12-2017

Cross Flow Filtration Modeling Using Analytical and Numerical Solutions Along with Implementation as a Web-Based Calculator

Lauren Paige McIntyre
Clemson University

Cross Flow Filtration Modeling Using Analytical and Numerical Solutions along with Implementation as a Web-Based Calculator

---

A Thesis
Presented to
the Graduate School of
Clemson University

In Partial Fulfillment
of the Requirements for the Degree
Master of Science
Mathematics

---

by
Lauren Paige McIntyre
December 2017

Accepted by:
Dr. Christopher Cox, Committee Chair
Dr. Nicos Andreas
Dr. Qingshan Chen
Dr. Matthew Saltzman

Abstract

In this thesis, we present a 1D model for the complex phenomenon of cross-flow filtration. We begin by developing the governing equations and providing analytical solutions that can be found using Laplace transforms for both the simple clean filter case, and the more complex filter with fouling in the form of cake, and depth plugging of the filter media. A walk through of the set up and implementation of a simple web-based application is given, and the code to produce the web application for this model is included in an appendix. The web application acts as a calculator, accepting model parameters and returning graphical output on the client side while the numerical solution is calculated on the server side. Lastly technique of a least-squares finite element approach is applied to the governing equations to obtain approximate solutions now under the assumption that viscosity is not constant, but varies linearly with respect to time. The major contribution from this thesis is the development of a web-based application for simulation of cross flow filtration, set in a framework that can be applied for a wide variety of modeling problems. This thesis is a significant step towards the long term goal to combine multiple disciplines including fluid dynamics, mathematics, and computer science, to produce an effective and robust modeling tool.

Acknowledgments

Support from the South Carolina Space Grant Research and Education Awards Program is gratefully acknowledged (Project title Regenerable Cross Flow Filtration for Manned Mission to Mars, PTE Federal Award Number NNX15AL49H, Subaward Number 521179-RP-CMCox).

I would also like to thank Ashwin Srinath and Dr. Matthew Saltzman for their insight and help with the web application, and Dr. Nicos Andreas for his expertise and vision for this work.

I would also like to thank my advisor, Dr. Chris Cox, for all of his support, patience, and guidance and especially for encouraging me to apply for my job at NASA.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

Cross flow filtration is a filtration technique where the initial carrier fluid passes over the filter media in a tangential direction, as opposed to dead-end filtration where the carrier fluid flows directly into the filter media in the perpendicular direction. In filtration, the carrier fluid flows into the filter, and the clean, desirable fluid exits the filter as permeate. Any fluid that does not exit the filter as permeate, or become trapped in the filter exits as retentate. As the carrier fluid moves across the filter, debris becomes trapped in the media's pores, allowing the permeate to flow through the media, and the dispersed phase to continuously concentrate into a slurry. The benefit of cross flow filters is that, given correctly chosen design parameters, it is possible for them to operate indefinitely, eventually achieving near constant permeate flow. This happens because the tangential flow washes away a significant portion of the filter cake, eliminating the need for cleaning and/or back flushing. Though to achieve this steady state, the media fouling must be drastically reduced, or all together prevented.

The driving force behind cross flow filtration is the pressure drop on the permeate side of the media. This pressure difference pulls the permeate through the media, leaving the remainder of the material which is larger than the pore size in the slurry, which can then be recovered for further processing. The media is chosen to have high separation efficiency, with pore size distribution intended to avoid fouling. For more background information on cross flow filtration, including applications, see [11], [17], [16], [22].

Cross flow filters are of particular interest to NASA because of their ability to achieve a dynamical fluid equilibrium and their remarkable media stacking density. These qualities will allow

cross flow filters to readily adapt to meet NASA mission specific objectives, such as minimum volume footprint and minimum lift off mass. Regenerable filters (such as dead end filters) have to be cleaned, or replaced and are of no use to NASA in space flight applications.

This thesis is outlined as follows: a model for cross-flow filtration is proposed for two filter domains and the analytical solutions for the clean filter are given. Next the model is modified to include fouling of the filter media in the form of cake build up and depth plugging, which also has analytical solutions. Next, a walk through of the set up of a simple web application, one that produces a sine wave, is given. The web application that was developed for this thesis is then presented, showing the interface and capabilities of the application, which models the filter with fouling. Lastly the assumption that viscosity must be constant is lifted, and a least-squares finite element method is applied to approximate a solution.

Chapter 2

Model

2.1 Geometries

We consider cross flow filters of both cylindrical and rectangular shape. The cylindrical domain shown in Figure 2.1 represents a filter with one tube determined by parameters of inside diameter D, length L, and media thickness H.

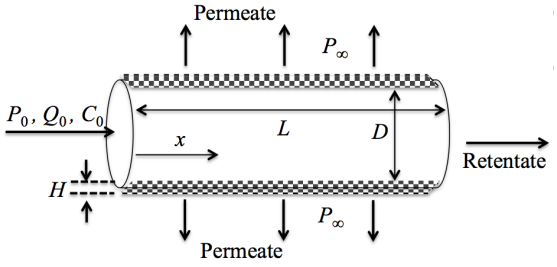

Figure 2.1: Schematic of a cylindrical cross flow filter

The rectangular domain shown in Figure 2.2 can be specified by the parameters of width W, length L, half-height of the flow domain d, and thickness of the filter media H.

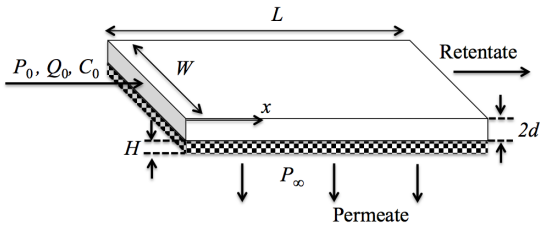

Figure 2.2: Schematic of a rectangular cross flow filter

In each figure, the grid pattern represents the filter media. In the cylindrical geometry, the media covers the entire circumference, and in the rectangular geometry either one or both walls may be filter media. We allow for the selection of number of permeable walls (1 or 2) in our model. The carrier fluid enters through an inlet of circular or rectangular shape at the left side of the filter, and permeate flows out of the filter media in the perpendicular direction. The remaining fluid and debris that does not filter to permeate, or is not deposited as cake or in the medias pores, exits the filter as retentate.

We denote x as the distance from the filter inlet in the direction of flow. The volumetric flow rate at the inlet is denoted Q_0, initial dispersed phase concentration as C_0, and pressure at the inlet as P_0. P_∞ represents permeate pressure.

2.2 The Clean Filter

We begin by examining the clean filter case, in which we assume our filter experiences no fouling effects. While this is not a realistic model, we believe the clean filter presents an important starting point. The clean filter case gives us a benchmark for expanding our model to consider fouling, the transient case, and even more complicated models without analytical solutions. All of these mathematical models at initial time $t = 0$ should reduce to this clean filter case with zero fouling. We consider crossflow filters for both the cylindrical and rectangular geometries. A

development of these equations is given in [4] and [7].

Mathematical Models

Define the volumetric flow rate and pressure at the filter inlet as Q_0 and P_0 respectively. Specific permeability is given by Γ, and pressure, $P(x)$, permeate flux, $\varphi(x)$, and concentrated slurry flow rate, $Q(x)$, as functions of position x. If we assume there is zero fouling of the media, that the flow is laminar, and the fluid incompressible, then for the steady state we have analytical solutions for $P(x)$, $\varphi(x)$, $Q(x)$. Using the governing equations above, we set up the following initial value problems, specifying the initial conditions at $x = 0$ for pressure, permeate flux, and volumetric flow.

Parameter	Description	Units
$P(x)$	pressure inside filter at position x	Pa
$\varphi(x)$	permeate flux at position x	$\mathrm{m}^3/(sec - m^2)$
$Q(x)$	volumetric flow rate inside filter at position x	m^3/sec
P_∞	permeate discharge pressure	Pa
H	filter media thickness	m
μ	viscosity	Pa-sec
Γ	specific media permeability	m^2
D	inside diameter	m
d	filter flow domain half-height	m
W	filter width	m
n_w	number of permeable walls (1 or 2)	

Table 2.1: Parameter values the clean filter

The governing equations for the cylindrical domain are seen below. The Darcy equation, Hagen-Poiseuille, and the continuity equation respectively:

$$P(x) - P_\infty = \frac{\mu H}{\Gamma}\varphi(x) \tag{2.1}$$

$$\frac{dP(x)}{dx} + \frac{128\mu}{\pi D^4}Q(x) = 0 \tag{2.2}$$

$$\frac{dQ(x)}{dx} + \pi D\varphi(x) = 0 \tag{2.3}$$

$$P(0) = P_0, \qquad \varphi(0) = \varphi_0, \qquad Q(0) = Q_0 \tag{2.4}$$

Governing equations for the rectangular domain:

$$P(x) - P_\infty = \frac{\mu H}{\Gamma}\varphi(x) \tag{2.5}$$

$$\frac{dP(x)}{dx} + \frac{3\mu}{2d^3W}Q(x) = 0 \tag{2.6}$$

$$\frac{dQ(x)}{dx} + n_w W\varphi(x) = 0 \tag{2.7}$$

$$P(0) = P_0, \qquad \varphi(0) = \varphi_0, \qquad Q(0) = Q_0 \tag{2.8}$$

Analytical Solutions

The solutions to both geometries are found by applying the method of Laplace transforms to the sets of governing equations, (2.1), (2.2), and (2.3) for cylindrical and (2.5), (2.6), and (2.7) for rectangular [4], [7]. This results in the following analytical solutions. The solutions for the volumetric flow, $Q(x)$, pressure $P(x)$, and permeate flux $\varphi(x)$ are identical up to geometry terms. Analytical solutions for cylindrical domain

$$Q(x) = -Q_{\text{ref}} \sinh\frac{x}{\lambda_c} + Q_0 \cosh\frac{x}{\lambda_c} \tag{2.9}$$

$$P(x) = P_\infty + (P_0 - P_\infty)\cosh\frac{x}{\lambda_c} - \Delta P_{ref}\sinh\frac{x}{\lambda_c} \tag{2.10}$$

$$\varphi(x) = \frac{Q_{\text{ref}}}{\pi D\lambda_c}\cosh\frac{x}{\lambda_c} - \frac{Q_0}{\pi D\lambda_c}\sinh\frac{x}{\lambda_c} \tag{2.11}$$

$$\lambda_c = \sqrt{\frac{D_3 H}{128\Gamma}}, \qquad \Delta P_{\text{ref}} = \frac{Q_0\mu H}{\pi D\lambda_c\Gamma}, \qquad Q_{\text{ref}} = \frac{\pi D^4}{128\mu\lambda_c}(P_0 - P_\infty) \tag{2.12}$$

Analytical solutions for rectangular domain

$$Q(x) = -Q_{ref}\sinh\frac{x}{\lambda_c} + Q_0\cosh\frac{x}{\lambda_c} \tag{2.13}$$

$$P(x) = P_\infty + (P_0 - P_\infty)\cosh\frac{x}{\lambda_c} - \Delta P_{\text{ref}}\sinh\frac{x}{\lambda_c} \tag{2.14}$$

$$\varphi(x) = \frac{Q_{\text{ref}}}{n_w W\lambda_c}\cosh\frac{x}{\lambda_c} - \frac{Q_0}{n_w W\lambda_c}\sinh\frac{x}{\lambda_c} \tag{2.15}$$

$$\lambda_c = \sqrt{\frac{2d^3 H}{3n_w\Gamma}}, \qquad \Delta P_{\text{ref}} = \frac{Q_0\mu H}{n_w\Gamma\lambda_c W}, \quad {}_{\text{ref}} = \frac{2d^3 W}{3\mu\lambda_c}(P_0 - P_\infty) \tag{2.16}$$

From these solutions we are able to identify several useful design parameters, $\lambda_c, \Delta P_{\text{ref}}$, and Q_{ref}. λ_c, which is determined by the geometry of the filter, the media thickness, and the media permeability, gives a system of characteristic length. ΔP_{ref} represents the pressure drop of the system, and Q_{ref}

the reference flow rate. Using this result it can be shown that so long as $Q_0 \neq Q_{\text{ref}}$, there exists a limiting value L_∞ which the filter length cannot exceed [4], [7]. Using this assumption that $L < L_\infty$ we know that we can model pressure drop in parametric form in terms of characteristic length and reference pressure drop.

