x}(x, y, t_k)$,$\frac{\partial v}{\partial y}(x, y, t_k)$, and $\Delta v(x, y, t_k)$. Denote these $G_x u_{i,j}, G_y u_{i,j}, Lu_{i,j}, G_x v_{i,j}, G_y v_{i,j}$ and $Lv_{i,j}$. Note that we can combine steps 2 and 3 from our gradient approximation strategy with step 2 from our Laplacian approximation strategy into the same loop to speed up the computation of the gradient and Laplacian for a set of points $g_{i,j}$.

2. Solve the pressure Poisson problem below to get our pressure approximation $p_{i,j}$.

$$\begin{aligned}
\Delta p(x, y) &= 2\rho(G_x u_{i,j} G_y v_{i,j} + G_y u_{i,j} G_x v_{i,j}) \\
\frac{\partial p}{\partial y}(x, y_l) &= \frac{2\mu v_{0,j}}{h^2} \qquad \frac{\partial p}{\partial y}(x, y_h) = \frac{2\mu v_{N_r-1,j}}{h^2} \\
p(x_l, y) &= p_l \qquad p(x_h, y) = p_r
\end{aligned}$$

3. Approximate $\frac{\partial p}{\partial x}(x, y, t_k)$ and $\frac{\partial p}{\partial y}(x, y, t_k)$ using $p_{i,j}$. Denote the approximations as $G_x p_{i,j}$ and $G_y p_{i,j}$.

4. Update $u_{i,j}$ and $v_{i,j}$ as follows:

$$\begin{aligned}
u'_{i,j} &= u_{i,j} - \frac{\Delta t}{\rho} G_x p_{i,j} - \Delta t(u_{i,j} G_x u_{i,j} + v_{i,j} G_y u_{i,j}) + \frac{\mu \Delta t}{\rho} Lu_{i,j} \\
v'_{i,j} &= v_{i,j} - \frac{\Delta t}{\rho} G_y p_{i,j} - \Delta t(u_{i,j} G_x v_{i,j} + v_{i,j} G_y v_{i,j}) + \frac{\mu \Delta t}{\rho} Lv_{i,j} \\
u_{i,j} &= u'_{i,j} \qquad v_{i,j} = v'_{i,j}
\end{aligned}$$

Output: $p_{i,j}, u_{i,j}$, and $v_{i,j}$ from the final time step. We refer to this as the output of our velocity solver.

Observe that in the derivation of the Poisson solver in section 2.1 that the matrix A is dependent only on the type of boundary conditions and location of inclusions on the domain D. A does not depend on the boundary values and the right hand side of Poisson's equation in step 2 above. Since only these aspects of the pressure Poisson equation are changed at each iteration, A can be constructed and decomposed once at the beginning of our algorithm. By performing an LU decomposition of A, solving $A\mathbf{p} = \mathbf{b}$ for different $\mathbf{b}$ reduces to just performing 1 forward substitution, 1 backward substitution, and 1 matrix vector multiplication as shown in section 2.1. This is more efficient than performing a full solve of $A\mathbf{p} = \mathbf{b}$ for every iteration.

3.2 Results for Plane Poiseuille Flow

We implement the strategy described in section 3.1 in a Python class `VelocitySolver`. We initialize a `VelocitySolver` with $x_l, x_h, y_l, y_h, p_l, p_r, \rho, \mu$ and h. For all the experiments we perform we took $p_l = 0.08$ Ba, $p_r = 0$ Ba, $\rho = 1$ g/cm^3, and $\mu = 0.01$ P. We benchmark `VelocitySolver` by having it approximate $u(x, y, t)$ and $v(x, y, t)$ on $D = [0 \text{ cm}, 0.01 \text{ cm}] \times [0 \text{ cm}, 0.01 \text{ cm}]$ without any inclusions. As shown in G. K. Batchelor's "Introduction to Fluid Dynamics", $u(x, y, t)$ and $v(x, y, t)$ can be found analytically [2]:

$$u(x, y, t) = \frac{p_l - p_r}{x_h - x_l} \frac{1}{2\mu} y((y_h - y_l) - y) \qquad v(x, y, t) = 0 \tag{66}$$

Figure 7 shows the maximum and mean absolute error between our solver's approximation and the true solution for $u(x, y, t)$ given by (66) for each time step over an interior grid with $N_r = N_c = 79$. Here we can see that our solver converges to $u(x, y, t)$ for later time steps. Our solver accurately approximates $v(x, y, t)$ as 0 from very early on.

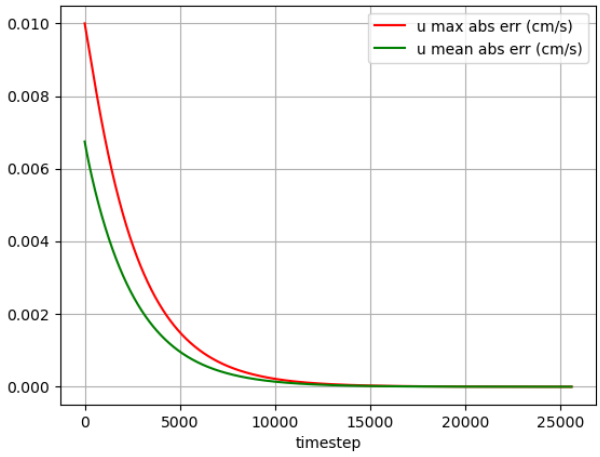


Figure 7: Convergence of $u(x, y, t)$ Approximation Error for Plane Poiseuille Flow

3.3 Addition of Inclusions

The next step was to approximate $u(x, y, t)$ and $v(x, y, t)$ when we add rectangular inclusions into the problem domain. As in section 2.3, we require that inclusions cannot be too close to each other or the domain boundary and they must have bounds that lie on the grid discretization of D. Suppose there is an inclusion that spans $[x_l + (j_l + 1)h, x_l + (j_h + 1)h] \times [y_l + (i_l + 1)h, y_l + (i_h + 1)h]$. We impose the following boundary conditions on $u(x, y, t)$ and $v(x, y, t)$:

$$u(x_l + (j_l + 1)h, y, t) = v(x_l + (j_l + 1)h, y, t) = 0 \quad \text{for } y \in \{y_l + (i + 1)h : i = i_l, i_l + 1, \ldots, i_h\} \tag{67}$$

$$u(x_l + (j_h + 1)h, y, t) = v(x_l + (j_h + 1)h, y, t) = 0 \quad \text{for } y \in \{y_l + (i + 1)h : i = i_l, i_l + 1, \ldots, i_h\} \tag{68}$$

$$u(x, y_l + (i_l + 1)h, t) = v(x, y_l + (i_l + 1)h, t) = 0 \quad \text{for } x \in \{x_l + (j + 1)h : j = j_l, j_l + 1, \ldots, j_h\} \tag{69}$$

$$u(x, y_l + (i_h + 1)h, t) = v(x, y_l + (i_h + 1)h, t) = 0 \quad \text{for } x \in \{x_l + (j + 1)h : j = j_l, j_l + 1, \ldots, j_h\} \tag{70}$$

We modify the initialization step from our strategy at the end of section 3.1 to set $u_{i,j} = v_{i,j}$ to `numpy.nan` for (i, j) that lie in inclusions. We keep step 1 of our iteration the same. Our strategies for approximating the gradient and Laplacian on $\{g_{i,j}\}$ work in the presence of inclusions as long as the boundary conditions on those inclusions given by (67) through (70) are incorporated. We modify the Poisson problem in step 2 as follows. Taking an inclusion spanning $[x_l + (j_l + 1)h, x_l + (j_h + 1)h] \times [y_l + (i_l + 1)h, y_l + (i_h + 1)h]$ as above, we impose the following boundary conditions on $p(x, y, t)$ [7]:

$$\begin{aligned}
\frac{\partial p}{\partial x}(x_l + (j_l + 1)h, y, t) &\approx \frac{2\mu u_{i,j_l-1}}{h^2} \quad \text{for } y \in \{y_l + (i + 1)h : i = i_l, i_l + 1, \ldots, i_h\} \\
\frac{\partial p}{\partial x}(x_l + (j_h + 1)h, y, t) &\approx \frac{2\mu u_{i,j_h+1}}{h^2} \quad \text{for } y \in \{y_l + (i + 1)h : i = i_l, i_l + 1, \ldots, i_h\} \\
\frac{\partial p}{\partial y}(x, y_l + (i_l + 1)h, t) &\approx \frac{2\mu v_{i_l-1,j}}{h^2} \quad \text{for } x \in \{x_l + (j + 1)h : j = j_l, j_l + 1, \ldots, j_h\} \\
\frac{\partial p}{\partial y}(x, y_l + (i_h + 1)h, t) &\approx \frac{2\mu v_{i_h+1,j}}{h^2} \quad \text{for } x \in \{x_l + (j + 1)h : j = j_l, j_l + 1, \ldots, j_h\}
\end{aligned}$$

These boundary conditions can be expressed in terms of our approximations $u_{i,j}$ and $v_{i,j}$ which means we can continue to use our Poisson solver as before to complete step 2. We incorporate these boundary conditions into the gradient computations in step 3 of our iteration. We keep step 4 the same.

