Modeling the Delivery of Dissolved Gases for Diabetes Research

Lui Tsumura

A thesis
submitted in partial fulfillment of the
requirements for the degree of

Master of Science

University of Washington

2024

Committee:
James Hermanson
John Kramlich

Program Authorized to Offer Degree:
Aeronautics & Astronautics

University of Washington

Abstract

Modeling the Delivery of Dissolved Gases for Diabetes Research

Lui Tsumura

Chair of the Supervisory Committee:
James Hermanson
Department of Aeronautics and Astronautics

Understanding cellular metabolism and its interaction with oxygen is vital in diabetes research. This requires a quantitative understanding of the oxygen concentration at the tissue. To address this issue, a 2-D flow model is developed to quantify the oxygen reaction rate of the tissue. In addition, an experimental setup was developed to provide a precise, rapid control over the delivered gas concentration. Mass flow controllers were implemented to accurately regulate the gas flow rate, integrating Arduino and Python programming to remove the need for manual adjustments. This approach also shortened the steady state time of the oxygen concentration within the media, thereby significantly cutting down the time required to conduct experiments.

The purpose of the computational model is to help understand the rate controlling processes governing the uptake by oxygen by the cells. This is broadly governed by two processes that are in series:

1. The transport of O_2 from the free stream to the reaction sites.
2. The rate of O_2 consumption at the reaction sites.

This model used the finite difference method to discretize the governing equations, which included oxygen diffusion, advection, and most importantly, the oxygen reaction around the tissue. Appropriate boundary conditions were applied at the inflow, outflow, walls, and for the tissue surface. To model the enzyme kinetics of the oxygen reaction, the Michaelis Menten equation was utilized along the surface of the cell. The concentration field was calculated by solving a large system of equations, for which the Newton Raphson method was employed. The concentration field provided a visual representation of the flow field, and was used to calculate the oxygen consumption rate (OCR) of the tissue, which was then compared with experimental data. The results were in broad agreement with the oxygen consumption rates seen in the experiments.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGEMENTS

I would like to express my immeasurable gratitude towards my advisor, Prof. Hermanson, for his kind assistance throughout my research. He always gave encouraging words to me, even during times when progress was slow and time was a limited. I would also like to thank Prof. Kramlich for giving insight to many of my work, as well as helping me understand work from the biology team. Finally, my sincere thanks go to my advisor from the biology team, Dr. Ian Sweet, for giving me the opportunity, and freedom to work on this research. This work was supported by National Institute of Health grant number 1R01GM148741-01.

Chapter 1. **INTRODUCTION**

Accurately delivering the dissolved gas concentration is essential in understanding cellular metabolism and its interaction with oxygen, particularly important for diabetes research. Understanding this process is complex and challenging due to the intricate interactions between the oxygen and cellular mechanisms. To model such complex systems, a 2-D flow model was developed to describe the transport and reaction of oxygen around a tissue sample. The model provides insight into how the oxygen reaction is governed, whether by the mass transport of oxygen to the cells, or through the volumetric reaction rate within the cells.

Ensuring a proper experimental setup is crucial for obtaining accurate data. Inadequate equipment or data recording methods can compromise data quality and reliability. To ensure a non-user-friendly setup, multiple mass flow controllers were installed to precisely control the inflow gas concentration. This enhancement led to an improvement in the overall quality of the experimental setup, as indicated by a rapid change in dissolved oxygen concentration.