Consider the cylindrical case:

$$\Delta P = P_0 - P(L) = (P_0 - P_\infty)(1 - \cosh\frac{L}{\lambda_c}) + \Delta P_{\text{ref}} \sinh\frac{L}{\lambda_c} \tag{2.17}$$

We can also calculate the average permeate flux by integrating over the length of the tube, from equation (2.3):

$$\begin{aligned}
\varphi_{\text{avg}} &= \frac{1}{L}\int_0^L \varphi(x)dx \\
&= \frac{1}{L}\int_0^L (\frac{Q_{\text{ref}}}{\pi D\lambda_c}\cosh\frac{x}{\lambda_c} - \frac{Q_0}{\pi D\lambda_c}\sinh\frac{x}{\lambda_c})dx \\
&= \frac{Q_{\text{ref}}}{\pi DL}\sinh\frac{L}{\lambda_c} - \frac{Q_0}{\pi DL}(\cosh\frac{L}{\lambda_c} - 1)
\end{aligned} \tag{2.18}$$

By taking n_g, the parameter for gravimetric separation efficiency, as $n_g = 1$, i.e. the debris is perfectly efficiently separated from the dispersed phase so that we have no fouling, we can derive an equation for the concentration of the slurry $C(x)$ by applying the continuity equation to the dispersed phase material. Here $C(x)$ denotes the concentration of the slurry at any point along the domain.

$$\frac{d(Q(x)C(x))}{dx} + \pi D\varphi(x)(1 - n_g)C(x) = 0 \tag{2.19}$$

Since we are assuming that no fouling is occurring, with $n_g = 1$, notice that equation (2.19) reduces to (2.20), and we can solve for $C(x)$ by integrating

$$\begin{aligned}
0 &= \frac{d(Q(x)C(x))}{dx} \\
&= \int_0^z \frac{d}{dx}Q(x)C(x)dz \quad = Q(z)C(z) - Q_0C_0
\end{aligned} \tag{2.20}$$

Solving for $C(x)$ and substituting equation (2.9)

$$C(x) = \frac{Q_0 C_0}{Q(x)} = \frac{Q_0 C_0}{Q_0 \cosh \frac{x}{\lambda_c} - Q_{\text{ref}} \sinh \frac{x}{\lambda_c}} \tag{2.21}$$

We can therefore predict the slurry concentration at any point x, along the length of the tube.

2.3 Fouling Effects

Now we examine the fouling effects that occur during cross-flow filtration, in particular surface cake and depth plugging. Any debris trapped by the filter, which does not pass through the media as permeate, is either concentrated as cake on the media's surface, or has penetrated the media and become trapped inside. It is most desirable for debris to accumulate as cake on the filter surface, as the cake can theoretically be periodically removed as needed in situations where almost constant permeate flow is desired over a long length of time. Debris accumulation as surface cake does not permanently affect the performance of the filter media, extending the life of the media and reducing the probability that it will need to be changed or replaced.

However, it is likely that even with the most carefully chosen filter media, some debris will also become trapped within the filter pores. If a significant amount of debris plugs the filter pores, media performance will decline, this progressive and irreversible effect can lead to unacceptable permeate flow, requiring the filter media to be removed and extensively cleaned, or altogether replaced, resulting in either unwanted downtime or unwanted costs. The modeling of the phenomena of surface cake buildup or depth plugging is clearly transient as it takes time for the debris to accumulate. Our goal in modeling these fouling effects is to understand how fouling relates to filter design so that media can be chosen to mitigate undesirable effects and prolong filter life. We consider the rectangular domain, as the results for either domain have been shown to be equivalent up to geometry parameters.

Solutions in the Presence of Surface Cake and Depth Plugging

Following the development in [9] we introduce two new variables, $\xi(t)$ and $C_d(t)$, which represent the cake thickness and dispersed phase concentration in filter media, respectively. For this model we operate under the assumptions that cake thickness, $\xi(t)$, and dispersed phase trapped in the pores, $C_d(t)$, are dependent only on time. We also assume that given ρ_d, the density of dispersed

phase, $C_d(t) \ll \rho_d$ and that the cake is incompressible. To find solutions for $\xi(t)$ and $C_d(t)$ we begin with the linear mass conservation equation. Note that all the parameters except the ones we are solving for are constant, that surface area $A = WL$, and ϵ represents porosity. The linear mass conservation equation simply says that all of the debris entering the filter must either build up in the cake bed, become trapped in the media pores, penetrate entirely through the media and exit as permeate, or exit the filter as retentate [9].

The linear mass conservation equation:

$$\begin{array}{ccccccccc} Q_0C_0t & = & n_wA\rho_c\xi(t) & + & n_wAH\epsilon C_d(t) & + & Q_\infty C_\infty t & + & Q_LC_Lt \\ \text{inflow} & & \text{cake bed} & & \text{depth plugging} & & \text{permeate} & & \text{retentate} \end{array}$$

To solve for the cake thickness and dispersed phase material trapped in pores we consider the gravimetric efficiency η_g defined as,

$$\eta_g = \frac{\text{debris collected}}{\text{debris fed}}$$

With gravimetric efficiency, we can separate the linear mass conservation equation into two parts, one to represent the debris trapped in cake and pores, and another to represent the debris exiting the filter as permeate and retentate as follows.

Cake and pores:

$$\eta_g Q_0 C_0 t = n_w A \rho_c \xi(t) + n_w AH\epsilon C_d(t) \tag{2.22}$$

retentate and permeate:

$$(1-\eta_g) Q_0 C_0 t = Q_\infty C_\infty t + Q_L C_L t \tag{2.23}$$

which for $t > 0$ reduces to:

$$(1-\eta_g) Q_0 C_0 = Q_\infty C_\infty + Q_L C_L \tag{2.24}$$

The distribution function $\Theta(t)$ represents the gravimetric ration of the mass in cake to the mass

trapped in the pores [3]:

$$\Theta(t) = \frac{\rho_c n_w A \xi(t)}{\epsilon n_w A H C_d(t)} = \frac{\rho_c \xi(t)}{\epsilon H C_d(t)} \tag{2.25}$$

Now using equations (2.22) and (2.25), results are obtained for the cake layer thickness and concentration of dispersed phase in filter media.

$$\begin{aligned}
\xi(t) &= \frac{\eta_g Q_0 C_0 t - n_w A H \epsilon C_d(t)}{n_w A \rho_c} \\
&= \frac{\eta_g Q_0 C_0 t}{n_w A \rho_c} - \frac{H \epsilon \rho_c \xi(t)}{\rho_c \Theta(t) \epsilon H} \\
&= \frac{\eta_g Q_0 C_0}{n_w A \rho_c} - \frac{\xi(t)}{\Theta(t)} \\
&= \frac{\eta_g Q_0 C_0}{n_w A \rho_c} t * \frac{\Theta(t)}{\Theta(t)+1} \\
&= \frac{K\Theta}{\Theta+1} t
\end{aligned} \tag{2.26}$$

$$\begin{aligned}
C_d(t) &= \frac{\eta_g Q_0 C_0 t}{n_w A H \epsilon} - \frac{n_w A \rho_c \xi(t)}{n_w A H \epsilon} \\
&= \frac{\eta_g Q_0 C_0}{n_w A H \epsilon} t - \frac{\rho_c \Theta(t) H \epsilon C_d(t)}{H \epsilon \rho_c} \\
&= \frac{\eta_g Q_0 C_0}{n_w A H \epsilon} t - \Theta(t) C_d(t) \\
&= \frac{\eta_g Q_0 C_0}{n_w A H \epsilon} \frac{t}{1+\Theta(t)} \\
&= \frac{K_d}{1+\Theta} t
\end{aligned} \tag{2.27}$$

Where the constants K and K_d are defined as:

$$K = \tfrac{\eta_g Q_0 C_0}{n_w A \rho_c}, \qquad K_d = \tfrac{\eta_g Q_0 C_0}{n_w A H \epsilon}$$

Andreas provides a modified Darcy's law, extending the equation to be a function of time

[2]. In this modified Darcy's law, there are two empirical model parameters, α and σ, where $\alpha\xi$ is the equivalent media thickness of the cake, and σ the variation in particle size distribution and location of retained particles which is usually approximately equal to 1. We assume all other governing equations remain valid.

Modified Darcy's Law:

$$\begin{aligned} P(x,t) - P_\infty &= \frac{\varphi(x,t)\mu H}{\Gamma}\left(\frac{1+\frac{\alpha}{H}\left(\frac{K\Theta(t)}{1+\Theta(t)}\right)}{1-\frac{\sigma}{\rho_d}\left(\frac{K_d}{1+\Theta(t)}\right)}\right) \\ &= \frac{\varphi(x,t)\mu H}{\Gamma}\left(\frac{1+\frac{\alpha}{H}\xi(t)}{1-\frac{\sigma}{\rho_d}C_d(t)}\right) \\ &= \frac{\varphi(x,t)\mu H}{\Gamma}\beta(t) \end{aligned} \tag{2.28}$$

Where the equation governing $\varphi(x,t)$ is given as:

$$\beta(t)\frac{d^2\varphi(x,t)}{dx^2} - \frac{1}{\lambda_c^2}\varphi(x,t) = 0 \tag{2.29}$$

Notice that the limiting cases here are given when $\Theta(t) \to \infty$ which is when fouling is in cake formation only, and $\Theta(t) = 0$, when fouling is in depth plugging only. Now our governing equations are given by the Hagen-Poiseuille equation, modified Darcy equation, and mass conservation. The analytical solutions are straightforward extensions of the solutions that were derived for the clean filter case. The filter design parameters for ΔP_{ref}, reference pressure drop, Q_{ref} reference flow rate, and λ_c characteristic length are the same as in the clean filter. However, the maximum filter length $L_\infty = \sqrt{\beta(t)L_{\infty\text{ clean filter}}}$. We must have that $\beta(t) \geq 1 \quad \beta(0) = 1$ such that $L_{\infty,\text{ clean filter}}$ remains a reasonable maximum filter length for the transient case.

Analytical Solutions for Transient Case with Fouling of the Rectangular Domain

$$Q(x) = -Q_{\text{ref}}\sinh\frac{x}{\lambda_c\sqrt{\beta(t)}} + Q_0\cosh\frac{x}{\lambda_c\sqrt{\beta(t)}} \tag{2.30}$$

$$P(x) = P_\infty + (P_0 - P_\infty)\sqrt{\beta(t)}\cosh\frac{x}{\lambda_c\sqrt{\beta(t)}} - \Delta P_{\text{ref}}\sqrt{\beta(t)}\sinh\frac{x}{\lambda_c\sqrt{\beta(t)}} \tag{2.31}$$

$$\varphi(x) = \frac{Q_{\text{ref}}}{n_w W\lambda_c\sqrt{\beta(t)}}\cosh\frac{x}{\lambda_c\sqrt{\beta(t)}} - \frac{Q_0}{n_w W\lambda_c\sqrt{\beta(t)}}\sinh\frac{x}{\lambda_c\sqrt{\beta(t)}} \tag{2.32}$$

$$Q(0,t) = Q_0 \quad P(0,t) = P_\infty + (P_0 - P_\infty)\sqrt{\beta(t)}, \quad t > 0 \tag{2.33}$$

Results are discussed further in Chapter 4.

Chapter 3

Development of a Web Based Calculator

This chapter will walk through the steps necessary to setup, implement, and run a simple calculator application using the Django framework for web applications. The application created will take user specified parameters for period and amplitude to plot the appropriate sine curve. This chapter assumes basic knowledge of the command line and the programming language Python. Note the Windows commands will be different from the Mac OS terminal commands which are used here.

3.1 Setup of Django Framework for Web Applications

1. **Django Background**

   Django works by providing the basic code needed to communicate between your computer or server (the back end), and the browser (the front end) [12]. The benefit of using Django is that computations are still done quickly and efficiently on the back end, where there is the most power, while the user interfacing and plotting are done on the front end, giving all the design features and interaction offered by HTML, JavaScript, and CSS. The major benefit lies in the ability to allow others to interact with your calculations, without needing any of the source code.

   The Client-Server Relationship

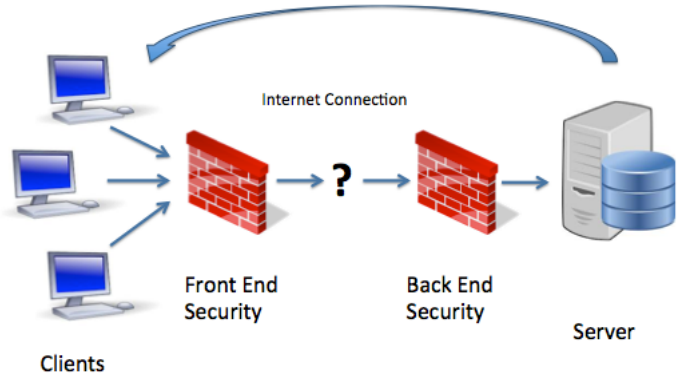

Figure 3.1: Client server data flow chart.