3.4 Issues with the Solver

As discussed in 3.2, our solver appears to work well in the case of a domain with no inclusions. However, we find that our solver is not perfect at approximating the fluid velocity even with only 1 inclusion. We test our solver by having it approximate $u(x, y, t)$ and $v(x, y, t)$ on $D = [0 \text{ cm}, 0.01 \text{ cm}] \times [0 \text{ cm}, 0.01 \text{ cm}]$ with an

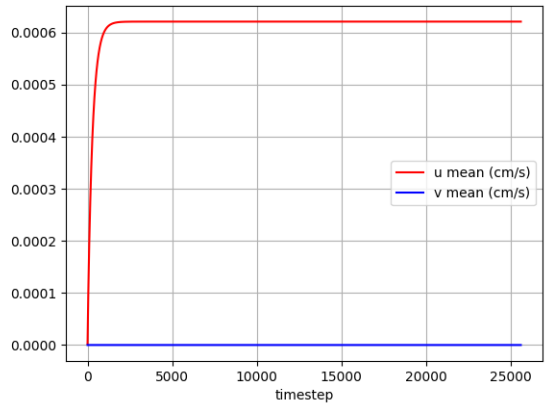


Figure 8: Convergence of the Mean of $\{u_{i,j}\}$ for Single Inclusion

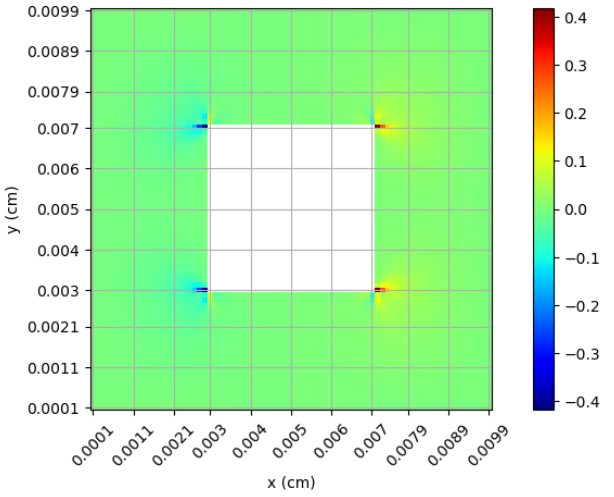


Figure 9: Velocity Divergence after 25,601 Time Steps for Single Inclusion

inclusion spanning $[0.003 \text{ cm}, 0.007 \text{ cm}] \times [0.003 \text{ cm}, 0.007 \text{ cm}]$. With a step size of $h = 0.000125$ cm, the interior of D is discretized such that $N_r = N_c = 79$ and our solver runs for 25,601 time steps. Figure 8 shows the mean of our approximations $u_{i,j}$ and $v_{i,j}$ for each time step. Here we can see that, over time, our approximations converge, which is a good sign. Figure 9 shows the divergence of our solver's approximation of the velocity after the final time step. Since the fluid of study is incompressible, the divergence of our solver's approximation of the velocities should be 0 everywhere on D. However, we can see that the divergence is nonzero at some points near the corners of the inclusion.

Figure 10 shows the flow rate across the horizontal axis of the domain calculated from $u_{i,j}$ after the final time step. The flow rate at column j is calculated as h multiplied with the sum over $u_{i,j}$ for $i = 0, 1, \ldots, N_r - 1$ and (i, j) not in an inclusion. In the figure, we can see that the flow rate is not constant as it should be. We spent a great deal of time looking into the potential source of these errors and even went as far as reimplementing our fluid velocity solver from scratch. However, we are unable to identify why the velocity for points around the inclusion corners had nonzero divergence. Furthermore, we find that increasing the grid size did not remove the error evident in Figure 9 but it did decrease the flow rate error. Let f_j denote the flow rate at column j and $\bar{f}$ as the mean flow rate over the columns of the interior of our domain grid. We define our error metric for flow rate as:

$$\max_{j=0,1,\ldots,N_c-1} \frac{\left|f_j - \bar{f}\right|}{f_j}$$

Figure 11 shows a plot of this error metric against the step size. For a step size of $h = 0.00025$ cm, our solver

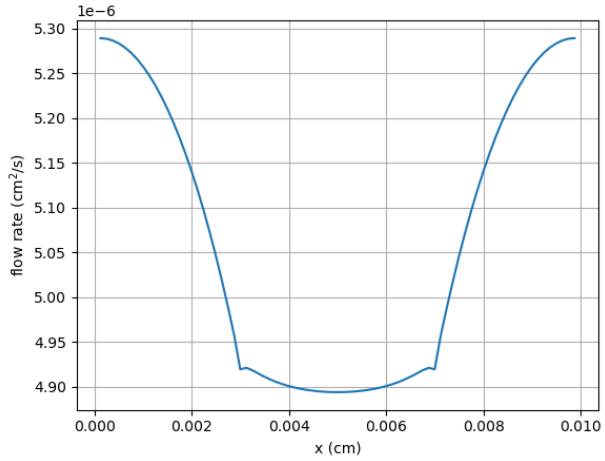


Figure 10: Flow Rate Over the Domain for Single Inclusion

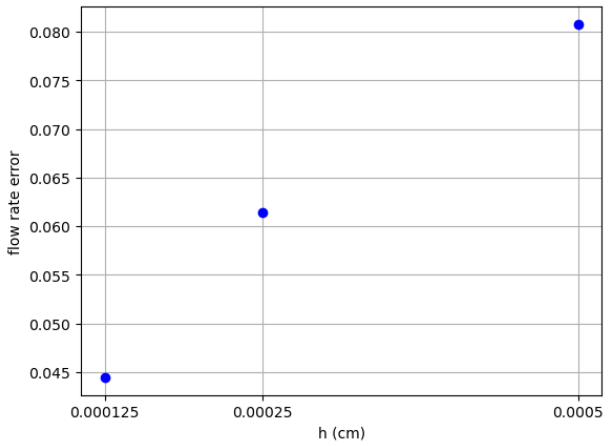


Figure 11: Flow Rate Error for Varying Discretization for Single Inclusion

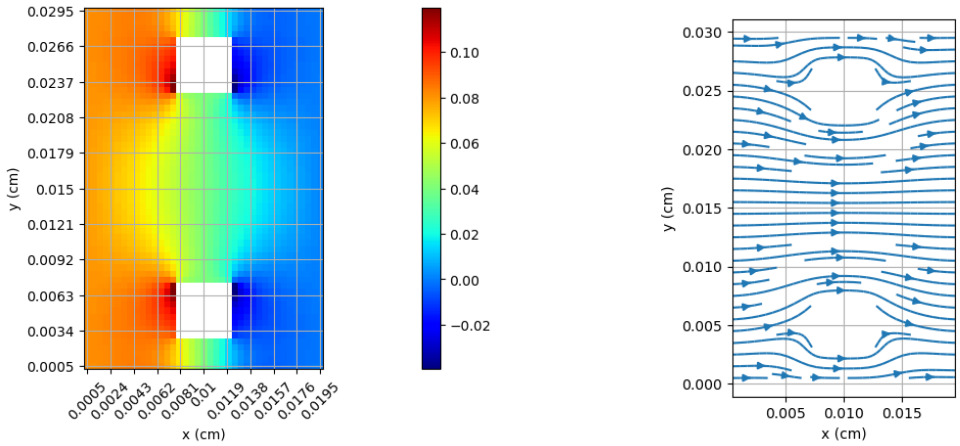
discretizes the interior of D with a grid such that $N_r = N_c = 39$ and runs for 6,401 time steps. For a step size of $h = 0.0005$ cm, $N_r = N_c = 19$ and the solver runs for 1,601 time steps. Here we can see that as the step size is decreased, the flow rate error decreases as well. In general, we wanted this error metric to be less than 0.10. We discuss in the next section that this cannot always be guaranteed.

4 Analysis of Inclusion Arrangements

As discussed in section 1, placing inclusions in the domain can be used to manipulate the flow of the fluid passed through the domain. In this section, we describe 4 numerical experiments where we study the effects of inclusion arrangement on fluid flow. In particular, we examine the effects on hydraulic resistance. In all 4 of the experiments, we keep $D = [0 \text{ cm}, 0.02 \text{ cm}] \times [0 \text{ cm}, 0.03 \text{ cm}]$, $p_l = 0.08$ Ba, $p_r = 0$ Ba, $\mu = 0.01$ P, and $\rho = 1$ g/cm^3. We calculate the hydraulic resistance from our solver's output as $(p_l - p_r)/\bar{f}$ for $\bar{f}$ defined on the previous page. We choose the mean flow rate due to the variance in flow rate observed with our solver discussed in the previous section. We end this section with a comparison of the hydraulic resistance calculated from our solver's output with an analytic approximation.

4.1 Experiment 1: Closing Inclusions

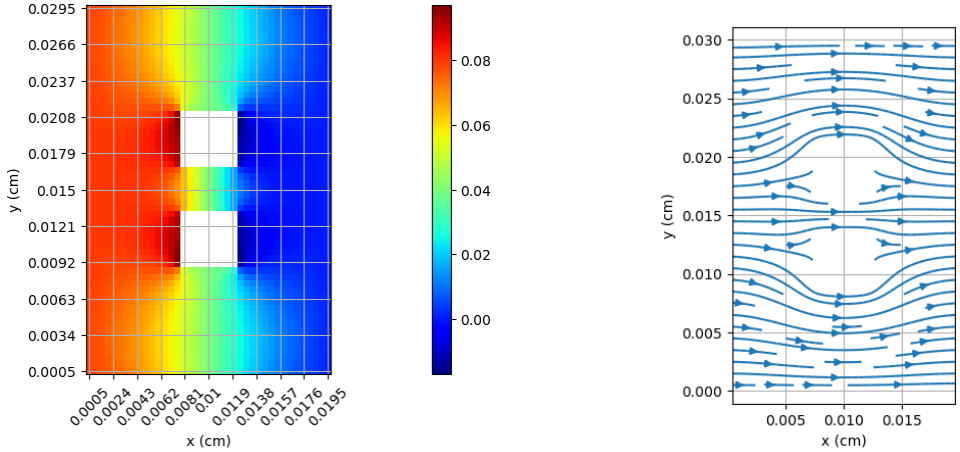
For our first experiment, we want to see how the vertical proximity of inclusions affect fluid flow. To do this, we place 2 inclusions symmetrically distant from the top and bottom boundaries of D and move them closer together over 7 runs. We calculate the hydraulic resistance at the end of each run from the output of our



(a) Interior Pressure (Ba)

(b) Streamlines

Figure 12: Interior Pressure and Streamline Plots Made from the Output of our Velocity Solver for Far Inclusions



(a) Interior Pressure (Ba)

(b) Streamlines

Figure 13: Interior Pressure and Streamline Plots Made from the Output of our Velocity Solver for Close Inclusions