1.1 EXPERIMENTAL SETUP

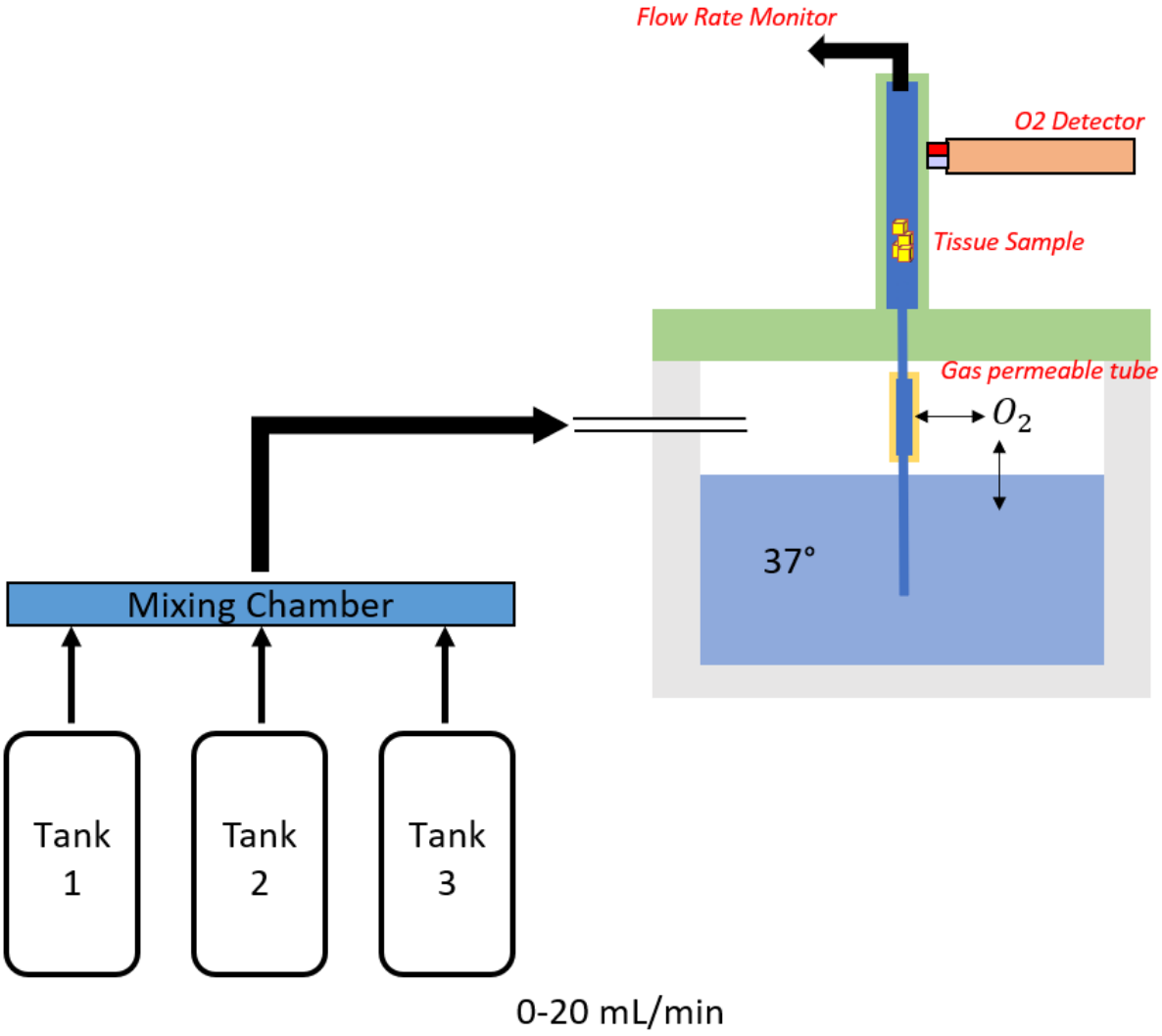


Figure 1. Experimental setup with three gas tanks

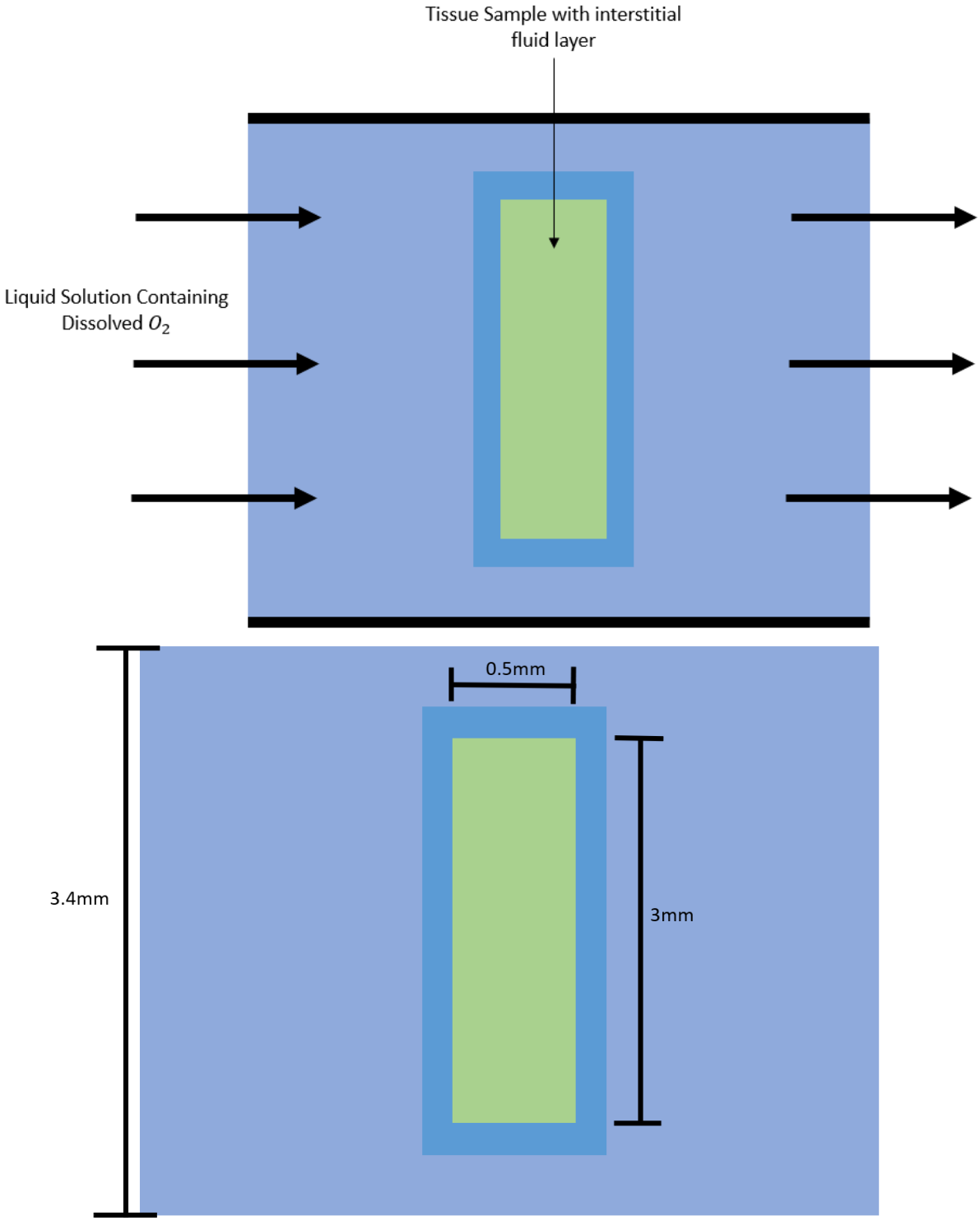


Figure 2. Schematic of the flow around the tissue sample, as well as its dimensions

The physical formulation of the experimental setup is shown in Fig. 1. It consists of sending streams of gas with different flow rates into a mixing chamber to create a desired inflow concentration. The mixed gas is then introduced into a perfusion chamber, where it diffuses into the liquid media mostly from direct contact, but also via the gas permeable tube. The liquid then flows through a tissue sample, where the tissue absorbs oxygen from the surrounding fluid. An O_2 detector then measures the oxygen concentration, followed by a flow rate monitor that records the flow rate at the outlet.

The computational model focused on the flow region around the tissue sample, shown in Fig. 2. In the model, a horizontal cylindrical tube contains a flow of aqueous solution with dissolved O_2 where tissue samples are fixed within the flow. O_2 is transported to the tissue sample by the combination of convection of fluid and the diffusion of O_2 through the fluid. O_2 at the surface of the tissue diffuses through stagnant, interstitial fluid within the tissue before reaching the cell wall, where the oxygen is consumed according to the Michaelis Menten equation [1].

Chapter 2. **MASS FLOW CONTROLLERS**

2.1 WIRING THE MASS FLOW CONTROLLERS

The addition of a mass flow controller eliminates the need for manual valve adjustments on the gas tanks to alter the gas concentration entering the system. By using a computer to achieve precise gas flow rates, the mass flow controllers facilitate a significant improvement in experimental quality and reliability.

Figure 3. FMA 5400A/FMA 5500A series mass flow controller [2]

The improved experimental setup utilized FMA 5504A mass flow controllers, which output flow rates between 0 to 20 ml/min. The three gas tanks were filled with air, and each one was connected to a mass flow controller. After changing the settings and properly wiring the mass flow controller to an Arduino, a Python code was written to develop a user-friendly interface for operating the controller. Using Python enabled the creation of a user-friendly UI, which greatly simplified the process of recording and controlling the flow rate.

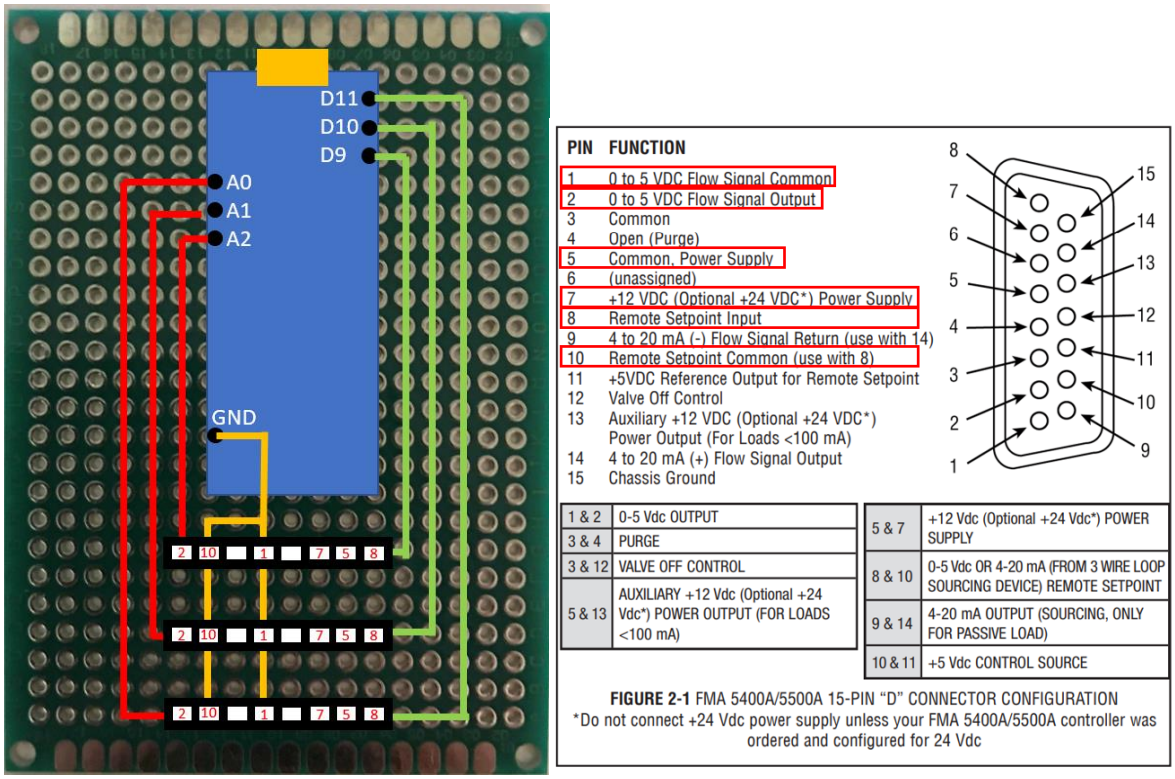


Figure 4. Wiring for the Arduino Nano (left). 15 Pin connector with important pins outlined in red (right)

15-pin connectors were utilized to make a connection between the Arduino and each of the mass flow controllers. However, several features on the controller were unnecessary for the experiment, so only 6 out of the 15 pins were used for wiring. Only the necessary wires from the 15-pin connectors were inserted into their respective female pins, as shown in Fig. 4. For each controller, a 12V DC power supply was connected to pin 7, with the common connected to pin 5. The analog pins on the Arduino were used to measure the flow rate of each controller, and the digital PWM pins were used to send a voltage signal between 0~5V to control the flow rate ranging from 0 to 20 mL/min. Once the controllers were prepared for use, they were connected to the gas tanks using a thin tubber tube for gas delivery.