It is important when implementing a web application to understand the way a web app is organized, and what function each piece servers. The "client" or "front end" is what the user will see and interact with. The underlying code is run in an internet browser, and is accessible from any computer via an internet connection. The "server" or "back end" is what will be hosting the application. This is where the underlying code for the computation is run. It can run locally on a personal machine, or be hosted remotely on a server. Anything on the back end is completely hidden from the user.

The Client and Server each utilize different programming languages for different purposes. Languages used server side are typically robust and powerful, whereas client side languages are tailored to design, interface, and style/layout. Some examples of Client languages would be HTML, CSS, and Javascript, all of which are used in this web application. Common server side languages include but are not limited to Python, C/C++, Perl, Ruby, Java, and PHP.

Figure 1 represents the organization of the application. A user interacts with the front end of the application and sends some information (such as model parameters) to the back end. Before the back end will accept these parameters however they have to be screened for malicious content. Once the input has passed security, the input is accepted by the server which then uses the input to produce a result. This result is then sent back to the front end and displayed to the user. There is no security in this direction, because the researcher will not be sending any malicious content to the front end.

The question mark in this graphic is where Django comes in to play. “Django is a high-level Python Web framework that encourages rapid development and clean, pragmatic design. By taking care of much of the hassle of Web development, you can focus on writing your app without needing to reinvent the wheel.” [12] For this application Django is used in place of AJAX/JSON to ”send and receive data from a server without interfering with the display or behavior of the webpage.” [12] Essentially, Django facilitates the passing of information between the client and server.

3.2 Django Organization

For a simple application you will only need to use the following Django files:

- URLS - control the navigation of the web app
- Views - a python function that takes a request and issues a response
- Templates - HTML files which tell each URL how and what to display on a given page. Gives the “structure” of the page.
- Settings - contains information such as dynamic root and staticfile root
- StaticFiles - the directory where image, CSS, and Javascript files are stored

2. **Installing Anaconda** To use Django a package manager and an environment manager are highly recommended. Other package and environment managers will work, but for this project I used Anaconda [1]. To install Anaconda, open the following link in your browser and follow the appropriate set of instructions for your operating system.

   `https://www.continuum.io/downloads`

3. **Creating a New Environment**

   For Django projects, it is a good idea to set up an environment. You can think of an environment as a completely isolated container within your computer, where you can utilize any version of Python and install libraries/packages and without affecting anything outside that container [15]. This will ensure that the correct packages and their versions are accessible for your program, and that any updates preformed at the system level will not break your code.

Within our environment, we will use Django (web application framework), NumPy (math tools package), and will include SQLite (required underlying database, not used in this application). To create the new environment type the following command in your shell. Note that envName is the name of the environment, and is specified by the user.

```
conda create --name envName django sqlite numpy
```

Now activate your new environment:

```
source activate envName
```

Note that when you are ready to deactivate your environment, it can be done with the command

```
source deactivate
```

3.3 Creating a New Django Project

This will be the first step actually using Django's framework. If you do not want your project in your home directory, you will need to navigate to the directory that you would like for your project to live in. Once in your desired directory, enter the following command to create your new project. We will call this project sin_site.

```
django-admin startproject sin_site
```

You should now have the following files and directories in your base directory as seen in Figure 3.2.

```
.
├── [ 251]  manage.py
└── [ 204]  sin_site
    ├── [   0]  __init__.py
    ├── [2.6K]  settings.py
    ├── [ 758]  urls.py
    └── [ 393]  wsgi.py

1 directory, 5 files
```

Figure 3.2: Original file organization structure

Now we can verify that the project setup worked by starting the server. This command will need to be run each time you would like to begin working on your project as it allows you to see your project in the browser.

```
python manage.py runserver
```

If the command was entered correctly you should see something similar to the output below.

```
December 21, 2015 - 18:14:34
Django version 1.8.4, using settings 'sin_site.settings'
Starting development server at http://127.0.0.1:8000/
Quit the server with CONTROL-C.
[21/Dec/2015 18:15:07] "GET / HTTP/1.1" 200 1767
```

Now open your browser (Internet Explorer not recommended) and type the url given in the output. You should see Figure 3.3 displayed in the browser.

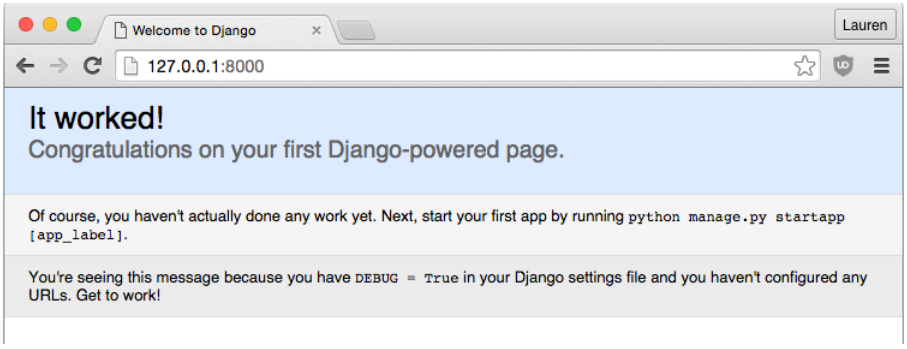

Figure 3.3: Display of successful Django setup in browser.

Now we need to create our application. The application lives within the project we just created. Projects can have more than one application associated with them, however for simple applications like this it is not necessary to have more than one. To create your application enter the command with your desired application name, we will use 'calculator'.

```
python manage.py startapp calculator
```

Your directory should now be organized as in Figure 3.4

```
.
├── [ 272]  calculator
│   ├── [   0]  __init__.py
│   ├── [  63]  admin.py
│   ├── [ 102]  migrations
│   │   └── [   0]  __init__.py
│   ├── [  57]  models.py
│   ├── [  60]  tests.py
│   └── [  63]  views.py
├── [3.0K]  db.sqlite3
├── [ 251]  manage.py
└── [ 340]  sin_site
    ├── [   0]  __init__.py
    ├── [ 138]  __init__.pyc
    ├── [2.6K]  settings.py
    ├── [2.8K]  settings.pyc
    ├── [ 758]  urls.py
    ├── [ 987]  urls.pyc
    ├── [ 393]  wsgi.py
    └── [ 597]  wsgi.pyc

3 directories, 16 files
```

Figure 3.4: New file organization structure after creating the application.

3.4 Adjusting the Settings

From your base directory, change into the directory that was automatically created by Django with the same name. You should be in the directory:

```
home/.../sin_site/sin_site
```

Note that the exact file path will be OS dependent. In a text editor, open the file settings.py. This is where we will change the appropriate settings for our project, and add some features that will allow the use of Javascript and CSS.

I. **Change** the 'DIRS' [ ]: setting, under the TEMPLATES class so that it contains the following:

```
TEMPLATES = [
    {
        'BACKEND': 'django.template.backends.django.DjangoTemplates',
        'DIRS': [os.path.join(BASE_DIR, 'templates')],
        'APP_DIRS': True,
```

```
]
```

II. **Change** the TIME_ZONE = 'UTC' setting to the appropriate time zone for your location [13].

III. **Add** the following code at the end of the file. This code segment allows us to use JavaScript and CSS in our front-end, and tells Django where to find the files that we will use to do so.

```
MEDIA_ROOT = os.path.join(BASE_DIR, 'media')

MEDIA_URL = '/media/'

STATIC_ROOT = os.path.join(BASE_DIR, 'static')
STATIC_URL = '/static/'

STATICFILES_DIRS = (
    os.path.join(BASE_DIR, 'sin_site/staticfiles'),
    )
```

3.5 Creating the Views

The views file in your project is what will be run on the back end for each view that you need. For our purposes we will need two views; one that tells us to get the sine waves, amplitude and period from the user, and a second view that takes those values, calculates the appropriate points, and returns an array of x and y values. Open your views.py file in the directory sin_site/calculator/. First, we need to import two python packages, NumPy and math. Add the statements to your file.

```
import numpy as np
import math
```

Next, we will define the first view that we need. This view tells Django that we need to post the form that will get the inputs from the user. We will create this form and the url associated with it later. All views will end by returning some render function. Render tells Django which

HTML file to display, and passes it any values that it will need. To define the first view, all we need to do is tell Django which HTML file to display. Create the view defined below.

```
def post_form_upload(request):
  return render(request, 'admin/post_form_upload.html')
```

The second view is where we will preform any calculations. This view is called after the first view we defined, which renders an HTML file that will get user input. We need to get this input to be able to use it in our view. To do so we will use the function request.GET.

```
def plot(request):
     if 'amplitude' in request.GET:
          if 'period' in request.GET:
               amplitude = float(request.GET['amplitude'])
               period = float(request.GET['period'])
```

Now that the view has the necessary parameters, we can use them to get the appropriate values for x and y, which we will store in vectors. In this application we will be plotting from 0 to 2π at 1000 points. To do this we just need a few additional lines added into the if statements.

```
def plot(request):
     if 'amplitude' in request.GET:
     if 'period' in request.GET:
          amplitude = float(request.GET['amplitude'])
          period = float(request.GET['period'])
          arrayx = np.linspace(0, np.pi*2, 1000)
          arrayy = np.sin(period*arrayx)*amplitude
          arrayx = np.array_str(arrayx)
          arrayy = np.array_str(arrayy)
```

To complete our views we need to return our render function, with the arrays that we calculated. We pass calculated values inside { } separated by commas. The name of the value will be placed in single quotes.

```
def plot(request):
     if 'amplitude' in request.GET:
     if 'period' in request.GET:
          amplitude = float(request.GET['amplitude'])
          period = float(request.GET['period'])
```

```
        arrayx = np.linspace(0, np.pi*2, 1000)
        arrayy = np.sin(period*arrayx)*amplitude
        arrayx = np.array_str(arrayx)
        arrayy = np.array_str(arrayy)
    return render(request, 'admin/plot.html', {'arrayy': arrayy,
        'arrayx': arrayx} )
```

3.6 Defining the URLs

The URL entered in the browser tells Django what we would like to see. Each URL is associated with a view and an HTML file. We need a URL for each view that we created. In the directory sin_site/calculator create a new file called urls.py. At the beginning import the packages,

```
from django.conf.urls import url, patterns, include
from django.conf.urls.static import static
from django.contrib import admin
from django.contrib.staticfiles.urls import staticfiles_urlpatterns
from sinsite import settings

from . import views
```

These packages mainly allow us to use Javascript and CSS. Next define your URLs as

```
urlpatterns = patterns('',
    url(r'^post/form_upload/$', 'calculator.views.post_form_upload',
        name='post_form_upload'),
    url(r'^post/plot/$', 'calculator.views.plot', name='plot'),
)
```

Lastly we need to add the code that will tell Django that we intend to use static and/or media files for JavaScript and CSS. Do this by adding,

```
urlpatterns = patterns('',
    url(r'^post/form_upload/$', 'calculator.views.post_form_upload',
        name='post_form_upload'),
    url(r'^post/plot/$', 'calculator.views.plot', name='plot'),
) + static(settings.MEDIA_URL, document_root=settings.MEDIA_ROOT)
```

```
urlpatterns += staticfiles_urlpatterns()
```

3.7 Organizing the Front-End

The front-end of our web app will need 3 different kinds of files, stored in two locations. The languages, with their file extension and purpose are listed below.

a. HTML (.html) - A markup language we will use to create our webpage.

b. JavaScript (.js) - A high-level dynamic programming language that we will use for plotting, and will allow for interaction with the user.

c. CSS (.css) - A style sheet language, that is used for describing the presentation of a webpage. Contributes mainly to the visual aspects of the page (font size, colors, fill colors, etc.).