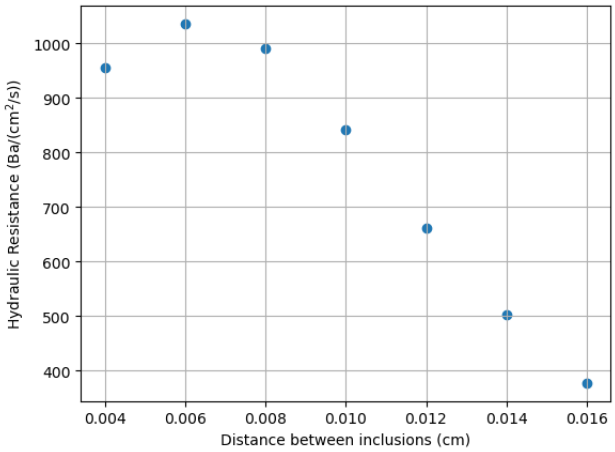


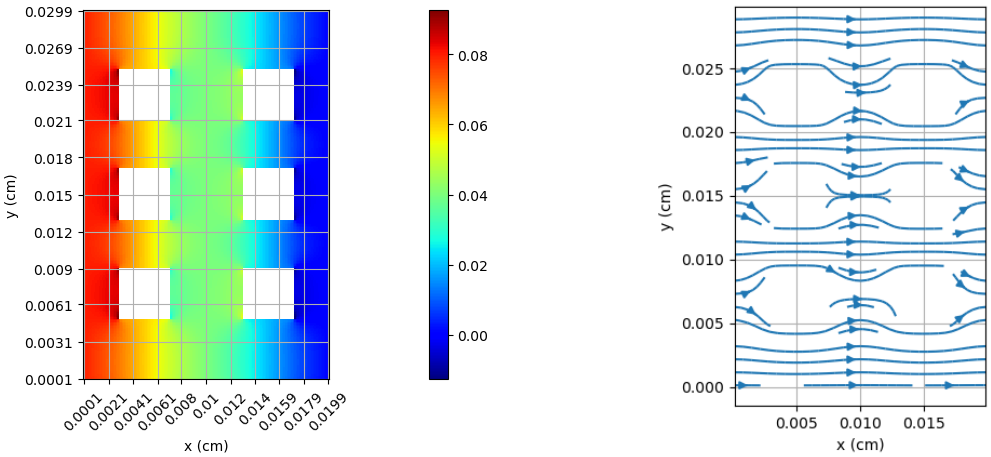
Figure 14: Hydraulic Resistance for Closing Inclusions

solver as described above. For these runs, D was discretized with $h = 0.0005$ cm which meant $N_r = 59$ and $N_c = 39$ and that the solver ran for 6,401 time steps. Figure 12 (a) shows a plot of the interior pressure approximated by our solver at the last time step when the inclusions are farthest apart. That top inclusion spans $[0.008 \text{ cm}, 0.012 \text{ cm}] \times [0.023 \text{ cm}, 0.027 \text{ cm}]$ while the bottom one spans $[0.008 \text{ cm}, 0.0012 \text{ cm}] \times [0.003 \text{ cm}, 0.007 \text{ cm}]$. Figure 12 (b) is a streamline plot constructed using the velocities calculated in the final time step on the same domain. Figure 13 has the pressure and streamline plots derived from the final time step when the inclusions are closest together. When they are closest, the top inclusion spans $[0.008 \text{ cm}, 0.012 \text{ cm}] \times [0.017 \text{ cm}, 0.021 \text{ cm}]$ while the bottom one spans $[0.008 \text{ cm}, 0.0012 \text{ cm}] \times [0.009 \text{ cm}, 0.013 \text{ cm}]$.

Starting from the set up displayed in Figure 12, we move up the bottom inclusion by 0.001 cm and the top inclusion down by 0.001 cm 6 times ending with the set up displayed in Figure 13. By doing this, we maintain for all of the 7 set ups that the top and bottom inclusions are symmetrically distant from the top and bottom domain boundaries. Figure 14 shows a plot of the distance between the top of the bottom inclusion and the bottom of the top inclusion versus the hydraulic resistance for our 7 runs in the experiment. There is a slight increase in hydraulic resistance between the first and second run, but then the hydraulic resistance decreases while the distance between inclusions increases. It is also important to note that the flow rate error, as defined in the previous section, remains between 0.038 and 0.083 for all 7 set ups. Since these errors are less than our threshold of 0.10, we are confident in these results.

4.2 Experiment 2: Staggering 6 Inclusions in 2 Columns

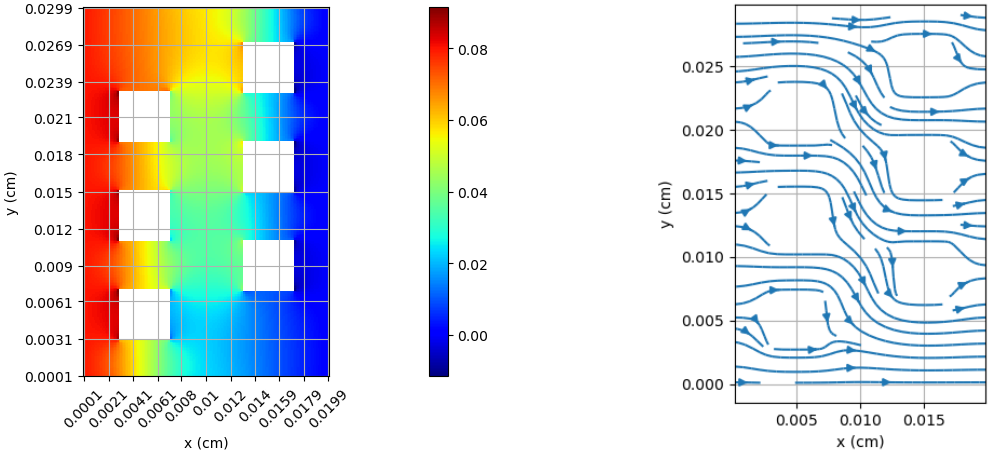
Suppose that we have 2 columns of with equal numbers of inclusions that the fluid must flow through when passing through the domain. Our second experiment looks to discover how the hydraulic resistance is affected by how much the inclusions in the 2 columns are horizontally aligned. In our first run, we place 6 inclusions



(a) Interior Pressure (Ba)

(b) Streamlines

Figure 15: Interior Pressure and Streamline Plots Made from the Output of our Velocity Solver for 6 Aligned Inclusions



(a) Interior Pressure (Ba)

(b) Streamlines

Figure 16: Interior Pressure and Streamline Plots Made from the Output of our Velocity Solver for 6 Staggered Inclusions

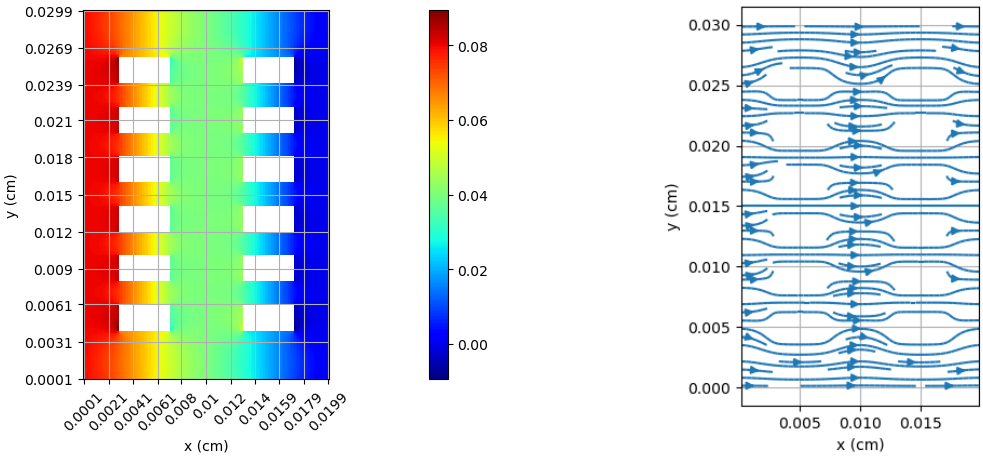
into 2 columns where the 3 inclusions in the second column completely line up horizontally with the 3 inclusions in the first column. In subsequent runs, we progressively shift the inclusions until the inclusions in the 2 columns line up the least. For these runs, we discretize D with $h = 0.000125$ cm which means $N_r = 239$ and $N_c = 159$ and that the solver runs for 102,401 time steps.

Figure 15 shows plots of the interior pressure and streamlines approximated by our solver at the last time step when the inclusions in the 2 columns are lined up completely. The first column has inclusions spanning $[0.003 \text{ cm}, 0.007 \text{ cm}] \times [0.005 \text{ cm}, 0.009 \text{ cm}]$, $[0.003 \text{ cm}, 0.007 \text{ cm}] \times [0.013 \text{ cm}, 0.017 \text{ cm}]$, and $[0.003 \text{ cm}, 0.007 \text{ cm}] \times [0.021 \text{ cm}, 0.025 \text{ cm}]$. The second column has inclusions spanning $[0.013 \text{ cm}, 0.017 \text{ cm}] \times [0.005 \text{ cm}, 0.009 \text{ cm}]$, $[0.013 \text{ cm}, 0.017 \text{ cm}] \times [0.013 \text{ cm}, 0.017 \text{ cm}]$, and $[0.013 \text{ cm}, 0.017 \text{ cm}] \times [0.021 \text{ cm}, 0.025 \text{ cm}]$. Figure 16 shows plots of the interior pressure and streamlines approximated by our solver at the last time step when the inclusions in the 2 columns are fully staggered. By this we mean that the inclusions in the 2 columns are lined up horizontally the least. The first column has inclusions spanning $[0.003 \text{ cm}, 0.007 \text{ cm}] \times [0.003 \text{ cm}, 0.007 \text{ cm}]$, $[0.003 \text{ cm}, 0.007 \text{ cm}] \times [0.011 \text{ cm}, 0.015 \text{ cm}]$, and $[0.003 \text{ cm}, 0.007 \text{ cm}] \times [0.019 \text{ cm}, 0.023 \text{ cm}]$. The second column has inclusions spanning $[0.013 \text{ cm}, 0.017 \text{ cm}] \times [0.007 \text{ cm}, 0.011 \text{ cm}]$, $[0.013 \text{ cm}, 0.017 \text{ cm}] \times [0.015 \text{ cm}, 0.019 \text{ cm}]$, and $[0.013 \text{ cm}, 0.017 \text{ cm}] \times [0.023 \text{ cm}, 0.027 \text{ cm}]$.

Starting from the set up displayed in Figure 15, we move the inclusions in the left column down by 0.0005 cm and the inclusions in the right column up by 0.0005 cm 4 times to end up with the set up displayed in Figure 16. At each of these steps, the inclusions in the right column line up 0.001 cm less with the inclusions in the left column. Let d_1 and d_3 be the distances in centimeters from the bottom boundary to the bottom of the bottom inclusions in the left and right columns respectively. Let d_2 and d_4 be the distances in centimeters from the top boundary to the top of the top inclusions in the left and right columns respectively. By staggering inclusions progressively in this fashion, we keep $d_1 = d_4$ and $d_2 = d_3$ at each staggering step. Our metric for the level of staggering is the difference in centimeters between the bottom of the bottom inclusion in the right column and the bottom of the bottom inclusion in the left column. This starts at 0 cm, then is 0.001 cm, and so on up to 0.004 cm. Figure 19 shows a plot of this distance versus the hydraulic resistance for the 5 runs. Here we can see that as inclusions become less aligned, the hydraulic resistance increases. We are confident in these results as the flow rate error remained between 0.041 and 0.047 for these 5 runs.