2.2 IMPROVING THE GAS-CONCENTRATION RESPONSE TIME

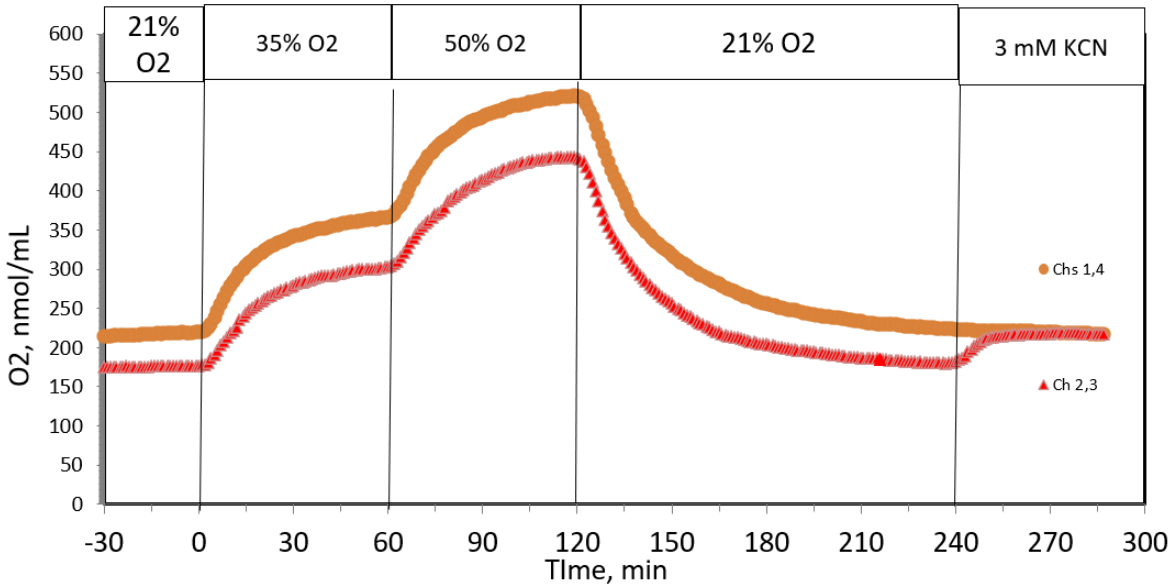


Figure 5. Time changes in oxygen concentration. With tissue (red); Without tissue (orange)

The original experimental setup consisted of two oxygen tanks: one with 50% O_2 concentration and the other with 21% O_2 concentration. The tank with 50% O_2 provided a constant flow rate, while the user manually adjusted the valve on the 21% O_2 tank to control the inflow concentration entering the system. The oxygen concentration of the liquid media was measured downstream after passing the tissue. The prolonged length of time required for the outflow oxygen concentration to reach steady state is apparent in the results shown in Fig. 5. Programming the mass flow controller on the 21% O_2 tank can decrease the steady state time, enabling experiments to be conducted more rapidly.

A rate equation for the inflow concentration was solved for so the mass flow controller can be properly primed. Block diagrams (see Fig. 6) were drawn to easily visualize the system,

as well as the use of transfer functions and Laplace transforms to calculate the inflow concentration which decreased the steady state time.

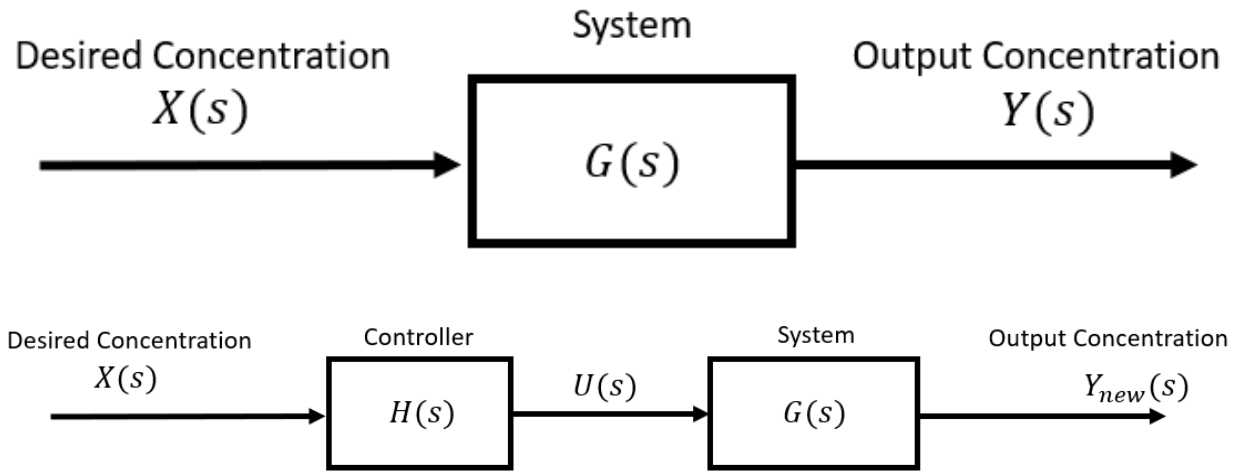


Figure 6. Block diagrams for the original system (top) and the improved system (bottom)

The original system was modeled using a simple input and output block diagram shown in Fig. 6. The transfer function $G(s)$ was solved for by knowing that the inflow concentration $X(s)$ was constant, and the outflow concentration $Y(s)$ was approximated to be an exponential function, as seen from the experimental results in Fig. 5. In the Laplacian domain, this was found to be:

$$X(s) = \frac{g}{s}, \quad Y(s) = \frac{g}{s} - \frac{g}{s+m}, \tag{2.1}$$

where g is the gain, defined to be the difference between the final and initial O_2 level ($g = f - i$). m is a constant which differs depending on the initial and final O_2 level. An estimate of $m = 0.1$ was found to be sufficient enough to approximate most experimental results. $G(s)$ was then found to be:

$$G(s) = \frac{Y(s)}{X(s)} = \frac{m}{s+m}, \tag{2.2}$$

The improved system shown in Fig. 6 added a controller to correct the inflow concentration which fed into the system. The desired outflow concentration $Y_{new}(s)$ was approximated similarly to Eq. (2.1):

$$Y_{new}(s) = \frac{g}{s} - \frac{g}{s+n}, \tag{2.3}$$

where n is a constant which differs depending on the desired outflow concentration. $U(s)$ represents the inflow O_2 concentration going into the system, and it was solved for using $G(s)$ and $Y_{new}(s)$ from Eq. (2.2) and (2.3) respectively:

$$U(s) = \frac{Y_{new}(s)}{G(s)} = \frac{g(m+n)}{m(s+n)} + \frac{g}{s}, \tag{2.4}$$

To get the inflow concentration $U(s)$ in the time domain, its inverse Laplace transform was taken. Additionally, the initial concentration i was added to ensure the equation started at the correct concentration at $t = 0$.

$$u(t) = g\left(1 + \frac{n}{m}\right)e^{-nt} + i, \tag{2.5}$$

Thus, to achieve the desired outflow concentration as in Eq. (2.3), the mass flow controller must be programmed to output a flow rate that, combined with the constant flow rate from the 50% O_2 tank, creates an inflow concentration described by Eq. (2.5).