First we need to create the directories where these files will live. Make sure you are in your base directory at this step (sin_site/). We need a directory called templates, and a subdirectory, admin. These directories will house our HTML files. Create them by typing the commands.

```
mkdir templates
cd templates
mkdir admin
cd admin
```

Now any HTML files that we need will be placed in the path sin_site/templates/admin/ . Our JavaScript and CSS files, however, will need a different path. Change back to the base directory (try typing cd ../..). For this step, we need to enter into the directory sin_site, create a new directory called staticfiles, then create two more directories underneath, one for .js files and one for .css.

```
cd ../..        <--(Should take you to base directory sin_site/)
cd sin_site
mkdir staticfiles
cd staticfiles
```

```
mkdir js
```

```
mkdir css
```

Once this step is completed, your files should be organized the same way as in the tree in Figure 3.5.

```
.
├── [ 306]  calculator
│   ├── [   0]  __init__.py
│   ├── [  63]  admin.py
│   ├── [ 102]  migrations
│   │   └── [   0]  __init__.py
│   ├── [  57]  models.py
│   ├── [  60]  tests.py
│   ├── [ 593]  urls.py
│   └── [ 561]  views.py
├── [3.0K]  db.sqlite3
├── [ 251]  manage.py
├── [ 374]  sin_site
│   ├── [   0]  __init__.py
│   ├── [ 138]  __init__.pyc
│   ├── [2.8K]  settings.py
│   ├── [3.0K]  settings.pyc
│   ├── [ 136]  staticfiles
│   │   ├── [  68]  css
│   │   └── [  68]  js
│   ├── [ 758]  urls.py
│   ├── [ 987]  urls.pyc
│   ├── [ 393]  wsgi.py
│   └── [ 597]  wsgi.pyc
└── [ 102]  templates
    └── [  68]  admin

8 directories, 17 files
```

Figure 3.5: New file organization structure after adding staticfiles sub directories

3.8 The HTML Form

In order to get parameters from the user, an HTML form is required. In the directory

```
templates/admin/
```

create a new file called

```
post_form_upload.html
```

Now we will create a simple HTML form. Note that we are no longer using the Python programming language. Enter the code below and save the file.

```
<html>
<head>
<title>Sine Plotting</title>
<head/>
<body>
Enter an Amplitude and Period for your Sine Wave
<form action="/sin/post/plot/" method="get">
<br/>Amplitude:
<input type="number" name="amplitude"><br/>
Period:
<input type="number" name="period"><br/><br/>
<input type="submit" value="Submit">
<form/>
</body>
</html>
```

Here the title is just a name for your webpage that will not be displayed anywhere. Within the form tags three input types are defined. Notice that at the first from tag we define that the action taken when the user submits the request is that the URL will change from the current

```
/sin/post/form_upload/
```

to

```
/sin/post/plot/
```

which we will define shortly.

There are two input statements that are being used to accept a user defined input, in this case a number, one for the amplitude and one for the period. The names used for this input should be the same used in the GET statements in the views.py folder defined before. The third input is defining a "submit" button.

3.9 Creating the Plot

To produce the graph for our results from the plot function in the views.py folder, I have used plotly [21]. Plotly is available for a variety of different languages and produces quality, interactive graphs. By using Javascript for the graph, we allow the user to have this interaction within the web application. They can hover over points to get their values, filter, zoom, download, etc. This is possible because Javascript is being used on the front end, if the plot was produced server side and sent to the client, it would be completely static, like looking at a jpg. To download plotly for javascript go to the following link and follow the instructions.

`https://plot.ly/javascript/`

Once the package is installed navigate to the directory

`staticfiles/js/`

and create a new file called index.js

First the user specified parameters need to be pulled into the Javascript using the

`getElementById() function`

and passing the same name that was returned in the plot function from views.py. In this case that will be “arrayx” and “arrayy”.

```
var arrayx = document.getElementById("arrayx").value;
var arrayy = document.getElementById("arrayy").value;
```

Next the data needs to be parsed into a usable format. For this call the Javascript function

`split()`

this function takes a regular expression as an input. The one used here will pull out all the individual numbers.

```
var newx = arrayx.split(/ {1,}/);
var newy = arrayy.split(/ {1,}/);
```

We store the graphing information in a Javascript trace data structure, then call plotly’s

`newPlot( )`

function to render the graph. Enter the remaining code below and save the file.

```
var trace = {
  x: newx,
  y: newy,
  mode: 'markers'
};

var data = [trace];
Plotly.newPlot('sinplot', data);
```

3.10 Displaying the Result with HTML

The last thing that needs to be done to have a functioning web application is to create an HTML file that will display the graph we created in Javascript to the user. Navigate back to the

`/templates/admin`

subdirectory and create a new file called

`plot.html`

In the header, we will once again defined a title, and this time a line needs to be included to tell the HTML file that you are going to be calling Javascript code, and where to look for this static file. If we were styling the page using CSS, we would declare an additional statement of the same format. The

`<script>`

tag in HTML tells the file what static files you will be using, type is either

`"text/javascript"`

or

```
 "text/css"
```

depending on the language being used. *Src* tells the file where to look for the code that is being called. Add the following header,

```
<html>
<head>
<title>Plot</title>
{% load static %}
<script type="text/javascript" src="{% static 'js/plotly-latest.min.js' %}"></script>
</head>
```

Next, in the body we need to load the x and y data from the views. We access these values by creating hidden values, hidden simply meaning that while they are contained in the webpage, they are not shown to the user. Access these values using the bracket notation as follows {{ variable }} . A container also needs to be specified for the graph we are containing, we specify the container using the *div* tag, assign the container and id, and specify the size of the image we want to display.

```
<body>
<input type="hidden" id="arrayx" name="arrayx" value="{{ arrayx }}">
<input type="hidden" id="arrayy" name="arrayy" value="{{ arrayy }}">

<div id="sinplot" style="width:600px;height:600px;"></div>
```

Finally, the javascript file that plots the curve is called using the script *tag*.

```
<script type="text/javascript" src="{% static 'js/index.js' %}"></script>

</body>
</html>
```

3.11 Starting the Server

Now that all of the files are written, we go back to the command line to initialize the server. Make sure you are in the /sinsite/ directory, and enter the following command to run the application.

```
python manage.py runserver
```

If the application was successfully launched, you will see something similar to the following in your terminal.

```
Performing system checks...

System check identified no issues (0 silenced).
October 09, 2017 - 19:06:50
Django version 1.8.4, using settings 'sinsite.settings'
Starting development server at http://127.0.0.1:8000/
Quit the server with CONTROL-C.
```

Opening the browser and navigating to the url designated for our form, $http://127.0.0.1:8000/sin/post/form\_upload/$ should populate your browser with the inputs for Amplitude and Period. Select values for these parameters and click 'Submit'. The url will automatically change and the graph will populate on the page as shown in Figure 3.11.

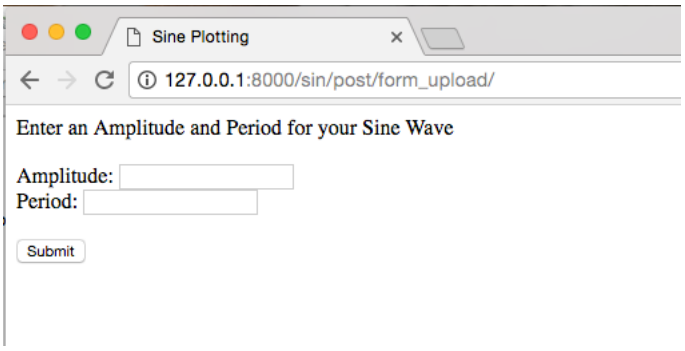

Figure 3.6: Form for parameter entry displayed in browser.

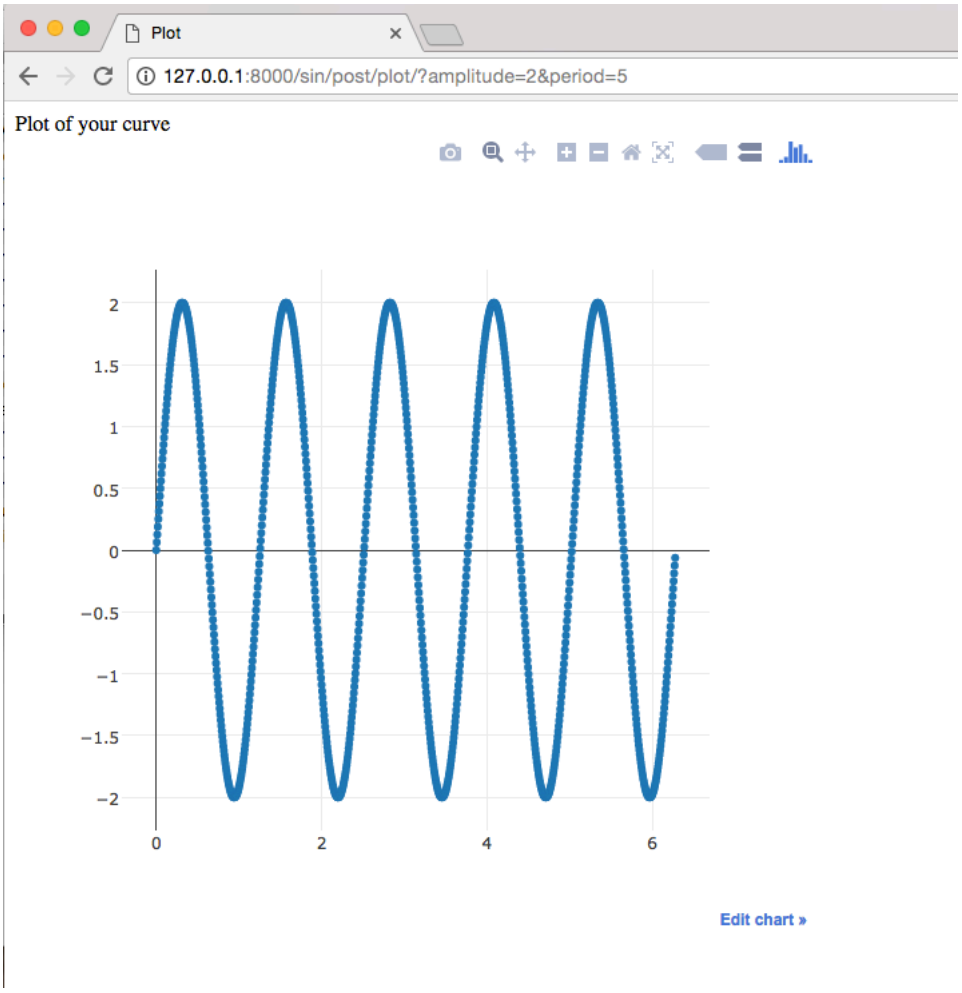

Figure 3.7: Graph for the given parameters.

3.12 Security

One of the only draw backs of a web based application is that if the application is run on a server and made accessible to the public, the server becomes susceptible. Any time user-specified input is taken and stored or used by the server, the potential for malicious input has to be taken into consideration. While there are many ways a server can be attacked, for a simple web application where only numerical input is taken there are only two main concerns,

- Injection - an attacker modified a back-end statement of command through unsanitized user input
- Cross-Site Scripting - an attacker inserts Javascript in the pages of a trusted site, allowing them to alter the contents of the site

To counter these threats security measures should be taken on both the front-end and back-end. These changes are implemented and can be seen in the filtration calculator source code, which is included in the appendix. The first two gates inputs will have to pass through are on the client side. In the HTML form, inputs should be specified with *type = "number"* (not effective for some versions of Internet Explorer). This will reject any input that doesn't fit the requirements of a number, only one decimal, no special characters etc. The next client side check occurs in the Javascript code, which again asserts the input is a number, and also requires that the input fall within a certain, reasonable range. By specifying that inputs must be numerical and fall within a maximum and minimum value, we eliminate the possibility that the input has long strings of characters and symbols which are actually commands that could harm the server if they were executed. The last security procedure occurs on the server itself in Python. The Python code only executes if every input is valid, and before accepting the input type casts them as numbers.

While these security measures should suffice for a simple application, consulting with an IT security professional is always recommended. It is very strongly discouraged that any form of input be accepted that is not numerical, including uploading files, specifying equations, etc. While the Django framework certainly supports this, much more extensive countermeasures will need to be taken to ensure the safety of the server.

Chapter 4

A Web Application for Simulating Cross-Flow Filtration

One of the main goals of research is to share results and insights found with others so that progress can continue to be made by others in the field. This process is fairly straightforward when it comes to most of mathematics, the results are written into a paper which is then published. In the field of computational science however, sharing results becomes difficult because of the often extensive use of programming to generate numerical results. This poses a several unique challenges when it comes to sharing research.

One difficulty is the compatibility across operating systems and package/language versions, and user experience level. It is often challenging for even experienced software developers to ensure their code will run once it is on a different computer, even more so for researchers whose backgrounds are not in computer science. Even if the program will run with no compatibility issues, setting up the appropriate environment for the program can be complicated and overwhelming. While some integrated development environments like Matlab are familiar and easy to use, they are often not suited for large scale computing, so programs are being written in languages like C++ or Python which, while very powerful, require a stronger programming background to use.

Even if you are sharing your work with a very experienced developer, another issue arises in that if you distribute your code, you are not able to withhold any proprietary information about your model, how you solved the problem, or your approach to the numerical approximation. This

can be undesirable for several reasons, for example if your research was sponsored by a company who wants to use the model competitively.

One practical solution to these problems is to create a web application. Web applications are desirable in that they eliminate the need to distribute, set up, and run any code, while still allowing others to interact with your model and see the results that you have generated. The user interacts with the model entirely through a web browser (Chrome, Firefox, etc), by navigating to a specified URL, and using the page as you would with any website. The code for the model runs entirely hidden from the user, so they may be allowed to pick parameters and see the output of those choices, without ever actually seeing the code that is generating those results. Some of the advantages and disadvantages of a web application are listed below.