4.3 Experiment 3: Staggering 12 Inclusions in 2 Columns

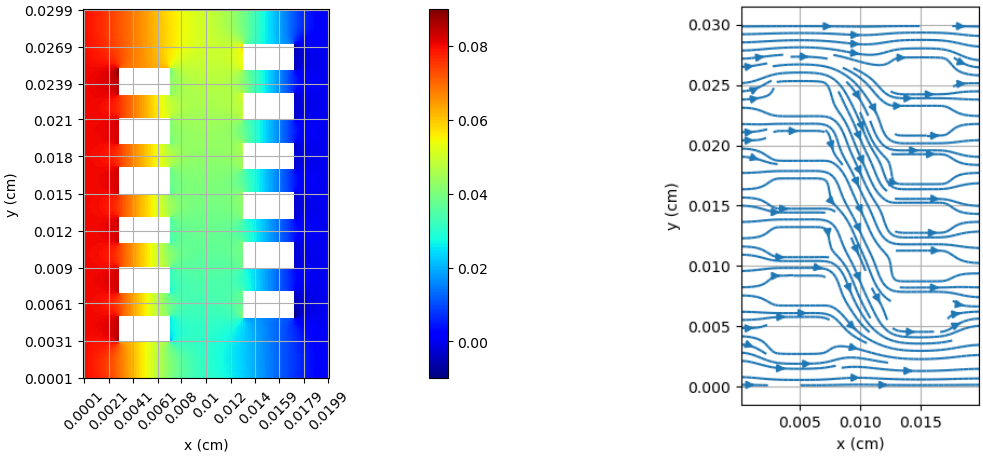
Our third experiment builds upon the second. Instead of 2 columns with 3 inclusions each, we had 2 columns with 6 inclusions each. We want to see if the hydraulic resistance follows the same trend as the inclusions



(a) Interior Pressure (Ba)

(b) Streamlines

Figure 17: Interior Pressure and Streamline Plots Made from the Output of our Velocity Solver for 12 Aligned Inclusions



(a) Interior Pressure (Ba)

(b) Streamlines

Figure 18: Interior Pressure and Streamline Plots Made from the Output of our Velocity Solver for 12 Staggered Inclusions

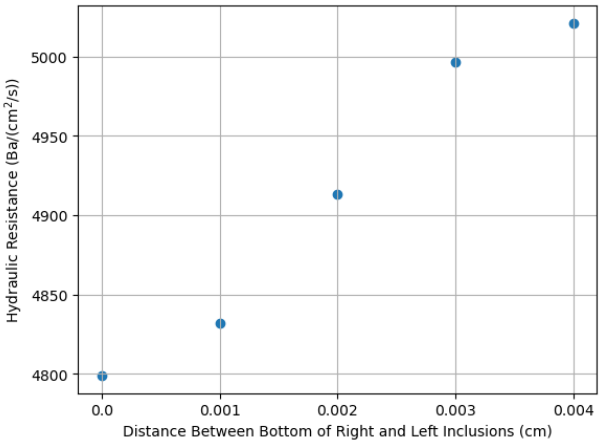


Figure 19: Hydraulic Resistance for 6 Staggering Inclusions

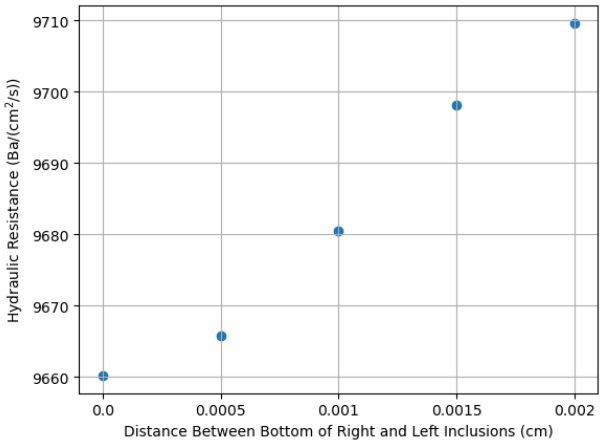


Figure 20: Hydraulic Resistance for 12 Staggering Inclusions

become less horizontally aligned. As in the second experiment, we discretize D with $h = 0.000125$ cm. Figure 17 shows plots of the interior pressure and streamlines of our first run, when the inclusions are lined up completely. As in the second experiment, the left column of inclusions spans x range 0.003 cm to 0.007 cm and the right column of inclusions spans x range 0.013 cm to 0.017 cm. The inclusions in both columns are placed at on y intervals [0.004 cm, 0.006 cm], [0.008 cm, 0.01 cm], [0.012 cm, 0.014 cm], [0.016 cm, 0.018 cm], [0.02 cm, 0.022 cm] and [0.024 cm, 0.026 cm].

Figure 18 shows plots of the interior pressure and streamlines approximated by our solver at the last time step when the inclusions in the 2 columns are staggered. The 2 inclusion columns lie on the same x ranges but on different y ranges. The first column has inclusions at y intervals [0.003 cm, 0.005 cm], [0.007 cm, 0.009 cm], [0.011 cm, 0.013 cm], [0.015 cm, 0.017 cm], [0.019 cm, 0.021 cm] and [0.023 cm, 0.025 cm]. The second column has inclusions at y intervals [0.005 cm, 0.007 cm], [0.009 cm, 0.011 cm], [0.013 cm, 0.015 cm], [0.017 cm, 0.019 cm], [0.021 cm, 0.023 cm] and [0.025 cm, 0.027 cm]. In these set ups, we have doubled the number of inclusions from the second experiment but halved their height. Since the inclusions' width is the same as in the second experiment, the area occupied by inclusions in the domain remains constant between the 2 experiments.

Starting from the set up in Figure 17, we move the inclusions in the first column down by 0.00025 cm and the inclusions in the second column up by 0.00025 cm 4 times to end with the set up in Figure 18. For each of these runs, the inclusions in the second column line up 0.0005 cm less with the inclusions in the first column. As in our second experiment, this keeps $d_1 = d_4$ and $d_2 = d_3$ for d_i defined previously. Figure 20 shows a plot of the distance between the bottom of the bottom inclusion in the right column and the bottom of the bottom inclusion in the left column versus the hydraulic resistance for the 5 runs. As in the case of the second experiment, we can see that as inclusions become less aligned, the hydraulic resistance increases.

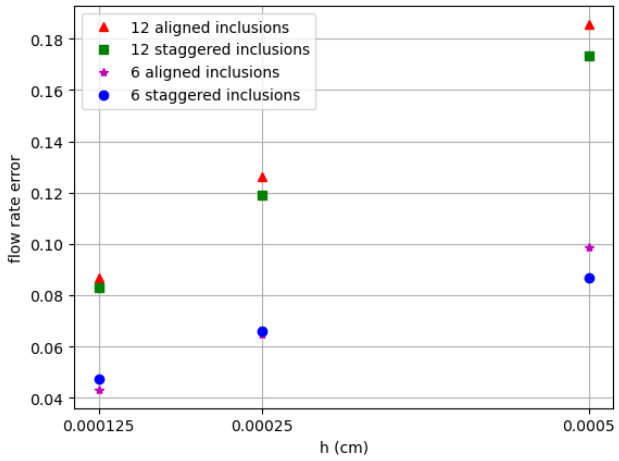


Figure 21: Flow Rate Error for Aligned and Staggered Setups (6 and 12 Inclusions)

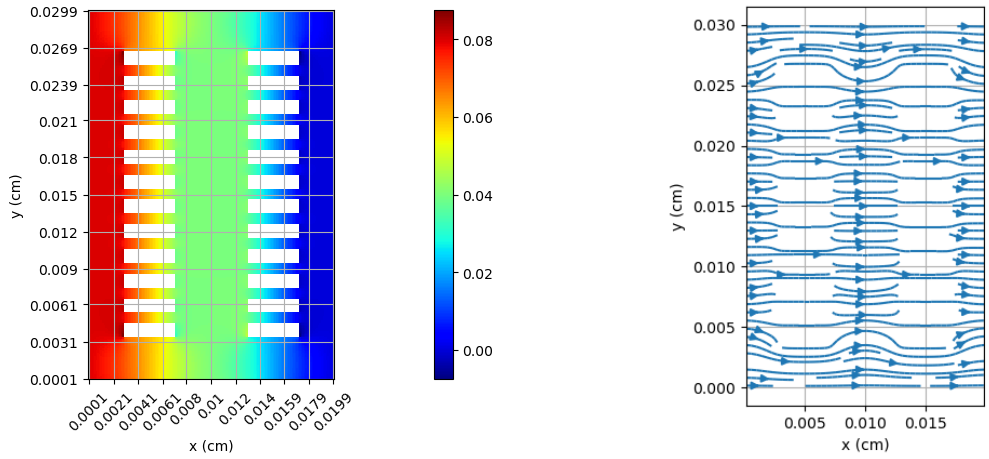
We are confident in these results because the flow rate error remained between 0.082 and 0.087 for these 5 runs.

In section 3.4, we show how the flow rate error decreased with step size. To see if this is also true for the set ups in our second and third experiments, we run our solver on the 6 and 12 inclusion aligned and fully staggered set ups with $h = 0.0005$ cm and $h = 0.00025$ cm. With $h = 0.0005$ cm, D is discretized such that $N_r = 59$ and $N_c = 39$ and the solver runs for 6,401 time steps. With $h = 0.00025$ cm, D is discretized such that $N_r = 119$ and $N_c = 79$ and the solver runs for 25,601 time steps. Figure 21 shows a plot of the flow rate error versus h for the aligned and fully staggered setups for 6 and 12 inclusions. We can see that the flow rate error is higher with more inclusions but the error decreases with h in all 4 set ups.

4.4 Experiment 4: Comparing Counts of Inclusions

In our fourth experiment, we study how the number of inclusions affects the hydraulic resistance in aligned and fully staggered set ups. To do this, we run our solver on 4 additional set ups. As before, we discretize D with $h = 0.000125$ cm. As in the second and third experiments, we place inclusions into 2 columns where the left column spans x ranges 0.003 cm to 0.007 cm and 0.013 cm to 0.017 cm. In our first set up, we split 24 inclusions into 2 columns such that the right inclusions line up horizontally with the left inclusions. Figure 22 shows plots of the interior pressure and streamlines for this set up. The first inclusion is placed on y interval $[0.0035 \text{ cm}, 0.0045 \text{ cm}]$ in both columns. We place inclusions with height 1 cm 1 cm apart up until the highest inclusions on y interval $[0.0255 \text{ cm}, 0.0265 \text{ cm}]$.