Chapter 3. **EFFECTIVENSS OF MASS FLOW CONTROLLERS**

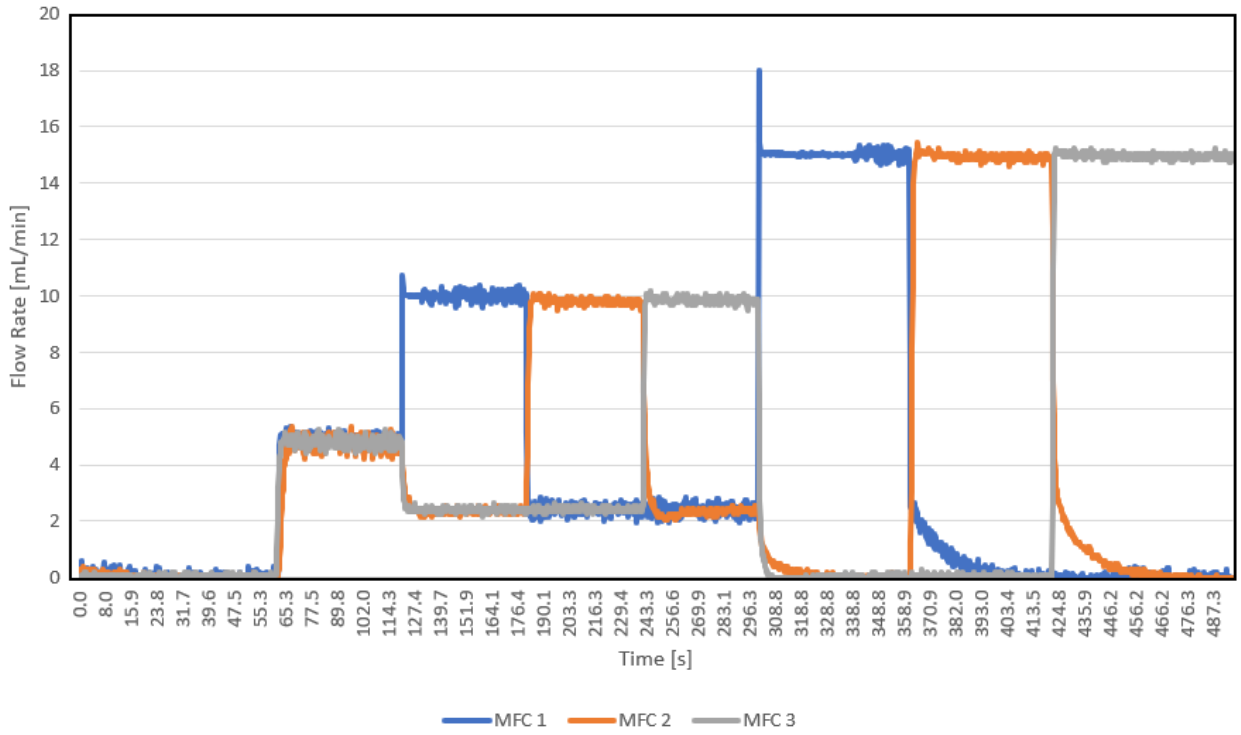


Figure 7. Changes in oxygen gas flow rate with three mass flow controllers

By successfully gaining control over the gas flow rate, the inflow concentration could be regulated using a computer, rather than manually adjusting the valve on each tank. The experimental data shown in Fig. 7 demonstrated this achievement by controlling three mass flow controllers simultaneously. However, the data show minor fluctuation and overshoots. The observed fluctuations can be somewhat accounted for by the mass flow controller's accuracy of $\pm 0.2\,\frac{ml}{min}$, with most of the experimental data falling inside this range. Interestingly, only the first mass flow controller exhibited an overshoot. While the reason for this is still uncertain, potential causes include loose wiring or poor soldering. Additional experimental data for this setup are given in the Appendix, showing similar behaviors.

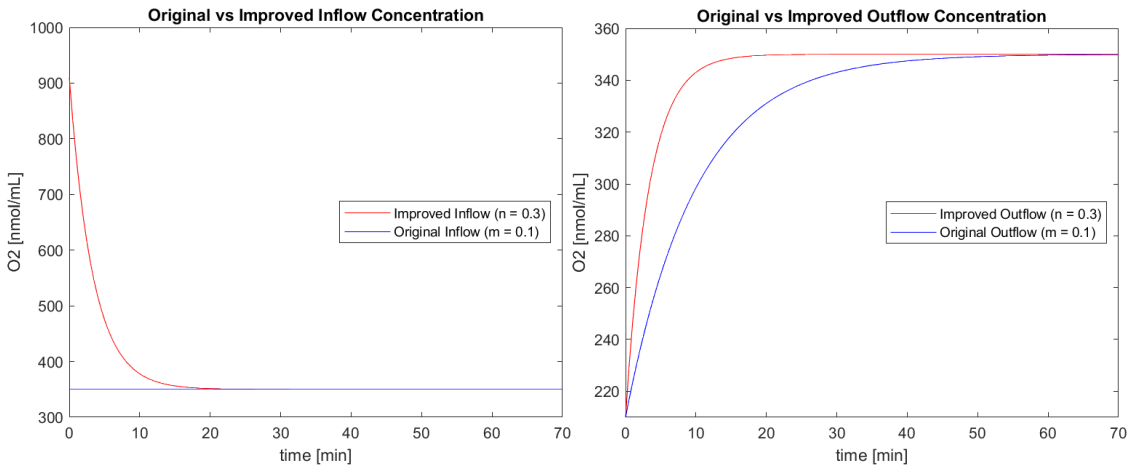


Figure 8. Comparison between original and improved inflow oxygen concentration (left) and outflow oxygen concentration (right)

Simulation of an original and improved flow, with and without mass flow controller respectively, is illustrated in Fig. 8. The previous experimental setup could only provide a constant inflow concentration, but the addition of a mass flow controller enabled the inflow concentration to be rapidly adjusted over time. By priming the mass flow controller with a time-varying inflow concentration, an improved outflow concentration response and reduced steady state time can be achieved, as depicted by the red lines. An increase in n (Eq. (2.3)) will result in a faster steady state time, though it may introduce overshoots or undershoots in the data. Additionally, steady state time will decrease only if n is greater than m. Therefore, users must determine the optimal values of n and m for their experimental setup before programming the mass flow controllers.

Chapter 4. **NUMERICAL MODELING**

The numerical model simulates the transport of oxygen within the flow region, diffusion through the interstitial fluid layer, as well as the oxygen reaction on the cell surface utilizing the Michaelis Menten equation. The governing equations are discretized using the finite difference method, forming a system of linear equations that is then solved using the Newton Raphson method. The resulting concentration field aids in understanding the mechanism of oxygen diffusion and reaction surrounding the cell, and with increased accuracy, it could be used to reliably compare with, or even replace, experimental data.

4.1 **GOVERNING EQUATIONS**

Symbol	**Definition**	**Value**	**Unit**
Δx	Cell size	0.005	mm
R	Radius of annulus	1.7	mm
L	Length of annulus	9	mm
V	Volume of tissue	3.534	mm^3
$\dot{V}$	Volume flow rate of fluid	0.38333	$\frac{mm^3}{s}$
U_{max}	Maximum fluid velocity	$8.1444 * 10^{-2}$	$\frac{mm}{s}$
A_s	Surface area of tissue	18.849	mm^2
D	Diffusion coefficient oxygen in water at 37°C	$3.027 * 10^{-3}$	$\frac{mm^2}{s}$

D_{isf}	Diffusion coefficient of oxygen in interstitial fluid layer	$1 * 10^{-4}$	$\frac{mm^2}{s}$
V_{max}	Max reaction rate of tissue	$2 * 10^{-3}$	$\frac{nmol}{mm^3 s}$
K_m	Michaelis Menten constant	$5 * 10^{-4}$	$\frac{nmol}{mm^3 s}$

Table 1. Parameters and their values used in model simulation

At the inlet of the tissue chamber, laminar flow is moving parallel to the chamber, until the flow encounters the tissue sample, which causes flow to deviate around it. The flow then relaxes to a uniform laminar flow profile behind the tissue. The local Reynolds number calculated by ANSYS Fluent gave a Reynolds number that was on the order of 10^{-4}, thereby Stokes flow was assumed throughout the flow region, and a simplified version of the Navier-Stokes equation provides an accurate description of the flow profile. Additionally, all equations are based on the steady state assumption to further simplify calculations and computation cost. As liquid flows parallel to the chamber as well as in the radial direction, the two-dimensional flow can be represented in a form consisting of a stream function (ψ) and vorticity (ω) that are described by the following two partial differential equations [4]:

$$\nabla^2\omega - \frac{\omega}{r^2} = 0 \tag{4.1}$$

$$-r\omega = \frac{\partial^2\psi}{\partial z^2} + \frac{\partial^2\psi}{\partial r^2} - \frac{1}{r}\frac{\partial\psi}{\partial r} \tag{4.2}$$

where $\nabla^2 = \frac{\partial^2}{\partial z^2} + \frac{\partial^2}{\partial r^2} + \frac{1}{r}\frac{\partial}{\partial r}$ and the axial and radial velocity (u and v) are recovered via:

$$u = \frac{1}{r}\frac{\partial\psi}{\partial r}\,, v = -\frac{1}{r}\frac{\partial\psi}{\partial z} \tag{4.3}$$

Solutions of these equations provide the fluid velocity field at each point in the flow. The concentration of O_2 at each point in the flow is then provided by the O_2 conservation equation, which describes O_2 transport by the flow of the fluid combined with diffusion and provides the rate at which O_2 is delivered to the surface of the tissue:

$$D\nabla^2 C = u\frac{\partial C}{\partial z} + v\frac{\partial C}{\partial r} \tag{4.4}$$

Once the O_2 has arrived at the surface of the tissue, it diffuses through interstitial fluid, cell membranes and cytosol to reach the mitochondria, where the oxygen is consumed. To have proper compatibility between the free stream fluid and the interstitial fluid, a simple flux equation is applied between the boundaries:

$$D\frac{\partial C}{\partial z} = D_{isf}\frac{C_1 - C_2}{d} \tag{4.5}$$

C_1 and C_2 represents the O_2 concentration on the free stream and the cell surface side respectively. D_{isf} represents the diffusion coefficient of oxygen in water, which was found using the Wilke-Change equation [3] for water at 37^oC. Once the O_2 reaches the mitochondria, it reacts according to the standard Michaelis Menten relation combined with the flux equation:

$$\frac{D_{isf}V}{A_s}\frac{C_1 - C_2}{d} = \frac{V_{max}C_2}{K_m + C_2} \tag{4.6}$$

where K_m is the Michaelis Menten constant, which represents the O_2 concentration when the reaction rate is half of V_{max}. Once the concentration field is calculated using the above equations, the oxygen consumption rate (OCR) is calculated using the following methods:

$$OCR = \frac{2\pi\Delta x}{V_{max}}\sum_{a=0}^{b}\frac{V_{max}C_a}{K_m + C_a}r_a \tag{4.7}$$

$$OCR = \frac{2\pi\Delta x}{V_{max}V}\sum_{j=0}^{N_y} r_j(u_{0,j}C_{0,j} - u_{Nx,j}C_{N_x,j}) \tag{4.8}$$

C_a and r_a are the oxygen concentration and the radial distance along the cell surface respectively. N_y is the total grid points in the radial direction. Eq. (4.7) calculates the OCR by solving the Michaelis Menten equation around the cell and taking its summation, while Eq. (4.8) uses the difference in mass flow rate between the inlet and outlet to obtain the OCR. Both equations are normalized by V_{max} so the values can be compared with experimental results. These equations were used to verify the integrity of the simulation, as they needed to be equal to each other.