Advantages

- Does not require users to have access to, modify, or run source code
- Widely and easily accessible
- No installation necessary
- No programming experience required
- No compatibility issues outside of web browser
- Capable of easy, straightforward user interface
- Can run locally for those with full access to source code (on personal computer), or remotely (on a server, accessed by URL)

Disadvantages

- Requires additional work to set up
- Requires an internet connection (remote)
- Security concerns when accepting user input (remote)

To create the filtration web application required approximately 2,000 lines of code across 4 different programming languages.

4.1 Domain Selection and Parameter Input

The first screen a user sees when they navigate to the web application's URL is the domain selection, which allows the user to choose either the rectangular or cylindrical domain by clicking on the corresponding picture of the geometries. Upon hover, the picture is highlighted, and clicking navigates automatically to the next page. This can be seen in Figure 4.1.

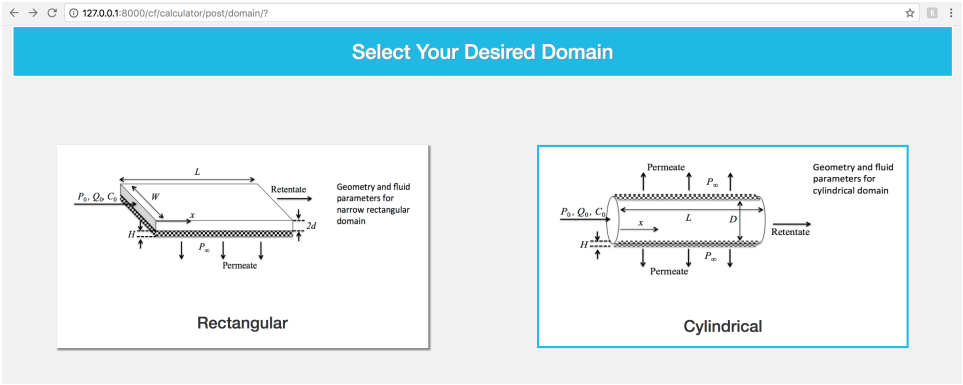

Figure 4.1: Domain selection

Now that the domain is selected, the application allows the user to specify input parameters for the model. These are broken down by category as seen in the following Figures, geometry parameters 4.2, flow parameters, 4.3, filter parameters 4.4, and debris parameters 4.5. Each input box is automatically populated with default input values, and upon hovering over the box, arrows in the right hand side of the input box allow the user to incrementally adjust their values. If a user tries to enter an invalid input, like letters, an alert is raised and the input is not accepted. The web page also includes a picture of the filter geometry selected, to aid a user in visualizing some of the parameters, this can be seen in Figure 4.6. Once the parameters are adjusted to the users liking, hitting Plot moves to the next page.

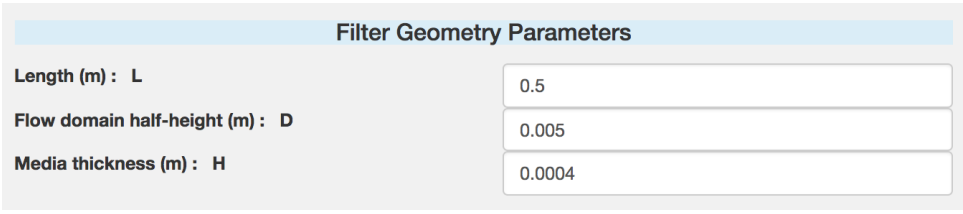

Figure 4.2: User defined geometry parameters

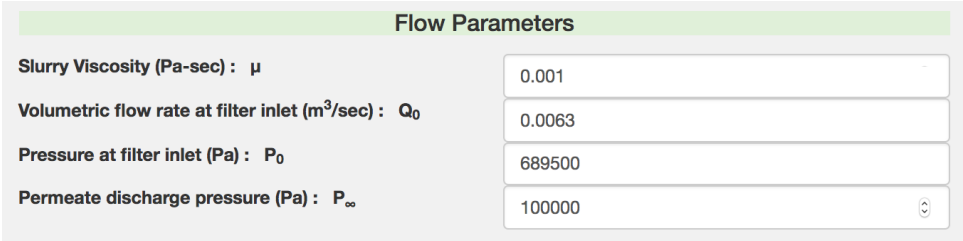

Figure 4.3: User defined flow parameters

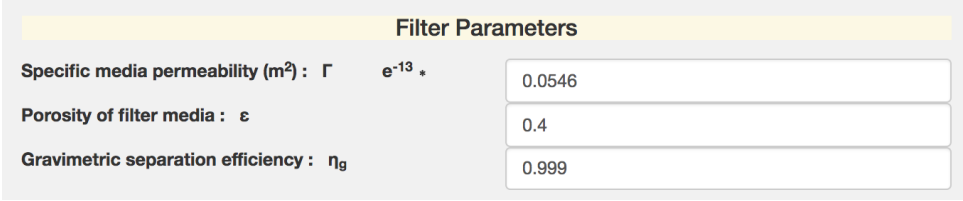

Figure 4.4: User defined filter parameters

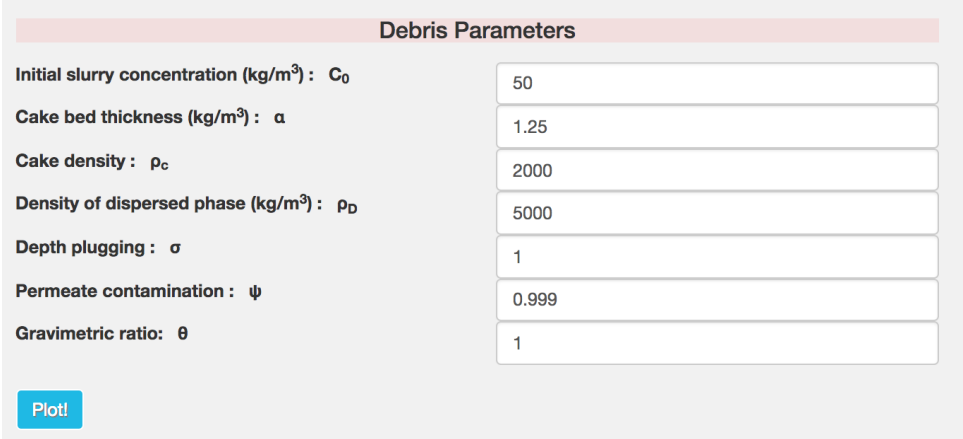

Figure 4.5: User defined debris parameters

Using this application, we can plot the permeate flux $\varphi(x)$ and volumetric flow $Q(x)$ with respect to position along the length of the tube, and time with $t = 0, 10, 20, 30$ for the results of the analytical solutions provided in equations (2.30), (2.31), (2.32). The solutions for the Rectangular domain are seen for permeate flux and volumetric flow in Figures 4.8 and 4.9 respectively, with parameters found in Figure 4.7. Similarly the results for the Cylindrical domain are shown in Figures 4.11 and 4.12 with parameters found in Figure 4.10.