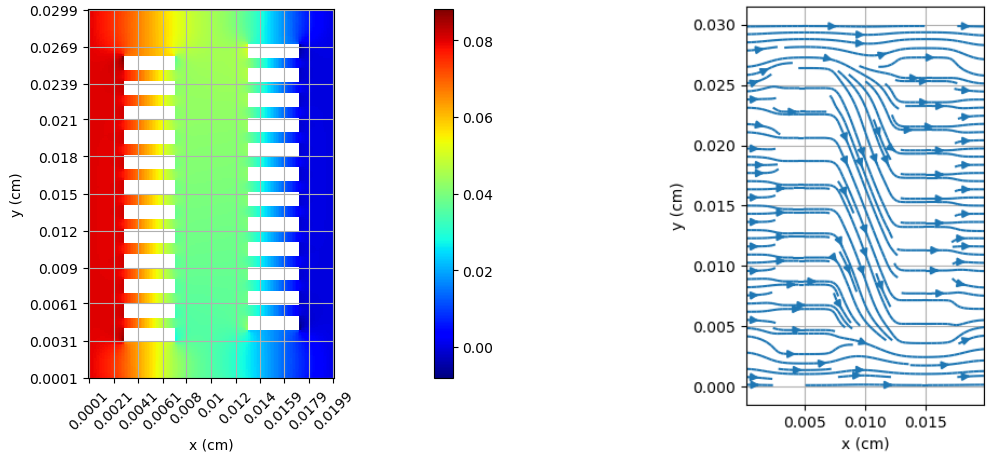
In our second set up, we shift the 12 inclusions in the right column up by 0.0005 cm and the 12 inclusions in the left column down by 0.0005 cm. Figure 23 shows plots of the interior pressure and streamlines approximated by our solver at the last time step when the inclusions in the 2 columns are staggered. In



(a) Interior Pressure (Ba)

(b) Streamlines

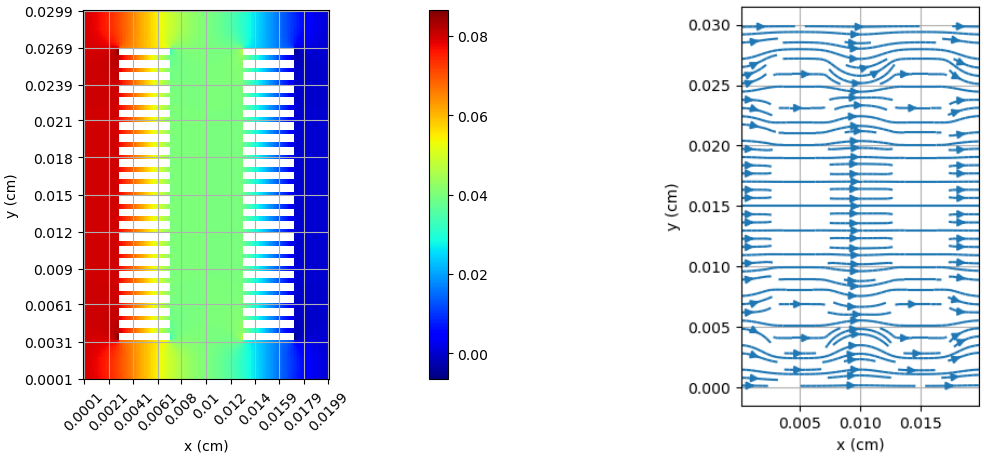
Figure 22: Interior Pressure and Streamline Plots Made from the Output of our Velocity Solver for 24 Aligned Inclusions



(a) Interior Pressure (Ba)

(b) Streamlines

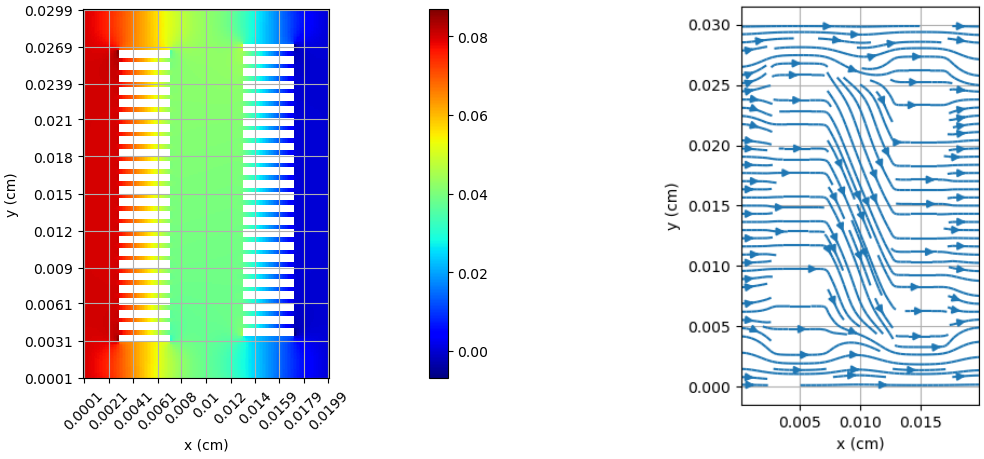
Figure 23: Interior Pressure and Streamline Plots Made from the Output of our Velocity Solver for 24 Staggered Inclusions



(a) Interior Pressure (Ba)

(b) Streamlines

Figure 24: Interior Pressure and Streamline Plots Made from the Output of our Velocity Solver for 48 Aligned Inclusions



(a) Interior Pressure (Ba)

(b) Streamlines

Figure 25: Interior Pressure and Streamline Plots Made from the Output of our Velocity Solver for 48 Staggered Inclusions

the third set up we split 48 inclusions into 2 columns such that the 24 inclusions in each of the columns are horizontally aligned. Figure 24 shows plots of the interior pressure and streamlines for this set up. The bottom inclusions are placed on y range $[0.00325 \text{ cm}, 0.00375 \text{ cm}]$. Inclusions with height 0.0005 cm are placed 0.0005 cm apart up until the highest inclusions on y interval $[0.02625 \text{ cm}, 0.02675 \text{ cm}]$. In our fourth set up, we shift the 24 inclusions in the right column up by 0.00025 cm and the 24 inclusions in the left column down by 0.00025 cm. Figure 25 shows plots of the interior pressure and streamlines for this set up.

Note that the 24 and 48 inclusion aligned set ups shown in Figures 22 and 24 have the distance between the top of the top inclusions and the top boundary equal to the distance between the bottom of the bottom inclusions and the bottom boundary. Similarly, the staggered set ups shown in Figures 23 and 25 have $d_1 = d_4$ and $d_2 = d_3$ for d_i defined for the previous 2 experiments. Furthermore, observe that going from the third experiment to the first 2 set ups, we doubled the number of inclusions but halved the area of each inclusion. Similarly, going from the third experiment to the second 2 set ups, we quadrupled the number of inclusions but quartered the area of each inclusion. Thus, we keep the area of the domain that lies in inclusions the same as in the second and third experiments.

Figure 26 shows a plot of the number of inclusions compared to the hydraulic resistance. When the inclusions are aligned or staggered, the hydraulic resistance increases with the number of inclusions. It is important to note that in these 4 set ups the flow rate error ranged from 0.134 to 0.158 which is above our desired threshold of 0.10. Therefore, we are least confident in these results of experiments 1 through 4. To reduce the error, the step size $h = 0.000125$ cm would need to be halved for these 4 set ups as well as the fully aligned and staggered set ups for 6 and 12 inclusions. We estimate that our solver would take over 2 days to produce results for $h = 0.0000625$ cm.

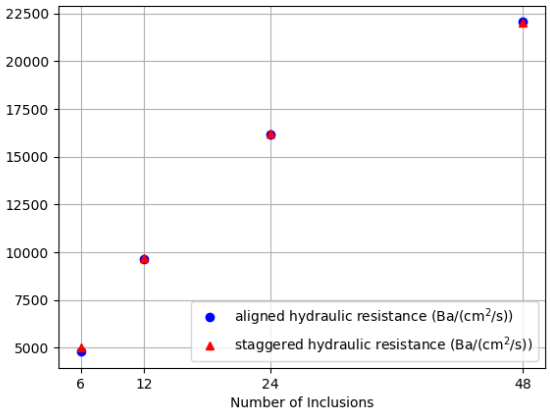


Figure 26: Hydraulic Resistance for Varying Number of Inclusions

4.5 Comparison with Analytic Approximation

We want to see if we can verify the results of our solver for our experiments 1 through 4. We do this using an analytic approximation of hydraulic resistance. Our approximation is built on the idea of treating the channels of fluid flow as resistors [17]. Channels are the areas in the domain where fluid is forced to flow due to the positioning of inclusions. In our case, a channel is rectangular with its length and height being defined by a combination of the surrounding inclusions and domain boundaries. Let $L(C)$ and $H(C)$ denote the length and height of a channel C. We can calculate the resistance $R(C)$ of such a channel using the following formula [2].

$$R(C) = \frac{12\mu L(C)}{H(C)^3} \tag{71}$$

Note that μ is fixed to be 0.01 P in all of our experiments as noted at the beginning of section 4. There are also 2 more formulas [17] taken from electrokinetics that we use. Suppose channels $C_1, C_2, \ldots, C_n$ lie in parallel. Then, the total equivalent resistance is given by:

$$R_{parallel} = \left(\sum_{i=1}^{n} \frac{1}{R(C_i)} \right)^{-1} \tag{72}$$

Suppose alternatively that this set of channels lie in series. Then, the total equivalent resistance is given by:

$$R_{series} = \sum_{i=1}^{n} R(C_i) \tag{73}$$

First, we derive the analytic approximation of the hydraulic resistance for our first experiment where 2 inclusions in a single column with equal horizontal span are moved vertically closer together. In these set ups, we have 3 channels. The first channel C_1 has a vertical span from the bottom domain boundary to the bottom of the lower inclusion and a horizontal span equivalent to the 2 inclusions. The second channel C_2 spans vertically from the top of the lower inclusion to the bottom of the higher inclusion and horizontally the same as C_1. The third channel C_3 spans vertically from the top of the higher inclusion to the top domain boundary and horizontally the same as C_1. Figure 27 (a) gives an example channel setup diagram.