4.2 METHOD AND SETUP

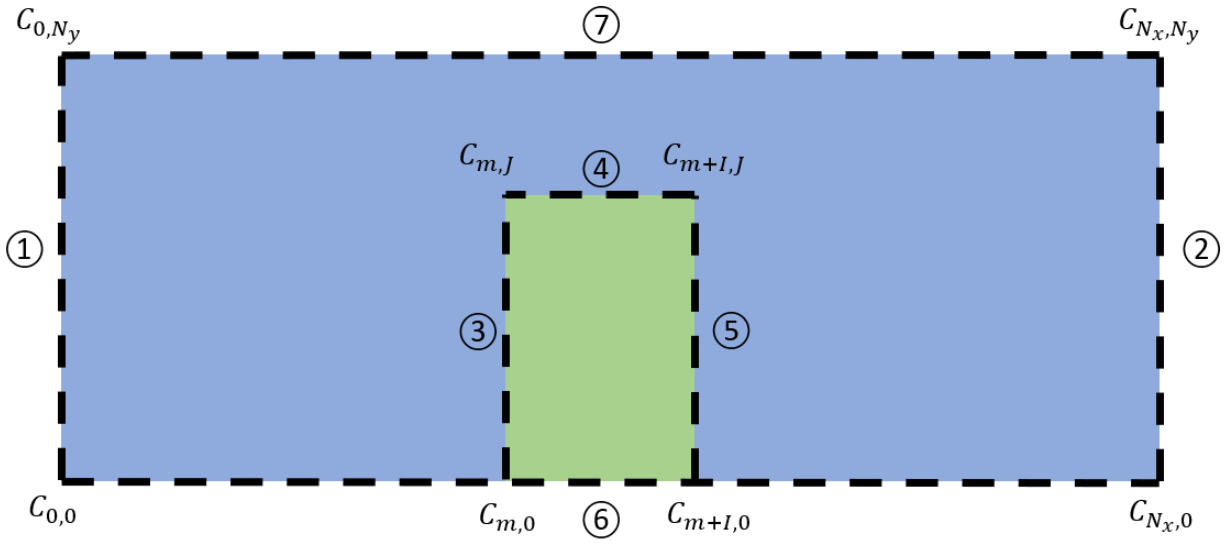


Figure 9. Regions for boundary conditions

The flow regions modeled in the simulations are shown in Fig. 9. The numerical solutions were calculated in two steps. First, the fluid flow equations (4.1) and (4.2) were numerically solved to obtain the radial and axial velocity profiles at each point in the free stream. The resulting profiles were cross checked using ANSYS Fluent software. Then the concentration field was determined by using the solved velocity field. To incorporate oxygen consumption and diffusion thickness layer into the model, Eq. (4.5) and (4.6) were solved concurrently, with their

interaction governed by boundary conditions. With the resulting equations, the O_2 concentration were solved for all given grid points. The O_2 consumption rate was determined by solving both Eq. (4.9) and (4.8), and making sure they equated each other before comparing with experimental results.

The flow domain in Fig. 2 was discretized into square cells with side lengths Δx by Δx. Thus, there are $\frac{R}{\Delta x} + 1$ grid points in the radial direction, and $\frac{L}{\Delta x} + 1$ grid points in the axial direction. These are denoted as N_y and N_x respectively. The Newton Raphson method was used in order to solve the nonlinear differential equations. This was accomplished by creating a system of linear equations in the form:

$$\boldsymbol{J}(\boldsymbol{y_{new}} - \boldsymbol{y}) = \boldsymbol{J}\Delta\boldsymbol{y} = -\boldsymbol{F} \tag{4.10}$$

where $\boldsymbol{J}$ is the Jacobian matrix with size N_xN_y by N_xN_y. $\Delta\boldsymbol{y}$ is the variable being solved for, and **F** is a vector valued function, both having a size of N_xN_y by 1. Both the $\boldsymbol{J}$ and $\boldsymbol{F}$ matrix were stored as a sparse matrix in order to conserve memory space and computation time. The system of equations was simply calculated by using the backslash operator in MATLAB. Due to the flow being axisymmetric around the centerline, only the region above the centerline was solved for. The solution was then mirrored about the centerline to obtain the full flow.

4.3 COMPUTING THE VELOCITY FIELD

The 2D model is centered around solving Eq. (4.1) and (4.2) while applying the correct boundary conditions. To avoid accessing any grid points outside the domain, the appropriate finite difference approximations were used along regions ①~⑦.

For region ①, the inflow stream function and vorticity were defined as a Dirichlet boundary condition. The velocity profile inside an annular tube is well known [5], and so the equations for stream function and vorticity was found using Eq. (4.3) and (4.2) respectively.

$$\psi = \frac{U_{max}}{2} r^2 \left(1 - \frac{1}{2}\left(\frac{r}{R}\right)^2\right), \qquad \omega = \frac{2U_{max} r}{R^2}, \tag{4.11}$$

In region ②, a second order backwards approximation was used to discretize the second derivative.

$$\frac{\partial^2 \psi_{N_x,j}}{\partial z^2} = \frac{-\psi_{N_x-3,j} + 4\psi_{N_x-2,j} - 5\psi_{N_x-1,j} + 2\psi_{N_x,j}}{2\Delta x^2}, \qquad \frac{\partial^2 \omega_{N_x,j}}{\partial z^2} = \frac{-\omega_{N_x-3,j} + 4\omega_{N_x-2,j} - 5\omega_{N_x-1,j} + 2\omega_{N_x,j}}{2\Delta x^2}, \tag{4.12}$$

On regions ③, ④, and ⑤, the stream function was set to a constant value $\psi = 0$. The vorticity was discretized as follows:

$$\omega_{m,j} = \frac{-8\psi_{m-1,j} + \psi_{m-2,j}}{2r_j \Delta x^2}, \quad region\ ③$$
$$\omega_{m+I,j} = \frac{-8\psi_{m+I+1,j} + \psi_{m+I+2,j}}{2r_j \Delta x^2}, \quad region\ ⑤ \tag{4.13}$$
$$\omega_{i,J} = \frac{-8\psi_{i,J+1} + \psi_{i,J+2}}{2R\Delta x^2}, \quad region\ ④$$

For region ⑥, there were multiple types of boundary conditions for ψ and ω:

$$\psi = 0, \qquad \frac{\partial \psi}{\partial r} = 0,$$
$$\omega = 0, \tag{4.14}$$

Finally, for region ⑦ a similar boundary condition to region ④ was applied.

$$\psi = \frac{U_{max}}{4} R^2,$$
$$\omega_{i,N_y} = \frac{7\psi_{i,N_y} - 8\psi_{i,N_y-1} + \psi_{i,N_y-2}}{2R\Delta x^2}, \tag{4.15}$$

After applying the above boundary conditions and solving the stream function and vorticity field, Eq. (4.3) was used to solve for the velocity field after discretizing using second order central approximation:

$$u = \frac{1}{r}\frac{\psi_{i,j+1} - \psi_{i,j-1}}{2\Delta x}, \quad v = -\frac{1}{r}\frac{\psi_{ij+1} - \psi_{i,j}}{2\Delta x}, \tag{4.16}$$

However, problem arose when attempting to solve for the u velocity at $r = 0$. To find a valid solution, L'Hopital's rule was applied to Eq. (4.3).

$$\lim_{r\to 0} \frac{1}{r}\frac{\partial \psi}{\partial r} = \frac{0}{0} \tag{4.17}$$

and so along the centerline, the differential equation for u is:

$$u = \frac{\partial^2 \psi}{\partial r^2} \tag{4.18}$$

While solving for the v velocity at region ②, second order backwards differencing was used to avoid using grid points outside the domain. After discretizing the boundary conditions, Eq. (4.1) and (4.2) were then discretized to obtain a system of linear equations so the Newton Raphson