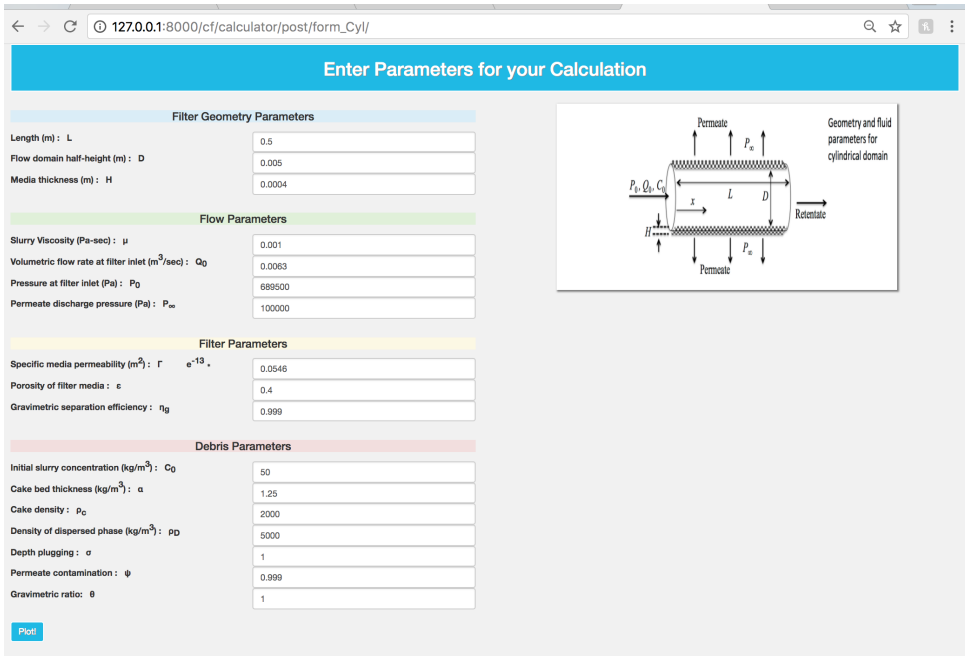

Figure 4.6: Entire parameter input page

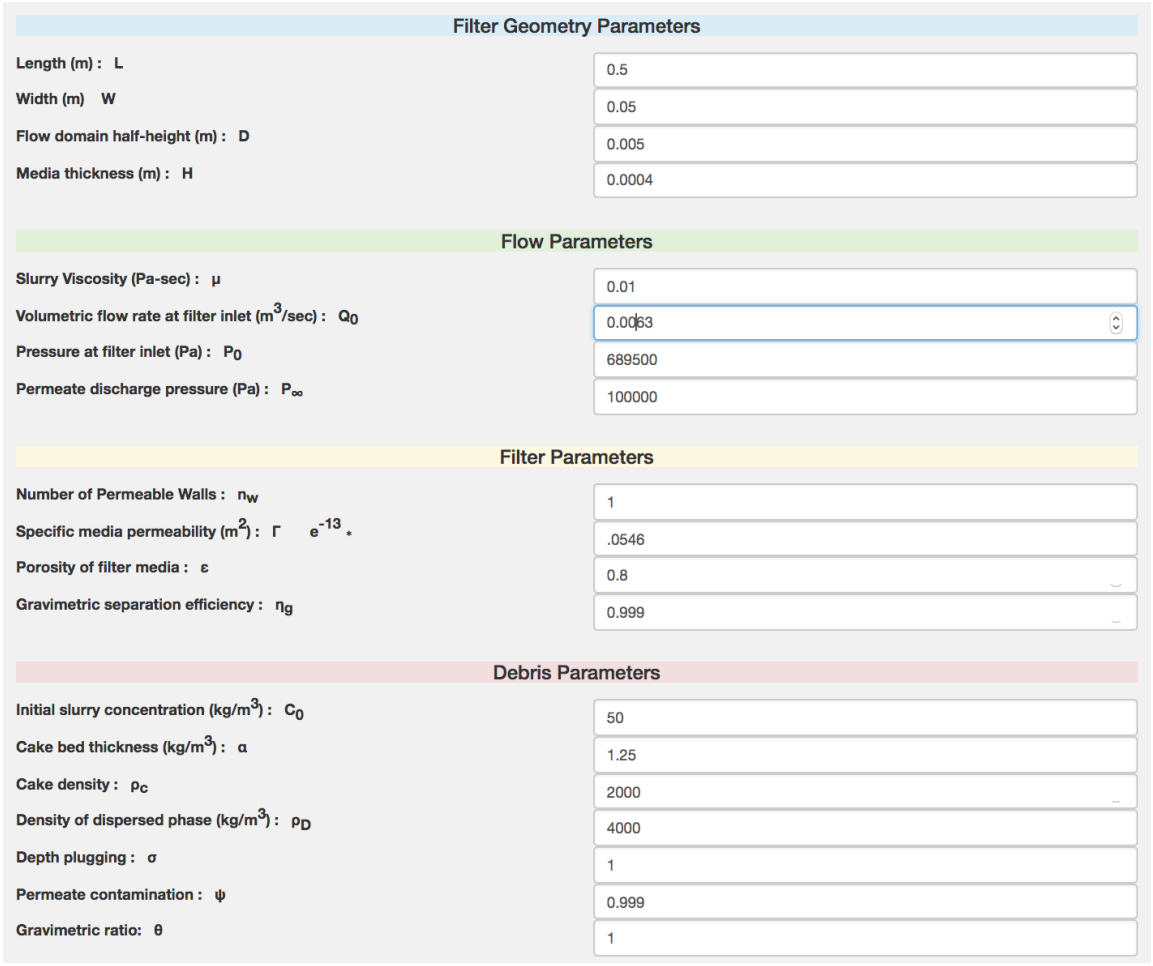

Figure 4.7: Parameters for the rectangular domain as specified in web app

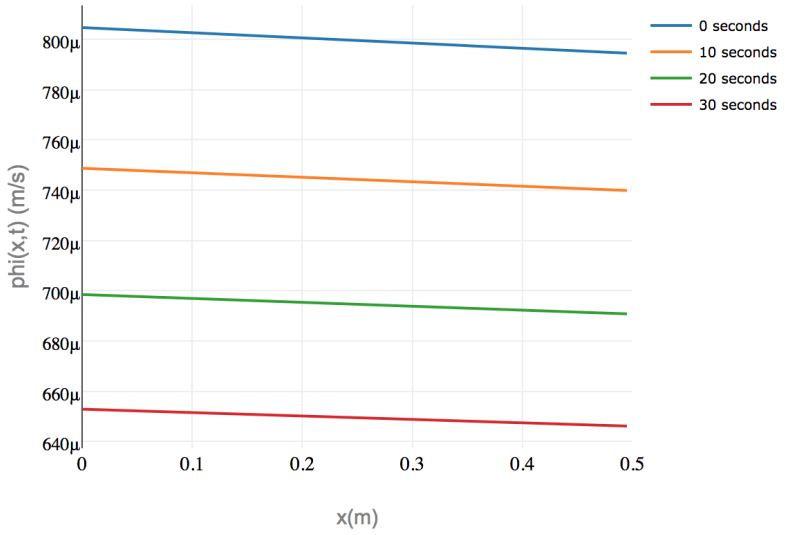

Figure 4.8: Rectangular permeate flux varying with respect to position and time

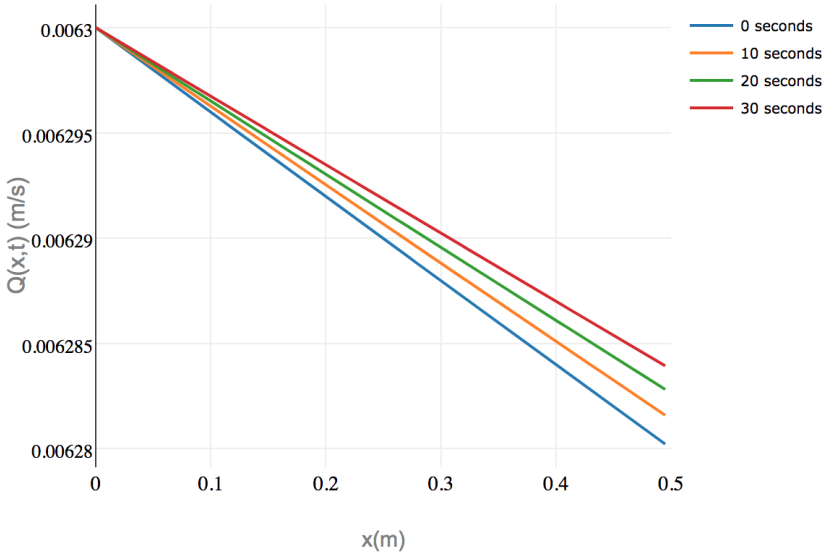

Figure 4.9: Rectangular slurry volumetric flow rate varying with respect to position and time

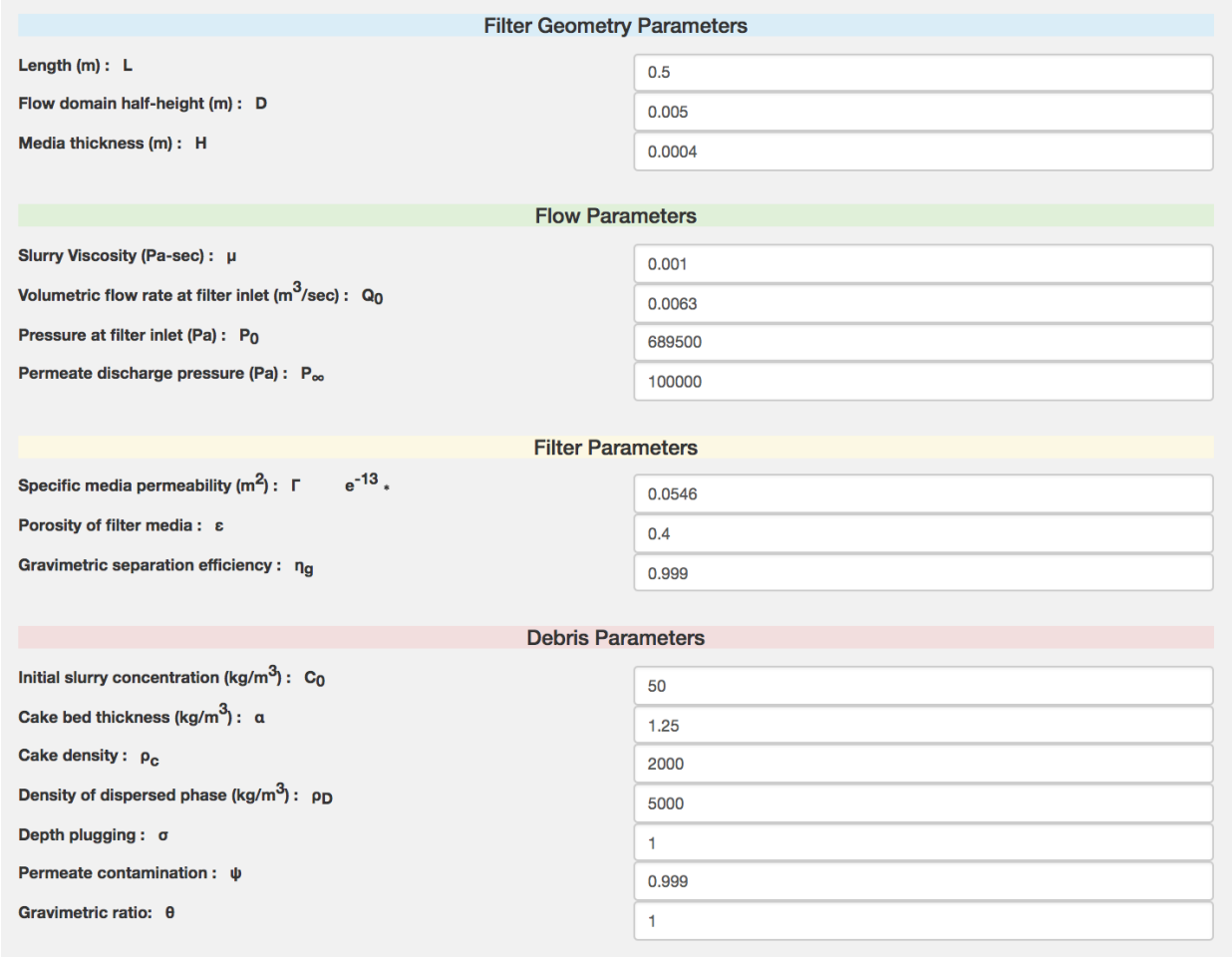

Figure 4.10: Parameters for the cylindrical domain

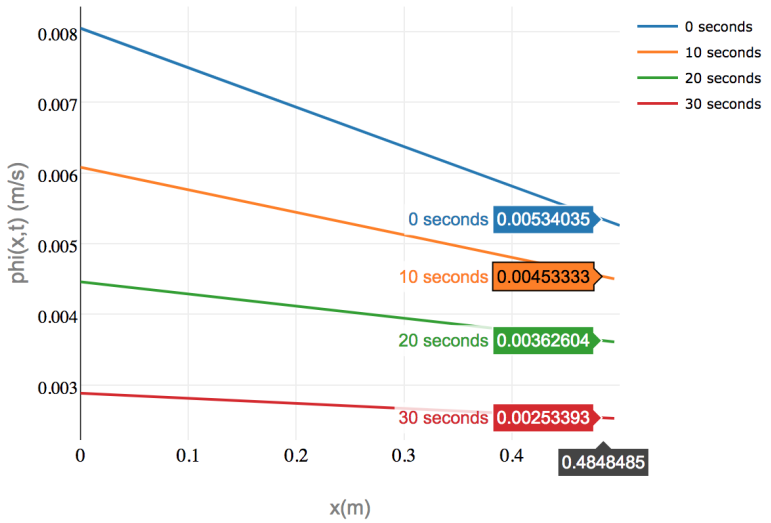

Figure 4.11: Cylindrical permeate flux varying with respect to position and time

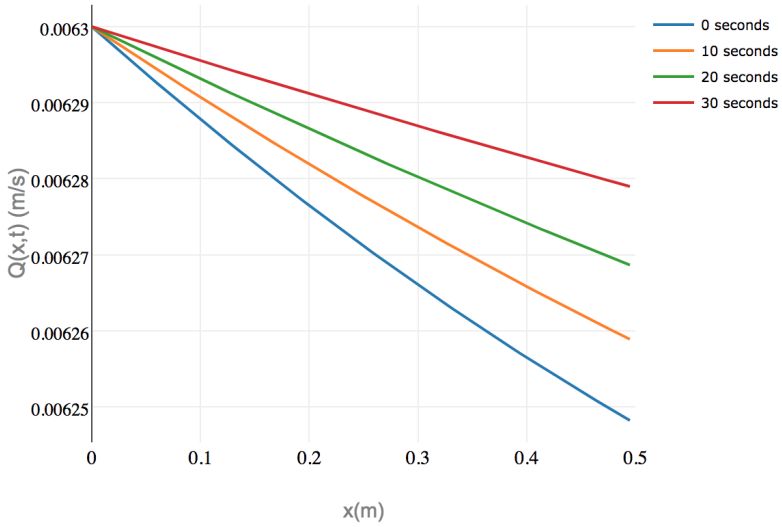

Figure 4.12: Clyindrical slurry volumetric flow rate varying with respect to position and time

4.2 Plotting the Inputs

Now that the user has defined parameters for the model, this information is passed to the server, where the computation itself is done, and the results passed back to the front end where plotting takes place using a package called Plotly in Javascript [21]. While it is a viable option to produce the plot in a package on the server side in Python, using a package like matplotlib, and send the image to the front end to be rendered, there is a major advantage to producing this graph on the front-end. When we allow Javascript to render the plot, the graph itself can be interacted with, manipulated, and the data explored and downloaded by the user. Javascript, by nature, interacts with browsers dynamically, so that instead of the user looking at a picture, they can have an interactive experience. You can see one of these features in Figure 4.13 where by hovering over one of the graphs, values for the x-axis and y-axis are displayed.

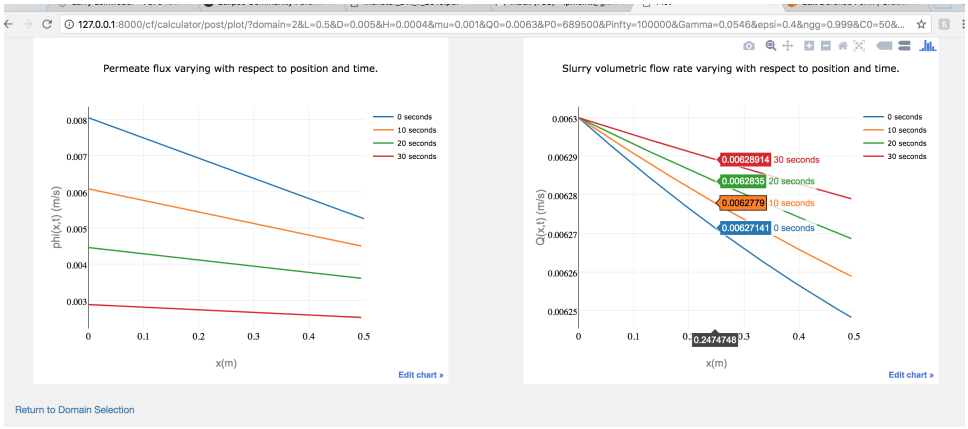

Figure 4.13: Simulated graphs

While the developer can see the underlying HTML, Javascript, and CSS that renders this web page, the code that does the computation for the model is entirely hidden, meaning the remote user can see results, but does not automatically have access to the code producing them. In an application produced in this framework, any code written in python will be hidden from the user.

Chapter 5

Numerical Model

To develop a more realistic model for cross-flow filtration, there are assumptions that needs to be eliminated. With these assumptions lifted there are no longer analytical solutions to the system of governing equations, so numerical methods are used to approximate solutions. We begin by applying finite elements to the one-dimensional clean filter case for bench marking, then assume that viscosity, μ is no longer constant. The governing equations for the case of the cylindrical domain will be used for the numerical solution.