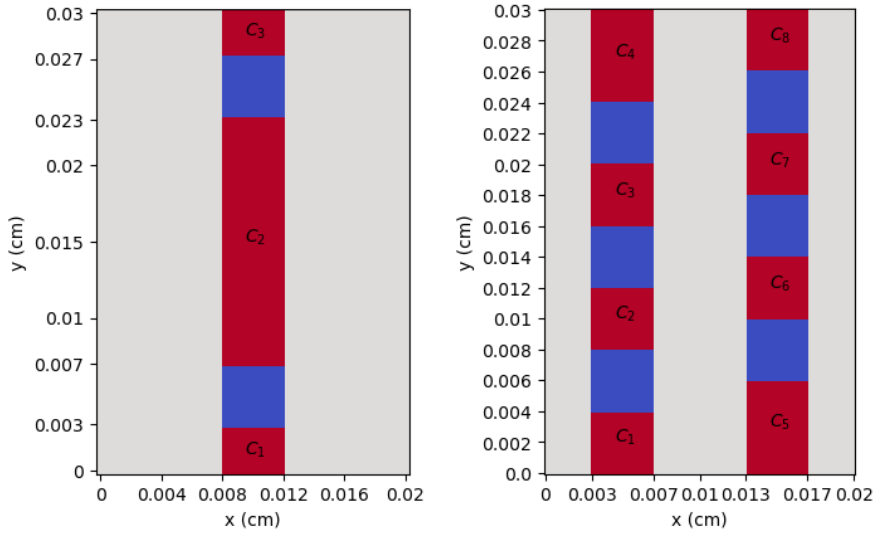
As described on the bottom of page 27, the distance from the bottom boundary to the bottom of the lower inclusion is equal to the distance from the top boundary to the top of the higher inclusion in each of the set ups for the 7 runs in this experiment. Let H_1 be the distance from the 2 inclusions to their respectively closer boundaries and H_2 be the distance between the inclusions. Thus, $H(C_1) = H(C_3) = H_1$ and $H(C_2) = H_2$. From page 29, we know $L(C_i) = 0.004$ cm. Figure 27 (a) is a diagram that shows the set up of C_i in the case

that $H_2 = 0.016$ cm. As we move inclusions closer together, H_2 decreases while H_1 increases. Applying (72) and (73), we can approximate the hydraulic resistance for one of these set ups as:

$$\left(\frac{2H_1^3}{12 \cdot 0.01 \text{ P} \cdot 0.004 \text{ cm}} + \frac{H_2^3}{12 \cdot 0.01 \text{ P} \cdot 0.004 \text{ cm}} \right)^{-1}$$

Figure 28 (a) includes our analytic approximation of the hydraulic resistance for H_2 varied from 0.004 cm to 0.016 cm. With a domain height of 0.03 cm and inclusions of height 0.004 cm each, we can calculate H_1 from H_2 as $H_1 = (0.03 - 0.008 - H_2)/2$ cm. Figure 28 (a) shows the hydraulic resistances calculated from the solver output and approximated using the above strategy normalized to $R_{0.004}$. By this we mean that the analytic approximation and solver approximation of the resistances are normalized to the analytic approximation and solver approximation for the resistance when $H_2 = 0.004$ cm respectively. From it we can see that the analytic approximations do not exactly match the solver results, but they share the same trend.

Next, we derive the analytic approximation of the hydraulic resistance when we have 2 columns of inclusions that may be staggered. As in experiment 1, the inclusions in experiments 2 through 4 all have equal length 0.004 cm. We also know that the vertical distance between inclusions is kept constant in the set ups of each of the experiments. In experiment 2, the distance is 0.004 cm, in experiment 3, the distance is 0.002 cm, and in experiment 4, the distances are 0.001 cm in the case of 24 inclusions and 0.0005 cm in the case of 48. Let the vertical distance between inclusions be called d_I. We also know in each of these experiments that $d_1 = d_4$ and $d_2 = d_3$ for d_i defined previously. When inclusions are fully aligned $d_1 = d_2 = d_3 = d_4$. As inclusions become more staggered, $d_1 = d_4$ decrease and $d_2 = d_3$ increase.



(a) Closing Inclusions (1 column) (b) Staggering Inclusions (2 columns)

Figure 27: Channel Setups for Analytic Approximation of Hydraulic Resistance

Let us derive the analytic approximation in the case that we have N inclusions split over 2 columns. We are interested in $N = 6, 12, 24$, and 48. Let C_1 be the first channel that spans vertically from the bottom domain boundary to the bottom of the lowest left column inclusion and horizontally equivalent to the left column inclusions. Let $C_2, C_3, \ldots, C_{N/2}$ be the channels that lie between the inclusions in the left column. They each span vertically between 2 inclusions and horizontally equivalent to C_1. Let $C_{(N/2)+1}$ be the channel spanning vertically from the top of the highest left inclusion to the top domain boundary and horizontally equivalent to C_1. Let $C_{(N/2)+2}$ be the channel spanning vertically from the bottom domain boundary to the bottom of the lowest right inclusion and horizontally equivalent to the right column inclusions. Let $C_{(N/2)+3}, C_{(N/2)+4}, \ldots, C_{N+1}$ be the channels that lie between the inclusions in the right column. They span vertically between 2 inclusions and horizontally equivalent to $C_{(N/2)+2}$. Let C_{N+2} be the final channel spanning vertically from the top of the highest right inclusion to the top domain boundary and horizontally equivalent to $C_{(N/2)+2}$. Figure 27 (b) shows a diagram of the channel set up for the run from experiment 2 where $d_1 = d_4 = 0.004$ cm and $d_2 = d_3 = 0.006$ cm.

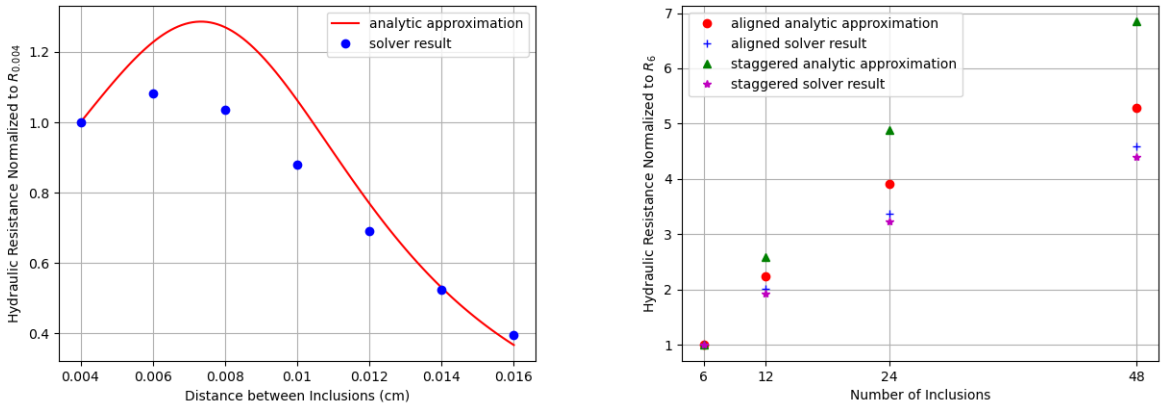
We know that $H(C_1) = d_1$, $H(C_{(N/2)+1}) = d_2$, $H(C_{(N/2)+2}) = d_3$, $H(C_{N+2}) = d_4$, and $H(C_i) = d_I$ for all other i. Thus, applying (72) we can compute the hydraulic resistance of the left column R_L and of the right column R_R.

$$R_L = \left(\frac{d_1^3}{12 \cdot 0.01 \cdot 0.004} + \frac{((N/2)-1)d_I^3}{12 \cdot 0.01 \cdot 0.004} + \frac{d_2^3}{12 \cdot 0.01 \cdot 0.004} \right)^{-1}$$
$$R_R = \left(\frac{d_3^3}{12 \cdot 0.01 \cdot 0.004} + \frac{((N/2)-1)d_I^3}{12 \cdot 0.01 \cdot 0.004} + \frac{d_4^3}{12 \cdot 0.01 \cdot 0.004} \right)^{-1}$$

We can calculate the total hydraulic resistance as $R_L + R_R$ by applying (73). Since $d_1 = d_4$ and $d_2 = d_3$, $R_L + R_R = 2R_L$ and we can write the resistance in terms of N, d_I, d_1 and d_2.

$$R(N, d_I, d_1, d_2) = 2 \left(\frac{d_1^3}{12 \cdot 0.01 \text{ P} \cdot 0.004 \text{ cm}} + \frac{((N/2)-1)d_I^3}{12 \cdot 0.01 \text{ P} \cdot 0.004 \text{ cm}} + \frac{d_2^3}{12 \cdot 0.01 \text{ P} \cdot 0.004 \text{ cm}} \right)^{-1} \tag{74}$$

Figure 28 (b) includes our analytic approximation of the hydraulic resistance for varying number of inclusions in both aligned and staggered cases. For the aligned cases, we include the approximations $R(6, 0.004, 0.005, 0.005)$, $R(12, 0.002, 0.004, 0.004)$,$R(24, 0.001, 0.0035, 0.0035)$, and $R(48, 0.0005, 0.00325, 0.00325)$. For the staggered cases, we include the approximations $R(6, 0.004, 0.003, 0.007)$,$R(12, 0.002, 0.003, 0.005)$,$R(24, 0.001, 0.003, 0.004)$, and $R(48, 0.0005, 0.003, 0.0035)$. Figure 28 (b) shows the hydraulic resistances approximated analytically and using our solver normalized to R_6. By this we mean that the analytic approximations for aligned inclusions, the solver approximations for aligned inclusions, analytic approximations for staggered inclusions, and the



(a) Experiment 1

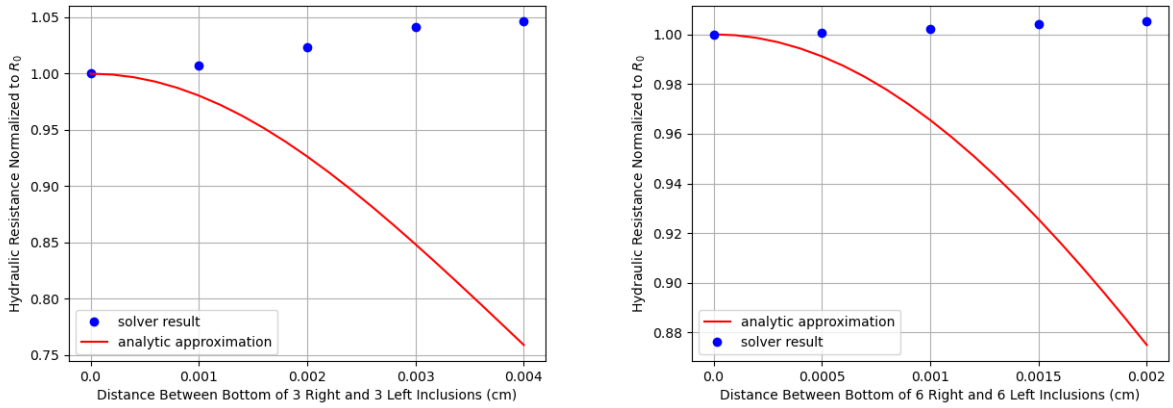
(b) Experiment 4

Figure 28: Normalized Comparison of Analytic and Velocity Solver's Approximation of Hydraulic Resistance for First and Last Experiments

solver approximations for staggered inclusions are normalized by $R(6, 0.004, 0.005, 0.005)$, the solver approximation for 6 aligned inclusions, $R(6, 0.004, 0.003, 0.007)$, and the solver approximation for 6 staggered inclusions respectively. Again, we see that the analytic approximation does not exactly match the solver results, but the two share the same trend in both aligned and staggered cases.