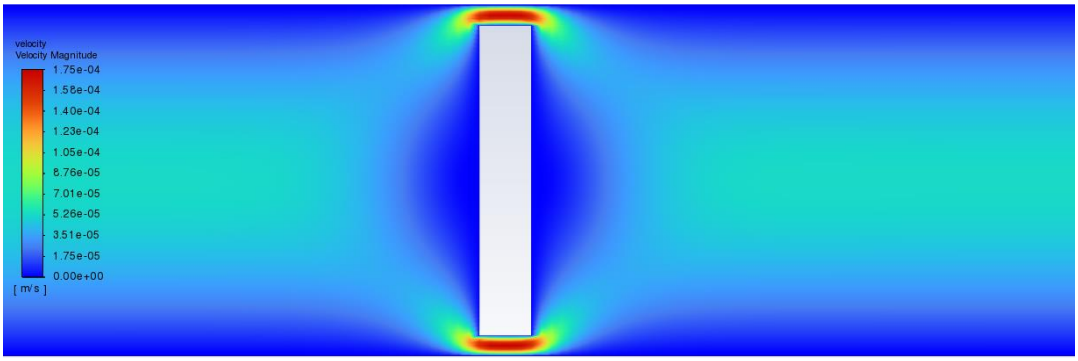
method could be used. The following discretization for ψ and ω were used to find their corresponding values at the next iteration:

$$\psi_{new} = \Delta\psi + \psi$$
$$\omega_{\text{new}} = \Delta\omega + \omega \tag{4.19}$$

Eq. (4.18) was then substituted into the governing equations (4.1) and (4.2). After moving all the delta terms to the left-hand side, and all the current values to the right-hand side, the equation was put into the form $\boldsymbol{J}\Delta\boldsymbol{y} = -\boldsymbol{F}$:

$$\begin{bmatrix} \frac{1}{r}\frac{\partial^2}{\partial r^2} + \frac{1}{r}\frac{\partial^2}{\partial z^2} - \frac{1}{r^2}\frac{\partial}{\partial r} & I \\ 0 & \nabla^2 - \frac{1}{r^2} \end{bmatrix} \begin{bmatrix} \Delta\psi \\ \Delta\omega \end{bmatrix} = - \begin{bmatrix} \frac{1}{r}\frac{\partial^2}{\partial r^2}\psi + \frac{1}{r}\frac{\partial^2}{\partial z^2}\psi - \frac{1}{r^2}\frac{\partial}{\partial r}\psi + \omega \\ \nabla^2\omega - \frac{1}{r^2}\omega \end{bmatrix} \tag{4.20}$$

The initial guess for both streamlines and vorticity were set to the inflow condition everywhere. Once $\Delta\psi$ and Δω were solved for, Eq. (4.21) and (4.3) were used in conjunction to solve for the velocity field at all grid points. Below is the velocity field calculated in the CFD software FLUENT with the same geometry and inflow conditions, as well as the velocity field computed in MATLAB.



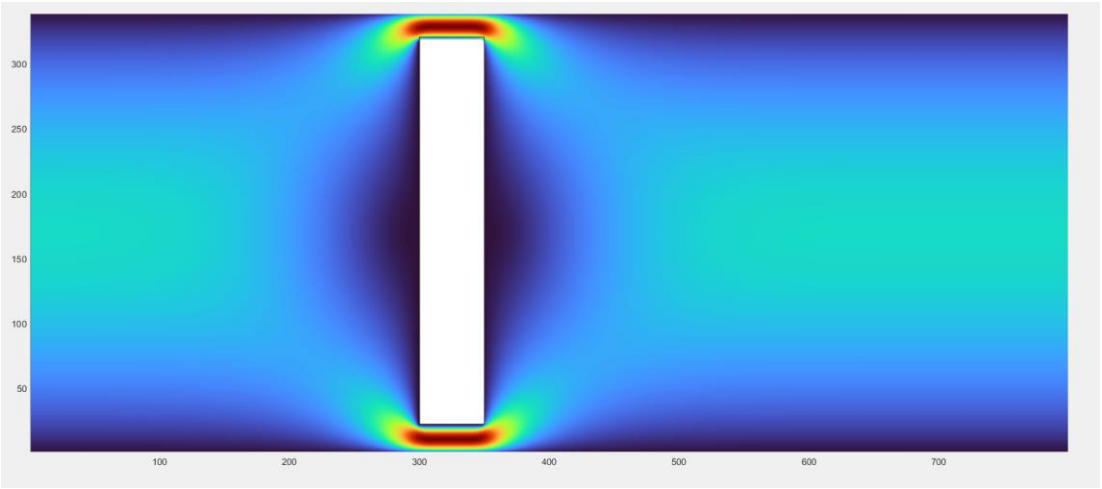


Figure 10. Velocity profile from Ansys (top); Velocity profile from MATLAB (bottom)

4.4 COMPUTING THE CONCENTRATION FIELD

The 2D model for the concentration field is centered around solving Eq. (4.4), (4.5), and (4.6) while applying the correct boundary conditions. Similar to the velocity field, in order to avoid accessing any grid points outside the domain, it was necessary to use appropriate finite difference approximation along regions ①~⑦.

Dirichlet boundary conditions were applied at region ①, where the concentration was set to the user specified inflow concentration. A second order backwards difference approximation at region ② was used to calculate the oxygen concentration:

$$\frac{\partial C_{N_x,j}}{\partial z} = \frac{C_{N_x-2,j} - 4C_{N_x-1,j} + 3C_{N_x,j}}{2\Delta x}$$

$$\frac{\partial^2 C_{N_x,j}}{\partial z^2} = \frac{-C_{N_x-3,j} + 4C_{N_x-2,j} - 5C_{N_x-1,j} + 2C_{N_x,j}}{2\Delta x^2} \quad (4.22)$$

The surface consumption on regions ③, ④, and ⑤ were modeled using Eq. (4.1). The grid points adjacent to those surfaces were modeled using Eq. (4.4) to represent diffusion through the ISF layer. Region ⑥ represents the centerline ($r = 0$), and is the only region which is able to access grid points outside the domain due to axisymmetric flow. Thus, along ⑥:

$$C_{i,j} = C_{i,-j}$$
$$\frac{\partial C}{\partial r} = 0 \tag{4.23}$$

Similar to when solving for the velocity, the same problem arises when trying to calculate the concentration at $r = 0$. L'Hopital's rule was applied once again. The following differential equation was used to solve for the concentration along the centerline:

$$D\left(\frac{\partial^2 C}{\partial z^2} + 2\frac{\partial^2 C}{\partial r^2}\right) = u\frac{\partial C}{\partial z} + v\frac{\partial C}{\partial r} \tag{4.24}$$

which was then discretized using second order central differencing. For region ⑦, the following boundary conditions were used:

$$\frac{\partial C}{\partial r} = 0$$
$$\frac{\partial^2 C}{\partial r^2} = \frac{-7C_{i,N_y} + 8C_{i,N_y} - C_{i,N_y}}{2\Delta x^2} \tag{4.25}$$

Care was taken at grid points where two boundary regions intersect, as both boundary conditions must be applied at said locations. After applying all the boundary conditions, a system of linear equations was generated, and the centration field was solved using the Newton Raphson

method. The calculated concentration field was then used to find the OCR using Eq. (4.7) and (4.8), which was then compared with experimental results.

Chapter 5. NUMERICAL MODELING RESULTS AND ANALYSIS

Coding a CFD program by scratch is both complex and computationally heavy, and therefore requires a coding language that is capable of quickly computing large system of equations with the help of useful built-in functions. MATLAB was chosen specifically for its wide range of built-in functions which are optimized for scientific computing tasks. The CFD code was divided into two core steps: creating the necessary matrix using input variables, and then using the Newton Raphson method. Computation time was mostly spent on solving the system of equations using MATLAB's backslash operator. With $\Delta x = 0.005$, it took roughly 30 seconds to calculate the outflow concentration with 8 iterations of Newton Raphson method. An attempt was made using an iterative method, but the solution converged much slower due to the enormous number of grid points required to iterate over.