5.1 Mathematical Model for the case of Position Dependent Viscosity

We consider the tubular filter with no fouling effects and in steady state. We assume the flow is 1-D laminar with permeate velocity $\ll$ cross flow velocity, $D \ll L$ so traverse changes in flow and pressure are negligible. The governing equations on $0 \leq x \leq L$ are a variation on equations (2.1), (2.2), (2.3), with boundary conditions:

$$P(0) = P_0 \qquad Q(0) = Q_0 \tag{5.1}$$

A natural phenomenon that occurs during filtration is that as permeate penetrates through the filter media, there is a higher concentration of dispersed phase in the slurry, changing the slurry's viscosity along the length of the filter. To model this phenomenon, we will assume that viscosity changes

with respect to x linearly and hence we can express an equation for viscosity as,

$$\mu(x) = \mu_0 + a_\mu \frac{x}{L} \tag{5.2}$$

with μ_0 the initial viscosity, and $\frac{a_\mu}{L}$ the gradient. Under these assumptions, should $a_\mu = 0$ the viscosity reduces to a constant and therefore will have an analytical solution that can be used for bench-marking. Before introducing the finite element approach used, we formulate the problem in terms of pressure, as is commonly used in practice, e.g. [25]. To formulate the problem in terms of a single, dependent variable, pressure, eliminate $\varphi(x)$ using equations (2.1) and (2.3):

$$Q'(x) + \frac{\pi D \Gamma}{\mu(x) H}(P(x) - P_\infty) = 0 \tag{5.3}$$

Now taking the derivative of equation (2.2):

$$P''(x) + \frac{128}{\pi D^4}(Q(x)\mu'(x) + Q'(x)\mu(x)) = 0 \tag{5.4}$$

Assuming (5.4) and (5.3) we can write:

$$P''(x) - \frac{128\Gamma}{HD^3}(P(x) - P_\infty) - \frac{\mu'(x)P'(x)}{\mu(x)} = 0 \tag{5.5}$$

Notice that by specifying $Q(0)$, we are also specifying $P'(0)$, since

$$\frac{dP}{dx}(0) = -\frac{128\mu}{\pi D^4} Q(0) \tag{5.6}$$

Therefore the inflow conditions for this problem will be on P and P'.

5.2 Least-Squares Finite Element Formulation

We use the least squares finite element method because of the ease with which inflow boundary conditions can be imposed. See [18] for details about the least squares finite element method.

Consider the general form of equation (5.6), with inflow boundary conditions:

$$p'' + cp' + dp = f \qquad \text{on} \quad a \leq x \leq b \tag{5.7}$$

$$p(a) = p_{0,0} \qquad p'(a) = p_{0,1} \tag{5.8}$$

where c and d and source term f may all depend on x. Now write equation (5.7) as a first order system in dependent variables p_1 and p_2:

$$p_2' + cp_2 + dp_1 = f \tag{5.9}$$

$$p_1' - p_2 = 0 \tag{5.10}$$

$$p_1(a) = p_{0,0} \qquad p_2(a) = p_{0,1} \tag{5.11}$$

Introducing vector functions

$$\boldsymbol{u} = \begin{bmatrix} p_1 \\ p_2 \end{bmatrix} \quad \boldsymbol{f} = \begin{bmatrix} f \\ 0 \end{bmatrix}$$

allows us to write a common, more abstract, form of the first order system of the differential equation in (5.9):

$$\boldsymbol{A}(\boldsymbol{u}) = \boldsymbol{f} \tag{5.12}$$

where

$$A(\boldsymbol{u}) = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \boldsymbol{u}' + \begin{bmatrix} d & c \\ 0 & -1 \end{bmatrix} \boldsymbol{u}$$

Using the typical approach for least-squares finite elements, we introduce function spaces $X_0 = \{q \in H^1(a,b) : q(a) = p_{0,0}\}$ and $X_1 = \{q \in H^1(a,b) : q(a) = p_{0,1}\}$. Then construct a quadratic functional $I : X_0 \times X_1 \to R$ in terms of the L_2 norms of the residuals of the differential equations in (5.9):

$$\begin{aligned} I(\boldsymbol{u}) &= \frac{1}{2}||\boldsymbol{A}(\boldsymbol{u}) - \boldsymbol{f}||^2 \\ &= \frac{1}{2}(\boldsymbol{A}(\boldsymbol{u}) - \boldsymbol{f}, \boldsymbol{A}(\boldsymbol{u}) - \boldsymbol{f}) \\ &= \int_a^b [(p_2' + cp_2 + dp_1 - f)^2 + (p_1' - p_2)^2] dx \end{aligned} \tag{5.13}$$

A necessary condition that $\boldsymbol{u} = p_1, p_2 \in X_0 \times X_1$ minimizes $I(\boldsymbol{u})$ is that its first variation vanishes at $\boldsymbol{u}$, i.e.

$$\lim_{t\to 0} \frac{d}{dt} I(\boldsymbol{u} + t\boldsymbol{v}) = 0 \tag{5.14}$$

for all $\boldsymbol{v} = q_1, q_2 \in X_0 \times X_1$. As a result, we have the least-squares variational formulation: Find $\boldsymbol{u} = p_1, p_2 \in X_0 \times X_1$ such that

$$\begin{aligned} \lim_{t\to 0} \frac{d}{dt} I(\mathbf{u} + t\boldsymbol{v}) &= (\boldsymbol{A}(\boldsymbol{u}) - \boldsymbol{f}, \boldsymbol{A}(\boldsymbol{v})) \\ &= \int_a^b \left[(p_2' + cp_2 + dp_1 - f)(q_2' + cq_2 + dq_1) + (p_1' - p_2)(q_1' - q_2)\right] dx = 0 \end{aligned} \tag{5.15}$$

for all $\boldsymbol{v} = q_1, q_2 \in X_0 \times X_1$.

5.3 Numerical Results

We first solve equation (5.5) for the constant viscosity case, where $\mu = \mu_0 + 0\frac{x}{L}$ and so $\mu' = 0$ with inflow boundary conditions $P(0)$ and $P'(0)$ specified. The exact solution is the same analytical solution found in Chapter 2,

$$P(x) = P_\infty + (P_0 - P_\infty)\cosh\frac{x}{\lambda_c} - \Delta P_{\text{ref}} \sinh(\frac{x}{\lambda_c}) \tag{5.16}$$

where

$$\lambda = \sqrt{\frac{D^3 H}{128\Gamma}} \qquad \Delta P_{\text{ref}} = \frac{Q_0 \mu H}{\pi D \lambda \Gamma} \tag{5.17}$$

with inflow conditions defined as

$$P(0) = P_0 \qquad P'(0) = -\frac{\Delta P_{\text{ref}}}{\lambda} \tag{5.18}$$

Table 5.1 shows the input parameters for this case. Continuous piecewise linear basis functions are used for all trial and test functions. Convergence results are in Table 5.2 and the solutions plots are in Figure 5.1.

Description	Parameter	Value
Length	L	0.5 m
Inside diameter	D	0.005 m
Media thickness	H	0.0004 m
Permeability	Γ	$0.546\times10^{-12}m^2$
Dynamic viscosity	μ	1×10^{-5} Pa-s
Inlet flow rate	Q_0	0.5 $\frac{m^3}{sec}$
Inlet pressure	P_0	850000 Pa
Permeate pressure	P_∞	100000 Pa

Table 5.1: Parameter values used for constant μ case

	L_2 Error		Max error	
n	p	$\frac{dp}{dx}$	p	$\frac{dp}{dx}$
8	4.39	34.58	17.35	78.54
16	1.10	8.65	4.34	19.63
32	0.274	2.16	1.09	4.91
64	0.068	0.54	0.27	1.23

Table 5.2: Convergence results for constant μ case

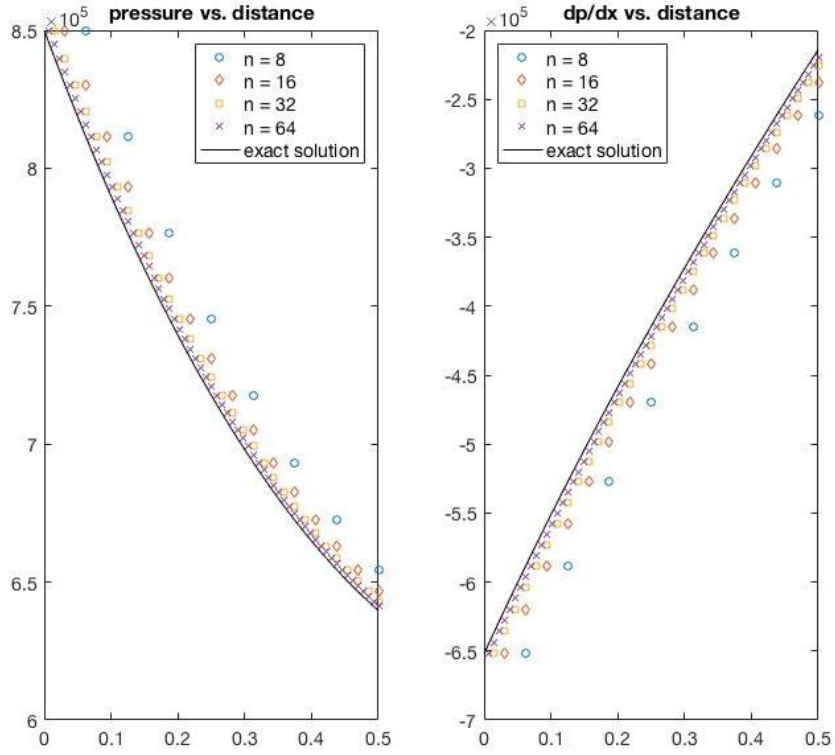

Figure 5.1: Results for constant μ

Now we consider the variable viscosity case, letting the viscosity vary linearly as defined in equation (5.2). Results are plotted in Figure 5.2 were computed using the parameters in Table 5.1, with μ_0 set to the value of μ in Table 5.1, and letting a_μ in equation (5.2) vary between 0 and 8×10^{-6}. Note the increase in pressure drop from the inlet to the outlet as a_μ increases. In a physical setting, the pressure at the inlet might be allowed to increase as viscosity increases in order to maintain a uniform permeate flow rate.

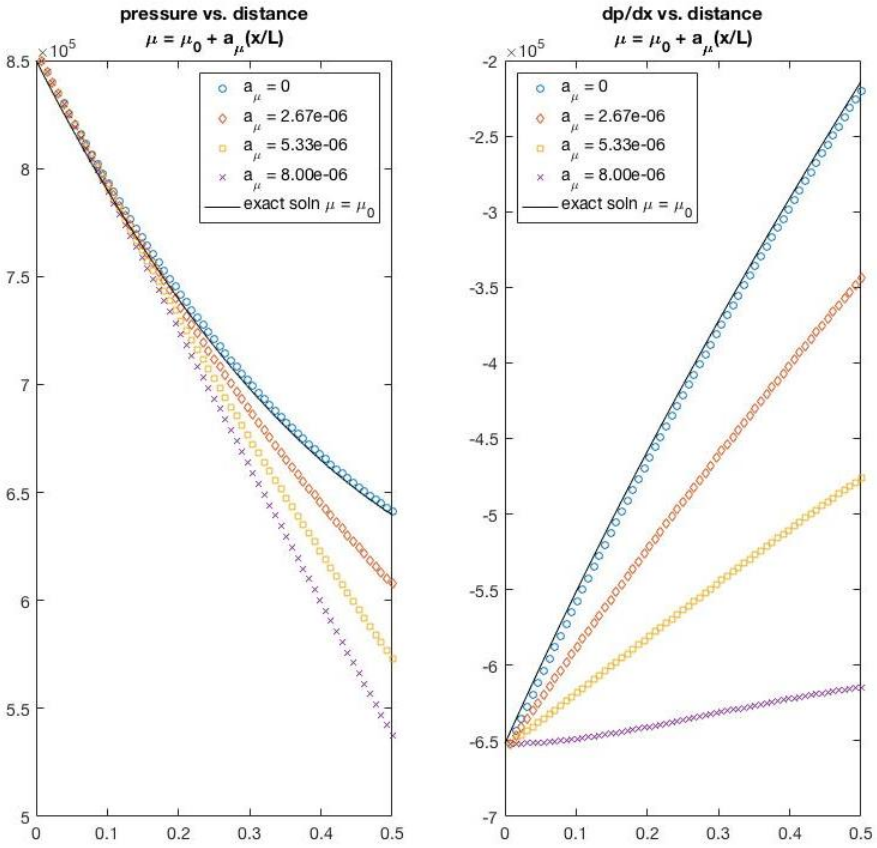

Figure 5.2: Results for variable μ

Chapter 6

Conclusions and Suggestions for Further Work

In this thesis we explored one dimensional mathematical models for cross-flow filtration. We began by considering the clean filter case for which analytical solutions are given. In the next case we presented a cross-flow filter with fouling in both the form of cake build up and depth plugging, and presented the corresponding analytical solutions. A web application simulating the filter with fouling was also created and presented here, along with instructions for the set up of much simpler model using the same framework, that is extensible to more complicated models. The web application presents an attractive alternative to traditional means of sharing simulations, in that it allows users to generate results without having to download, install, set up, alter, or run any code, eliminating all versioning and compatibility issues. While the software engineering required for a more complicated model can be extensive, once a framework is established extending to other mathematical models is much less intensive.