Figure 29 (a) includes our analytic approximation of the hydraulic resistance $R(6, 0.004, d_1, 0.01 - d_1)$ for d_1 from 0.005 cm down to 0.003 cm. Note, here we take $d_2 = 0.01 - d_1$ which matches the relationships of $d_1 = d_3$ and $d_2 = d_4$ for the 5 setups we ran in experiment 2 described in section 4.2. Figure 29 (a) shows the hydraulic resistance approximated analytically and using our solver normalized to R_0. By this we mean all the analytic and solver approximations are normalized to the analytic and solver approximations of the resistance when $d_1 = 0.005$ cm respectively. Recall, $d_1 = 0.005$ cm corresponds to there being 0 staggering. Here we can see that the analytic approximations of resistance decrease when the inclusions are staggered more. This is opposite of the trend of the solver results.

Figure 29 (b) includes our analytic approximation of the hydraulic resistance $R(12, 0.002, d_1, 0.008 - d_1)$ for d_1 from 0.004 cm down to 0.003 cm. Note, here we take $d_2 = 0.008 - d_1$ which matches the relationships of d_1 and d_2 for the 5 setups we ran in experiment 3 described in section 4.3. Figure 29 (b) shows the hydraulic resistance approximated analytically and using our solver normalized to R_0. Similar to above, that means the approximations made using the 2 techniques are normalized to the techniques' respective approximations for the resistance when there is 0 staggering. For the 12 inclusion setups, there is 0 staggering when $d_1 = 0.004$ cm. Here, like in the case of Figure 29 (a), the trend of the analytically approximated resistance is opposite of that of the solver approximations.



(a) Experiment 2

(b) Experiment 3

Figure 29: Normalized Comparison of Analytic and Velocity Solver's Approximation of Hydraulic Resistance for Second and Third Experiments

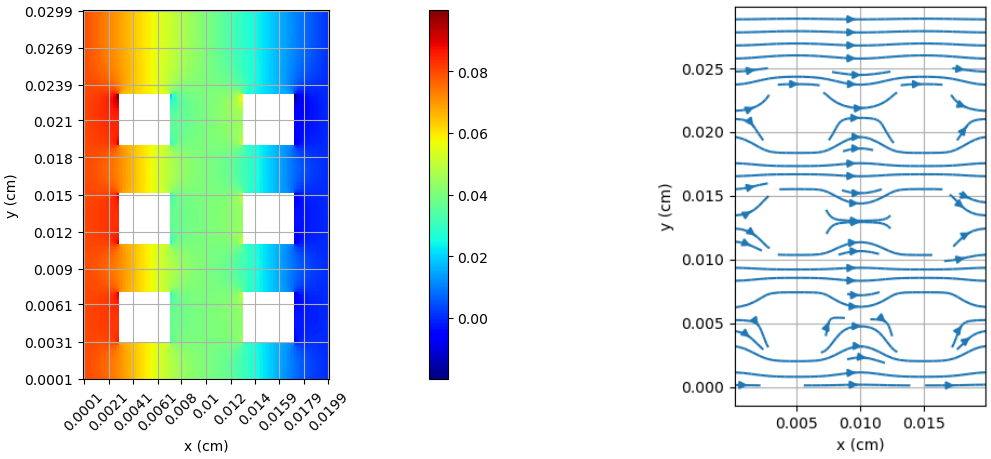
To understand this discrepancy, let us return to equation (74), our resistance approximation in the case of inclusions split over 2 columns. In particular, we are interested in how $R(N, d_I, d_1, d_2)$ changes when inclusions become more staggered for fixed N and d_I. In this case, seeing how $R(N, d_I, d_1, d_2)$ behaves with more staggering is equivalent to seeing how $(d_1^3 + d_2^3)^{-1}$ behaves as d_1 decreases and d_2 increases. In our experiments, staggering is applied by starting $d_1 = d_2 = d$ and then each staggering step is equivalent to $d_1 = d - \Delta d$ and $d_2 = d + \Delta d$. We are interested in seeing how $((d - \Delta d)^3 + (d + \Delta d)^3)^{-1}$ behaves for Δd increasing from 0. Expanding terms shows that $((d - \Delta d)^3 + (d + \Delta d)^3)^{-1} = (2d^3 + 6d\Delta d^2)^{-1}$. One of the effects of staggering inclusions is that we create a larger channel at the top of the left column and at the bottom of the right column. Due to the fact that $(2d^3 + 6d\Delta d^2)^{-1}$ decreases with increasing Δd, this implies that our analytic approximation is only taking into account the effect of the larger channels produced by staggering inclusions. We believe that the analytic approximation is essentially treating staggering as equivalent to gradually growing 2 larger channels at the same time as shrinking 2 others. We want to see if our solver also reports that the resistance decreases when we do not stagger columns but instead grow 2 channels while shrinking 2 others. To do this, we perform 2 additional experiments.

In the first of these experiments, we start with the same setup as in the second experiment. However, instead of shifting the 3 left inclusions down and the 3 right inclusions up by 0.0005 cm 4 times, we shift all inclusions in both columns down by 0.0005 cm 4 times. By doing this, the sizes of the channels for each run in this experiment are equivalent to the sizes of the channels in the runs of experiment 2. However, here the 2 growing channels are both at the top of the domain while both shrinking channels are at the bottom. Figure 15 shows the interior pressure and streamlines approximated by our solver at the last time step for the setup of the first run of this experiment. Figure 30 shows plots of the interior pressure and

streamlines approximated by our solver at the last time step when the inclusions in the 2 columns have been fully shifted downward. The inclusions lie in 2 columns on the same x ranges as in experiments 2 through 4: [0.003 cm, 0.007 cm] and [0.013 cm, 0.017 cm]. The inclusions in both columns lie on y ranges [0.003 cm, 0.007 cm],[0.011 cm, 0.015 cm], and [0.019 cm, 0.023 cm]. Starting from the set up in Figure 15, we move the inclusions in both columns down by 0.0005 cm 4 times to end up with the set up in Figure 30.

Figure 32 (a) includes the resistance as approximated by our solver as a function of the distance between the highest left inclusion and the top domain boundary for our 5 set ups. In the figure, the resistance values are normalized to the resistance approximated by our solver when the inclusions are aligned and symmetrically distant from the domain boundaries. To make an analytic approximation of the resistance as the aligned inclusions are moved down, we can make use of a similar channel set up to the one described on page 39. The only difference is that $d_1 = d_3$ and $d_2 = d_4$ as opposed to $d_1 = d_4$ and $d_2 = d_3$. From (74), we can see that this means the resistance of a staggered setup for a particular d_1 and d_2 is equivalent to the resistance of a shifted aligned setup for the same d_1 and d_2. Figure 32 (a) demonstrates this by showing a plot of $R(6, 0.004, d_1, 0.01 - d_1)$ for d_1 from 0.005 cm down to 0.003 cm. Again, in these experiments, $d_2 = 0.01 - d_1$. In Figure 32 (a) these approximations are normalized by $R(6, 0.004, 0.005, 0.005)$. Here we can see that the solver results, like the analytic approximation, show that the resistance decreases as we widen 2 channels and shrink 2 others in the manner described above. We are confident in the results of our solver because its flow rate error remained between 0.034 and 0.043 for these 5 runs.

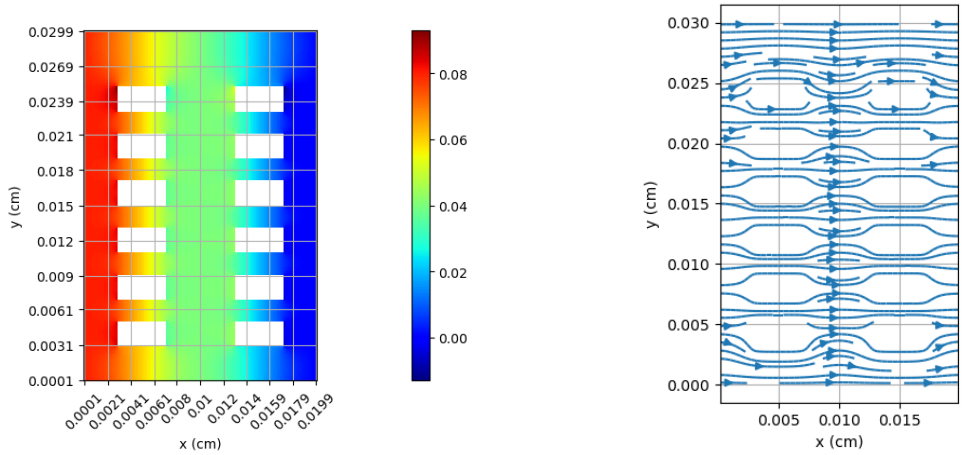
In the second of our new experiments, we start with the same setup as the third experiment. However, instead of shifting the 6 left inclusions down and the 6 right inclusions up by 0.00025 cm 4 times, we shift all inclusions in both columns down by 0.00025 cm 4 times. By doing this, the sizes of the channels for each run in this experiment are equivalent to the sizes of the channels in the runs of experiment 3. However, here the 2 growing channels are both at the top of the domain while both shrinking channels are at the bottom. Figure 17 shows the interior pressure and streamlines approximated by our solver at the last time step for the setup of the first run of this experiment. Figure 31 shows plots of the interior pressure and streamlines approximated by our solver at the last time step when the inclusions have been fully shifted downward. The inclusions lie in 2 columns on the same x ranges as in experiments 2 through 4: [0.003 cm, 0.007 cm] and [0.013 cm, 0.017 cm]. The inclusions in both columns lie on y ranges [0.003 cm, 0.005 cm],[0.007 cm, 0.009 cm],[0.011 cm, 0.013 cm],[0.015 cm, 0.017 cm],[0.019 cm, 0.021 cm], and [0.023 cm, 0.025 cm]. Starting from the set up in Figure 17, we move the inclusions in both columns down by 0.00025 cm 4 times to end up with the set up in Figure 31.