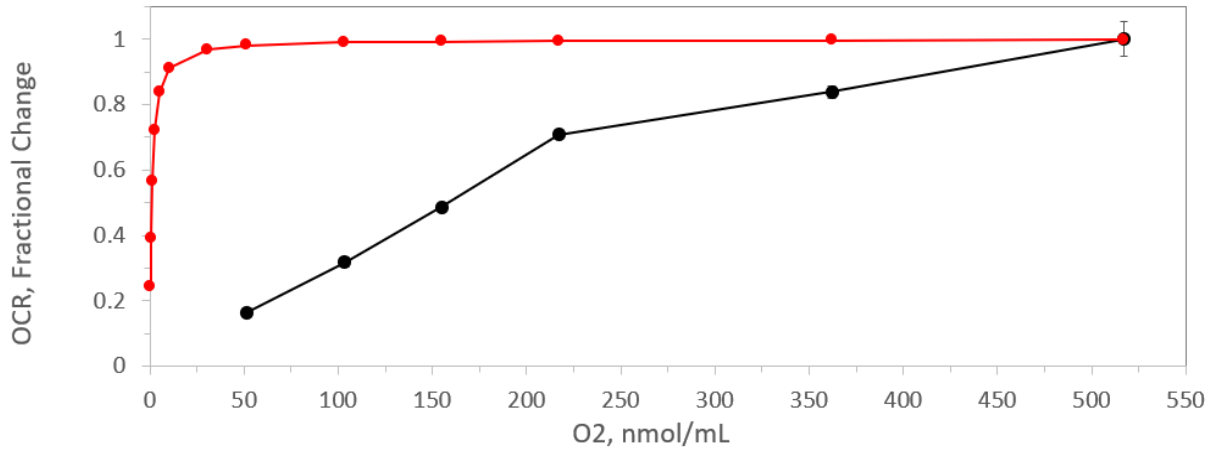
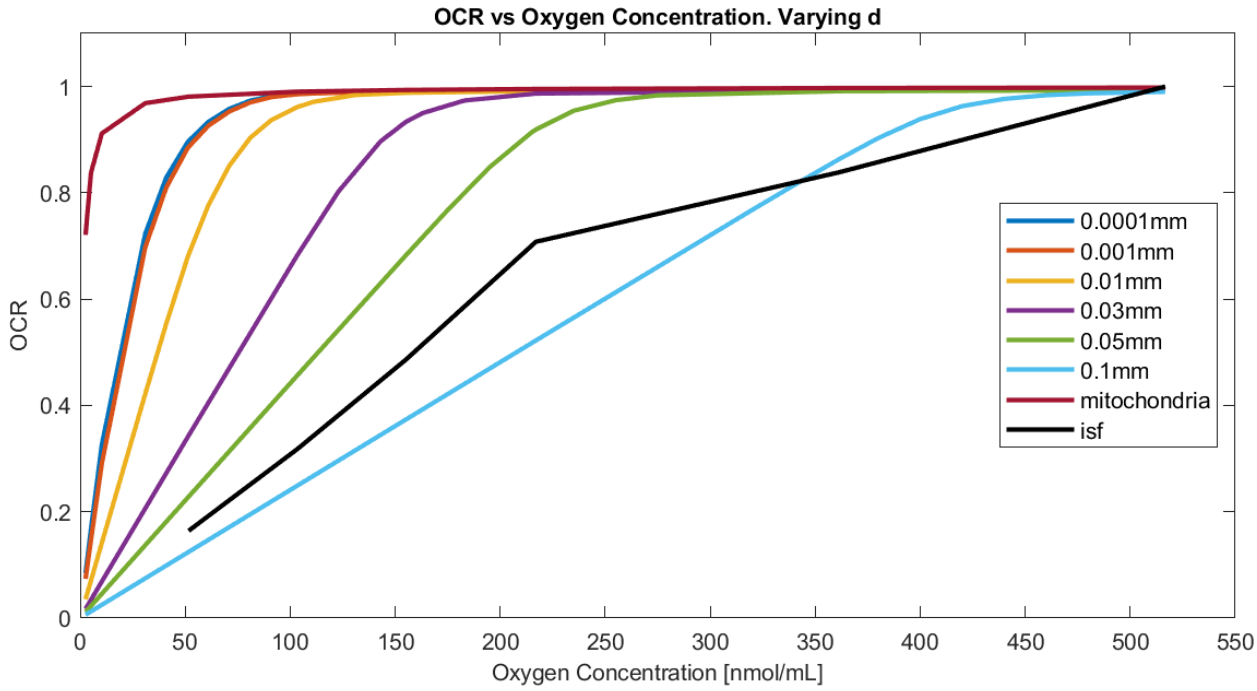


Figure 11. Experimental data for mitochondria only (red line) and ISF layer (black line).

The experimental data used to compare the simulation are shown in Fig. 11. The data set for the red line has a ISF diffusion thickness of zero without flow, while the data set for the black line has a non-zero thickness with flow. The data with ISF layer are too sparse to clearly show the expected asymptotic behavior when normalized correctly, but still illustrates the general idea of how adding the diffusion layer thickness affects the OCR. The mitochondria data also give a reference value for V_{max} and K_m, which does not change between the two lines. By increasing the diffusion thickness d, less oxygen is able to transport to the surface of the tissue due to an increase in diffusion resistance. Thus, the resulting behavior is a shift towards the right in the oxygen consumption rate. On the other hand, as the diffusion thickness approaches zero, oxygen is able to reach the mitochondria without any resistance, resulting in an overall increase to the oxygen consumption (red line). The graphs below show the independent variation of the values d and K_m, and they display the expected behaviors similar to that of the experimental results.



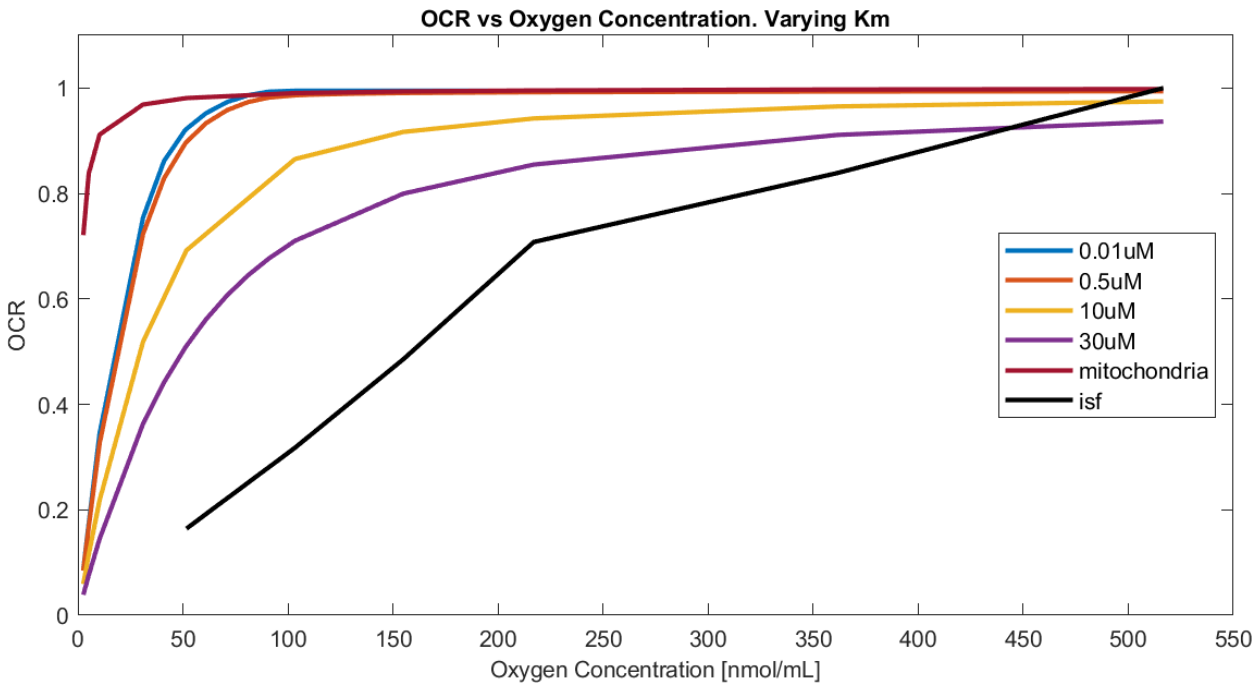


Figure 12. Calculation of OCR through varying d and K_m

An increase in d causes a shift towards the right in the oxygen consumption behavior, which roughly matched the expected behavior from the experimental data. Increasing Km causes the graph to shift downwards, which is also what was expected. However, the effects of changing Km is much less significant compared to that of changing d. Changing a single variable will not yield a significant result, and so it is necessary to observe the effects of changing multiple variables simultaneously.

Chapter 6. **CONCLUSION**

The mass flow controller demonstrated enhanced control and accuracy over the flow rate, maintaining a tolerance approximately within $\pm 0.2 \frac{ml}{min}$ from the setpoint, as shown in Fig. 7. The overshoots observed in the figure may be attributed to inadequate soldering, and it did not pose a significant problem to the experimental data. The analytical solution successfully reduced the time required to achieve steady state, which is a big step towards an improved experimental setup. However, it is essential to test and compare the analytical solution against actual experimental data to validate its reliability.

The 2D simulation includes the effects of surface consumption and the diffusion resistance from the ISF layer, which replicated the OCR behavior from the experimental results. The model successfully implemented the Michaelis Menten equation in conjunction with equations that govern oxygen transport.

6.1 **FUTURE WORK**

If the mass flow controllers exhibit significant noise or overshoots, the outflow concentration may deviate from the solution provided by Eq. (2.5). Thus, the primed mass flow controller should be evaluated in a practical experimental setup to determine its reliability.