Suggestions for future work:

- Integrate finite element solution into web application
- Develop the finite element solution to include more realistic scenarios, including:
  - Channels of varying thickness
  - Two and three-dimensional geometries

  - More complex flow region/filter media boundary conditions
  - The 'flowing cake' phenomenon, using non-Newtonian theology equations
- Multiple filter units and other scaling-related studies

We close with some observations made in the last section of [10].

- The models presented in this paper can be classified as solutions to Category I problems, i.e. predicting results based on a given set of parameters. The long-term goals of this effort include Category II problems that determine input parameters to produce a desired result. The Category II problem for the filter problem involves prescribing a filter performance envelope for which an optimal and feasible set of filter design parameters can be obtained.
- While Category I problems have unique solutions, the solutions of Category II problems are not unique, but instead form a family of solutions that satisfy a set of composite parameters and require other auxiliary constraints to identify the best solution.
- There are significant unknowns, yet to be considered, that may compromise the performance of any filter intended for realistic applications. For example, applications motivating this thesis are related to deep space missions of long duration. For these applications, possible continuous and irreversible accumulation of dispersed phase materials in the pore volume (i.e. the slow death of the media) needs to be investigated and thoroughly understood.

Appendices

Appendix A Code for Chapter 2 web application development of sine wave

settings.py

```
"""
Django settings for sinsite project.

Generated by 'django-admin startproject' using Django 1.8.4.

For more information on this file, see
https://docs.djangoproject.com/en/1.8/topics/settings/

For the full list of settings and their values, see
https://docs.djangoproject.com/en/1.8/ref/settings/
"""

# Build paths inside the project like this: os.path.join(BASE_DIR, ...)
DEBUG = True
TEMPLATE_DEBUG = DEBUG

import os
BASE_DIR = os.path.dirname(os.path.dirname(os.path.abspath(__file__)))



# Quick-start development settings - unsuitable for production
# See https://docs.djangoproject.com/en/1.8/howto/deployment/checklist/

# SECURITY WARNING: keep the secret key used in production secret!
SECRET_KEY = 'f##j9mau5t+nb-343!+uh(-*e*=*#k63%)&6^o)kjiu=(d)ru6'

# SECURITY WARNING: don't run with debug turned on in production!

ALLOWED_HOSTS = []



# Application definition

INSTALLED_APPS = (
```

```
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
)

MIDDLEWARE_CLASSES = (
    'django.contrib.sessions.middleware.SessionMiddleware',
    'django.middleware.common.CommonMiddleware',
    'django.middleware.csrf.CsrfViewMiddleware',
    'django.contrib.auth.middleware.AuthenticationMiddleware',
    'django.contrib.auth.middleware.SessionAuthenticationMiddleware',
    'django.contrib.messages.middleware.MessageMiddleware',
    'django.middleware.clickjacking.XFrameOptionsMiddleware',
    'django.middleware.security.SecurityMiddleware',
)

ROOT_URLCONF = 'sinsite.urls'


TEMPLATES = [
    {
        'BACKEND': 'django.template.backends.django.DjangoTemplates',
        'DIRS': [os.path.join(BASE_DIR, 'templates')],
        'APP_DIRS': True,
        'OPTIONS': {
            'context_processors': [
                'django.template.context_processors.debug',
                'django.template.context_processors.request',
                'django.contrib.auth.context_processors.auth',
                'django.contrib.messages.context_processors.messages',
            ],
        },
    },
]

WSGI_APPLICATION = 'sinsite.wsgi.application'
```

```
# Database
# https://docs.djangoproject.com/en/1.8/ref/settings/#databases

DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.sqlite3',
        'NAME': os.path.join(BASE_DIR, 'db.sqlite3'),
    }
}


# Internationalization
# https://docs.djangoproject.com/en/1.8/topics/i18n/

LANGUAGE_CODE = 'en-us'

TIME_ZONE = 'EST'

USE_I18N = True

USE_L10N = True

USE_TZ = True


# Static files (CSS, JavaScript, Images)
# https://docs.djangoproject.com/en/1.8/howto/static-files/
MEDIA_ROOT = os.path.join(BASE_DIR, 'media')

MEDIA_URL = '/media/'

STATIC_ROOT = os.path.join(BASE_DIR, 'static')
STATIC_URL = '/static/'

STATICFILES_DIRS = (
    os.path.join(BASE_DIR, 'sinsite/staticfiles'),
    )
```

views.py

```
from django.shortcuts import render, render_to_response
from django.http import HttpResponse
from django.template.response import TemplateResponse
from django.template import RequestContext, loader
import numpy as np
import math



def post_form_upload(request):
  return render(request, 'admin/post_form_upload.html')

def plot(request):
  if 'amplitude' in request.GET:
    if 'period' in request.GET:
      amplitude = float(request.GET['amplitude'])
      period = float(request.GET['period'])
      arrayx = np.linspace(0, np.pi*2, 1000)
      arrayy = np.sin(period*arrayx)*amplitude
      arrayx = np.array_str(arrayx)
      arrayy = np.array_str(arrayy)
  return render(request, 'admin/plot.html', {'arrayy': arrayy, 'arrayx': arrayx} )
```

urls.py

```
from django.conf.urls import url, patterns, include
from django.conf.urls.static import static
from django.contrib import admin
from django.contrib.staticfiles.urls import staticfiles_urlpatterns
from sinsite import settings

from . import views



urlpatterns = patterns('',
  url(r'^post/form_upload/$', 'sin.views.post_form_upload', name='post_form_upload'),
  url(r'^post/plot/$', 'sin.views.plot', name='plot'),
) + static(settings.MEDIA_URL, document_root=settings.MEDIA_ROOT)
```

```
urlpatterns += staticfiles_urlpatterns()
```

index.js

```
var arrayx = document.getElementById("arrayx").value;
var arrayy = document.getElementById("arrayy").value;



var newx = arrayx.split(/ {1,}/);
var newy = arrayy.split(/ {1,}/);

var trace = {
  x: newx,
  y: newy,
  mode: 'markers'
};

var data = [trace];
Plotly.newPlot('sinplot', data);

console.log(arrayx);
console.log(newx);
```

A file to manage security was not created for this application, instead an excerpt from the cross-flow filtration model's security is included.

input_validation.js

```


function formValidation () {
  var L, D, H, mu, Q0, P0, Pinfty, Gamma, C0;


  domain = document.forms["inputForm"]["domain"].value;
  L = document.forms["inputForm"]["L"].value;
  D = document.forms["inputForm"]["D"].value;
  H = document.forms["inputForm"]["H"].value;
  mu = document.forms["inputForm"]["mu"].value;
  Q0 = document.forms["inputForm"]["Q0"].value;
  P0 = document.forms["inputForm"]["P0"].value;
  Pinfty = document.forms["inputForm"]["Pinfty"].value;
```

```
  Gamma = document.forms["inputForm"]["Gamma"].value;
  C0 = document.forms["inputForm"]["C0"].value;

  if(isNaN(domain) || domain ~= 1 || domain ~=2 ) {
    return false;
  }

  if(isNaN(L) || L < 0.00000000001 || L > 100000 ) {
    alert("Invalid Input for Variable L");
    return false;
  }

  if(isNaN(D) || D < 0.00000000001 || D > 100000 ) {
    alert("Invalid Input for Variable D");
    return false;
  }

  .
  .
  .
  .

  return true;

}
```

post_form_upload.html

```
<html>
<head>
<title>Sine Plotting</title>
<head/>
<body>
Enter an Amplitude and Period for your Sine Wave
<form action="/sin/post/plot/" method="get">
<br/>Amplitude:
<input type="number" name="amplitude"><br/>
```

```
Period:
<input type="number" name="period"><br/><br/>
<input type="submit" value="Submit">
<form/>
</body>
</html>
```

plot.html

```
<html>
<head>
<title>Plot</title>
{% load static %}
<script type="text/javascript" src="{% static 'js/d3.js' %}"></script>
<script type="text/javascript" src="{% static 'js/plotly-latest.min.js' %}"></script>
</head>
<body>
<input type="hidden" id="arrayx" name="arrayx" value="{{ arrayx }}">
<input type="hidden" id="arrayy" name="arrayy" value="{{ arrayy }}">
Plot of your curve
<div id="sinplot" style="width:600px;height:600px;"></div>

{% load static %}
<script type="text/javascript" src="{% static 'js/index.js' %}"></script>

</body>
</html>
```

Bibliography

[1] Anaconda. Anaconda documentation. `https://conda.io/docs/index.html`, 2017.

[2] N. Andreas. A generalization of the Darcy equation to represent transient filter behavior. *Advances in Filtration and Separation Technology*, 7:102–105, 1993.

[3] N. Andreas. Distribution function for simultaneous cake and depth filtration deduced from continuity equation and pressure drop response. *Fluid/Particle Separations Journal*, 16(2):155–158, 2004.

[4] N. Andreas and C. Cox. New cross flow filter module design parameters: A theoretical analysis of cross flow filter performance limits. *Filtration*, 13(4):247–256, 2013.

[5] N. Andreas and C. Cox. New crossflow filter module design parameters: A theoretical analysis of crossflow filter performance limits. *Filtration Solutions*, 13(4):247–256, 2013.

[6] N. Andreas, C. Cox, K. Azuma, and M. Tamura. Transient behavior of cross flow filters: Fouling effects. Preprint.

[7] N. Andreas, C. Cox, T. Kato, and M. Tamura. A model for transient cross flow filtration in a narrow rectangular domain. *Separation and Purification Technology*, 156(1):36–41, 2015.

[8] N. Andreas, C. Cox, T. Kato, and M. Tamura. A model for transient cross flow filtration in a narrow rectangular domain. *Separation and Purification Technology*, 2015.

[9] N. Andreas, C. Cox, T. Kato, and M. Tamura. Regenerable cross flow HEPA filter with integral biohazard treatment stage. In *45th International Conference on Environmental Systems*, ICES-2015-335.

[10] N. Andreas, C. Cox, L. McIntyre, T. Kato, and M. Tamura. design optimization of regenerable super hepa cross flow filters for application on spacecraft and harvesting of martian atmospheric carbon dioxide. In *47th International Conference on Environmental Systems, 16-20 July 2017, Charleston, South Carolina*, ICES-2017-222.

[11] P. Blanpain and M. Lalande. Investigation of fouling mechanisms governing permeate flux in the crossflow microfiltration of beer. *Filtration Separation*, pages 1065–1069, December 1997.

[12] Django. Django documentation. `https://docs.djangoproject.com/en/1.11/`, 2017.

[13] Django. Django time zones. `https://docs.djangoproject.com/en/1.8/topics/i18n/timezones/`, 2017.

[14] R. Green, R. Vihayakumar, and J. Agui. Development of test protocols for international space station particulate filters. In *International Conference on Environmental Systems*.

[15] F. Heisler. Python programming by example. `https://realpython.com/`, 2017.

[16] S. Jacob and M. Jaffrin. Purification of brown cane sugar solutions by ultrafiltration with ceramic membranes: Investigation of membrane fouling. *Separation Science Technology*, 35(7):989–1010, 2000.

[17] V. Jegatheesan, D. Phong, L. Shu, and R. Aim. Performance of ceramic micro- and ultrafiltration membranes in treating limed and partially clarified sugar cane juice. *Journal of Membrane Science*, 327:69–77, 2009.

[18] B-N. Jiang. *The Least-Squares Finite Element Method*. Springer-Verlag, 1998.

[19] J. Osorio. Simulation of fluid flow through porous media. Master's thesis, Universidad Nacional de Colombia, 2015.

[20] J. Perry and M. Abney. Evaluation of an atmosphere revitalization subsystem for deep space exploration missions. In *International Conference on Environmental Systems*.

[21] Plotly. Plotly documentation. `https://plot.ly/javascript/`, 2017.

[22] S. Ripperger and J. Altmann. Crossflow microfiltration state of the art. *Separation and Purification Technology*, 26:19–31, 2002.

[23] V. Sibanda, R. Greenwood, and J. Seville. Particle separation from gases using cross-flow filtration. *Powder Technology*, 2001.

[24] C. Vassilieff, T. Doneva, and L. Ljutov. Cross-flow microfiltration of bentonite-in-water dispersions: Initial transient effects at low concentration. *Membrane Science*, 1996.

[25] W. Wang, X. Jia, and G. Davies. A theoretical study of transient cross-flow filtration using force balance analysis. *The Chemical Engineering Journal*, 60:55–62, 1995.