(a) Interior Pressure (Ba)

(b) Streamlines

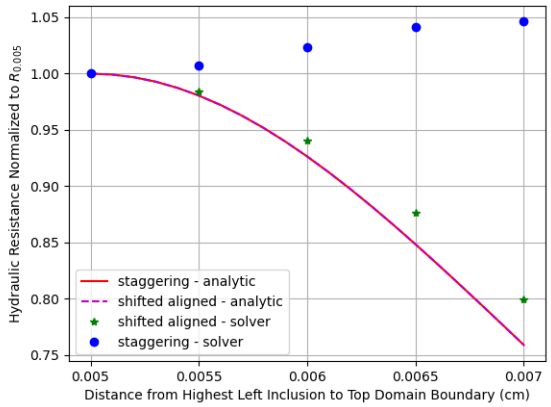
Figure 30: Interior Pressure and Streamline Plots Made from the Output of our Velocity Solver for 6 Shifted Aligned Inclusions



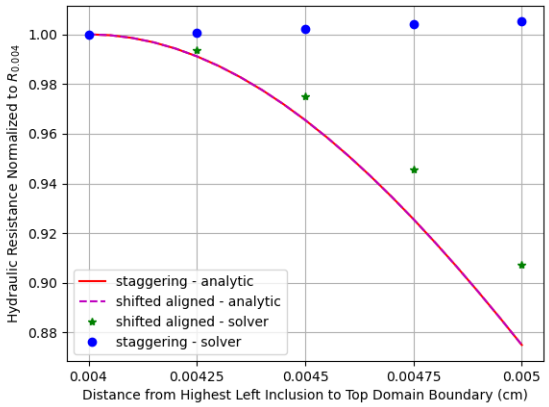
(a) Interior Pressure (Ba)

(b) Streamlines

Figure 31: Interior Pressure and Streamline Plots Made from the Output of our Velocity Solver for 12 Shifted Aligned Inclusions



(a) Shifting 6 Aligned Inclusions



(b) Shifting 12 Aligned Inclusions

Figure 32: Normalized Comparison of Analytic and Velocity Solver's Approximation of Hydraulic Resistance for Shifting Aligned Inclusions

Figure 32 (b) includes the resistance as approximated by our solver as a function of the distance between the highest left inclusion to the top domain boundary for our 5 set ups. In the figure, the resistance values are normalized to the resistance approximated by our solver when the inclusions are aligned and symmetrically distant from the domain boundaries. As in the case of 6 inclusions, the staggered and shifted aligned setups for a particular d_1 and d_2 have equal resistance. Figure 32 (b) demonstrates this by showing a plot of $R(12, 0.002, d_1, 0.008 - d_1)$ for d_1 from 0.004 cm down to 0.003 cm. In these experiments, $d_2 = 0.008 - d_1$. In Figure 32 (b) these approximations are normalized by $R(2, 0.002, 0.004, 0.004)$. Here we see, like in the previous experiment, that the solver results have the same trend as the analytic approximation. We are confident in the results of our solver because its flow rate error remained between 0.079 and 0.086 for these 5 runs.

Let us return now to the difference in the analytic and solver approximations for staggered inclusions. If one were to look only at the streamline plots in the case of staggered inclusions, we would be inclined to think that the resistance only increases over D with more staggering because the streamlines become more vertical. The more vertical streamlines indicate that the fluid is being forced to more drastically change its path through D. One could look at Figures 16 (b), 18 (b), 23 (b), and 25 (b) for examples of this. However, the analytic approximation, despite being inaccurate, shows us that there is a second change in fluid flow with staggering. That is, with the creation of larger channels, the resistance also is decreasing as more fluid can flow through those larger channels. This theory is supported by the results of our 2 new experiments. Thus, staggering introduces 2 effects on fluid flow that counteract each other. We have discovered that the analytic approximation, despite being much easier to compute, does not capture both of these effects. Our velocity solver on the other hand is able to capture both effects of staggered inclusions on the resistance.

This demonstrates why our solver is important: it approximates the behavior of a fluid throughout the entire domain.

5 Conclusions

Modelling the flow of fluids is an important task in the creation of microfluidic devices. This thesis is particularly beneficial to the study of constructing devices that use obstacles to manipulate fluid flow. At the beginning of this thesis, we formulated our task as solving the incompressible Navier-Stokes equations subject to certain boundary conditions. Then, we went on to describe our construction of a solver to Poisson's equation. After that, we created our fluid velocity solver. Both solvers were constructed with the idea in mind of allowing for arbitrary placement of inclusions. We also discussed benchmarks we performed on our solvers as well as evidence of issues that we discovered. Lastly, we described a series of computer experiments we conducted with our fluid solver on domains with various inclusion arrangements. In our last section, we compared an analytic approximation of hydraulic resistance with an approximation computed from our solver's output. From this comparison we discovered that our solver is able to provide more accurate approximations of the resistance for certain arrangements. Experiments 2 and 3 provide clear examples of studies that greatly benefit from a fluid solver like ours.

5.1 Directions of Future Work

There are a variety of areas in which one could expand on this work. The most obvious being further analysis of our solver to identify if the errors in our solver are due to the approach we took or are due to mistakes in the Python implementation. There is also room to improve the method driving the velocity solver. The method we use for solving the Navier-Stokes equations is explicit but one could study implicit methods as replacements. There are also ways the solver could be sped up. As described in sections 2.4 and 3.1, our strategies for approximating the gradient involve iteration with nested loops. This could definitely be a bottleneck for our fluid solver as gradients are approximated at each velocity solver time step. If these computations could be expressed in terms of matrix multiplications this may allow for a faster solver. This may be especially useful if our implementation was ported to a Python implementation that uses GPUs. Currently, our solver is built primarily on NumPy and SciPy which are both restricted to run on CPUs only. GPUs may prove useful in speeding up matrix related operations.

There are also ways one could add to the existing solver. One area, in which we have already started to work on, is incorporating the immersed boundary method [12] into the solver to model the movement of an immersed boundary cell. So far, we have ported an existing MATLAB implementation for domains without

inclusions into Python. The ported code still needs further modifications to be fully incorporated with our solver. In section 4, we studied a variety of arrangements of rectangular inclusions, but one could definitely study more. For example, one may be interested in the effects of the horizontal distance between the 2 columns of inclusions discussed in the various experiments in section 4. One could also investigate adding support to our solver for non rectangular inclusions in the domain. As seen in the work done by Torino et. al [14], obstacles in microfluidic devices can take on a variety of shapes.

Acknowledgements

We would like to thank Brandeis University postdoctoral fellow Ying Zhang for providing us with her MATLAB implementation for the Immersed Boundary Method which we have ported to Python.

References

[1] Patrice Bacchin, Quentin Derekx, Damien Veyret, Karl Glucina, and Philippe Moulin. Clogging of microporous channels networks: role of connectivity and tortuosity. *Microfluidics and nanofluidics*, 17(1):85–96, 2014.

[2] Cx K Batchelor and GK Batchelor. *An introduction to fluid dynamics*. Cambridge university press, 2000.

[3] Gangadhar Eluru, Pavan Nagendra, and Sai Siva Gorthi. Microfluidic in-flow decantation technique using stepped pillar arrays and hydraulic resistance tuners. *Micromachines*, 10(7):471, 2019.

[4] Matteo Giacomini and Ruben Sevilla. Discontinuous galerkin approximations in computational mechanics: hybridization, exact geometry and degree adaptivity. *SN Applied Sciences*, 1(9):1–15, 2019.

[5] Michael T Heath. *Scientific computing: an introductory survey, revised second edition*. SIAM, 2018.

[6] Medusa: Coordinate Free Meshless Method implementation. Poisson's equation — medusa: Coordinate free mehless method implementation,, 2021. [Online; accessed 11-April-2022].

[7] Hans Johnston and Jian-Guo Liu. Finite difference schemes for incompressible flow based on local pressure boundary conditions. *Journal of Computational Physics*, 180(1):120–154, 2002.

[8] Gess Kelly and Thomas G. Fai. Multiscale model of clogging in microfluidic devices with grid-like geometries, 2021.

[9] Xiaoye S. Li. An overview of SuperLU: Algorithms, implementation, and user interface. *ACM Transactions on Mathematical Software*, 31(3):302–325, September 2005.

[10] Daniel Mark, Stefan Haeberle, Günter Roth, F von Stetten, and Roland Zengerle. Microfluidic lab-on-a-chip platforms: requirements, characteristics and applications. *Microfluidics based microsystems*, pages 305–376, 2010.

[11] Cleve B Moler. *Numerical computing with MATLAB*. SIAM, 2004. [Online; accessed 27-April-2022].

[12] Charles S Peskin. The immersed boundary method. *Acta numerica*, 11:479–517, 2002.

[13] Jure Slak and Gregor Kosec. Medusa: A c++ library for solving pdes using strong form mesh-free methods. *ACM Transactions on Mathematical Software*, 2021.

[14] S Torino, M Iodice, I Rendina, G Coppola, and E Schonbrun. Hydrodynamic self-focusing in a parallel microfluidic device through cross-filtration. *Biomicrofluidics*, 9(6):064107, 2015.

[15] Lloyd N Trefethen and David Bau III. *Numerical linear algebra*, volume 50. Siam, 1997.

[16] Pauli Virtanen, Ralf Gommers, Travis E. Oliphant, Matt Haberland, Tyler Reddy, David Cournapeau, Evgeni Burovski, Pearu Peterson, Warren Weckesser, Jonathan Bright, Stéfan J. van der Walt, Matthew Brett, Joshua Wilson, K. Jarrod Millman, Nikolay Mayorov, Andrew R. J. Nelson, Eric Jones, Robert Kern, Eric Larson, C J Carey, İlhan Polat, Yu Feng, Eric W. Moore, Jake VanderPlas, Denis Laxalde, Josef Perktold, Robert Cimrman, Ian Henriksen, E. A. Quintero, Charles R. Harris, Anne M. Archibald, Antônio H. Ribeiro, Fabian Pedregosa, Paul van Mulbregt, and SciPy 1.0 Contributors. SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. *Nature Methods*, 17:261–272, 2020.

[17] Wikibooks. Microfluidics/hydraulic resistance and capacity — wikibooks, the free textbook project, 2019. [Online; accessed 20-April-2022].