Regarding the 2-D flow model, the values for the diffusion thickness was slightly unrealistic, and still require more work regarding its accuracies. The simulation can be improved by introducing permeation terms to represent the gas exchange between various cell layers. The values for V_{max} and K_m should be compared with values obtained by other researchers in the

same field. Additionally, getting a better understanding of the system's geometry would greatly benefit the simulation's realism. This will open up possibilities on using a volumetric consumption to solve for oxygen concentration inside the tissue. By refining the governing equations and gaining a deeper understanding of the system's geometry, the CFD simulation could become capable of replicating, and potentially even replacing, experimental data in the future.

BIBLIOGRAPHY

[1] Michaelis L, Menten ML, Johnson KA, Goody RS. The original Michaelis constant: translation of the 1913 Michaelis-Menten paper. Biochemistry. 2011 Oct 4;50(39):8264-9. doi: 10.1021/bi201284u. Epub 2011 Sep 9. PMID: 21888353; PMCID: PMC3381512.

[2] User's Guide. FMA 5400A/FMA 5500A Mass Flow Controllers. Omega Engineering, Inc. 2019.

[3] Kanji Miyabe, Ryo Isogai, Estimation of molecular diffusivity in liquid phase systems by the Wilke–Chang equation, Journal of Chromatography A, Volume 1218, Issue 38, 2011, Pages 6639-6645, ISSN 0021-9673.

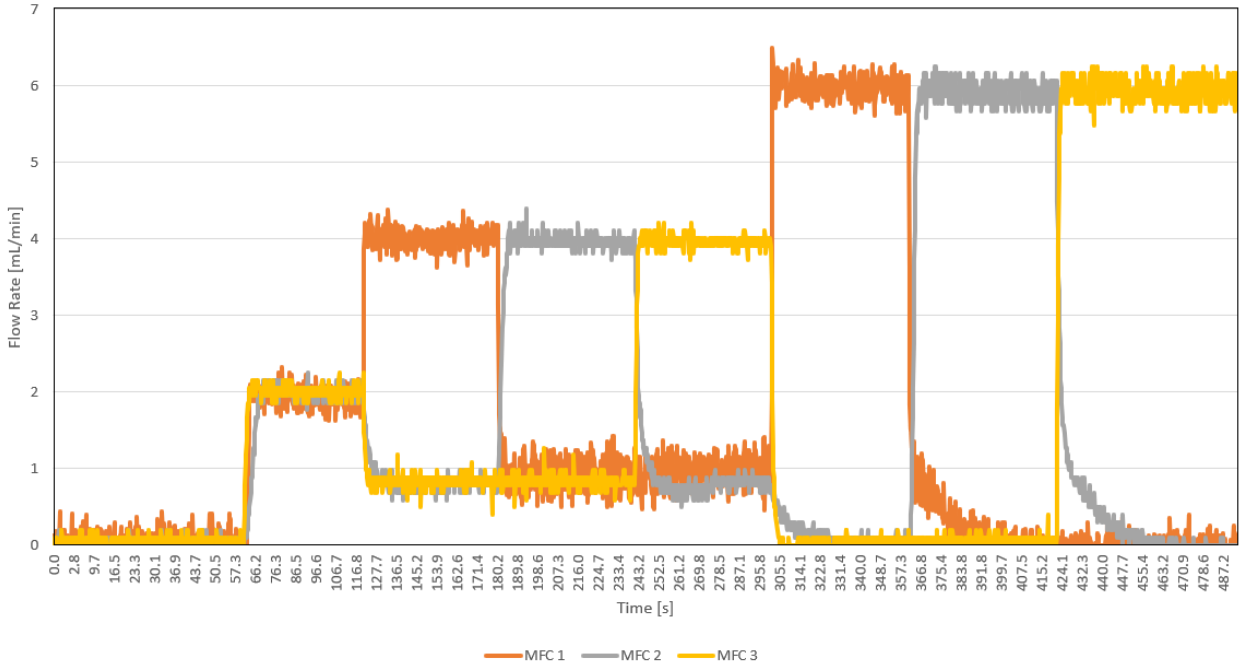
[4] Jeffrey R. Chasnov. *Scientific Computing*, LibreTexts, 2022.

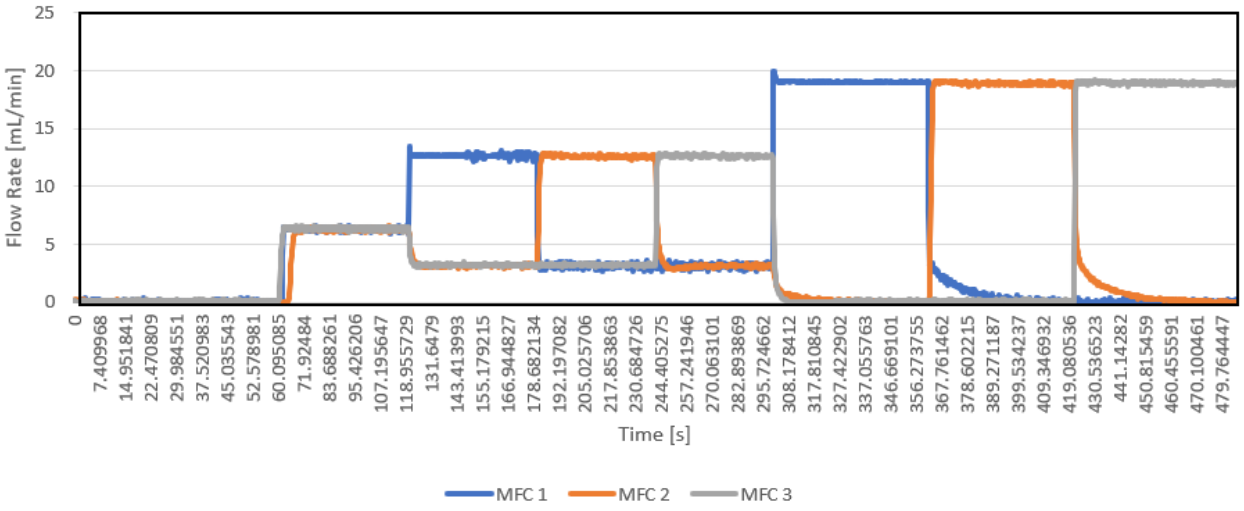
[5] Štigler, Jaroslav. (2012). Analytical Velocity Profile in Tube for Laminar and Turbulent Flow. 10.13140/2.1.3153.5046.

[6] Dash RK, Bassingthwaighte JB. Simultaneous blood-tissue exchange of oxygen, carbon dioxide, bicarbonate, and hydrogen ion. Ann Biomed Eng. 2006 Jul;34(7):1129-48. doi: 10.1007/s10439-005-9066-4. Epub 2006 May 30. PMID: 16775761; PMCID: PMC4232240.

APPENDIX

A.1 ADDITIONAL RESULTS FOR THE MASS FLOW CONTROLLER SETUP





A.2 MATLAB CODE FOR CALCULATING THE 2D VELOCITY FIELD

```
clear all;
% Define input variables
dx = 0.005;                 % [mm] Grid spacing
xmax = 8;                   % [mm] Length of tube
ymax = 1.7;                 % [mm] Radius of tube
Umax = 8.444199*10^(-2);    % [mm/s] Fluid velocity at centerline
tol = 10^(-8);              % Required tolerance for NR method
left_ret = 350 - 1;         % Location of left face of tissue
right_ret = 450 + 1;        % Location of right face of tissue
top_ret = (0.2/dx) + 1 - 1; % Location of top face of tissue
bot_ret = y_size;           % Location of bottom face of tissue

% Define mesh grid and velocity fields (all zeros)
x = 0:dx:xmax;
y = 0:dx:ymax;
x_size = length(x);
y_size = length(y);
lap_size = x_size*y_size;
[X,Y] = meshgrid(x,y);
u = zeros(y_size,x_size);
v = zeros(y_size,x_size);

% Initialize psi field and vorticity fields
y_flipped = flip(y)';
psi_inflow = (Umax/2).*y_flipped.^2 .*(1 - (1/2).*(y_flipped./ymax).^2);
psi = repmat(psi_inflow,1,x_size);
psi(top_ret:bot_ret,left_ret:right_ret) = 0;
psi(end,:) = 0;

w_inflow = (2*Umax/ymax^2).*y_flipped;
w = repmat(w_inflow,1,x_size);
w(top_ret+1:bot_ret,left_ret+1:right_ret-1) = 0;

p_col = reshape(psi',[],1);
w_col  = reshape(w',[],1);
val_col = vertcat(p_col,w_col);
val_col = sparse(val_col);

y_coeff = reshape(Y',[],1);
y_coeff(1:x_size) = 1;
y_coeff = flip(y_coeff);

% Create matrix with appropriate governing equations and boundary conditions
w_lapmat1 = [zeros(x_size,1) zeros(x_size,1) zeros(x_size,1) zeros(x_size,1)
zeros(x_size,1)];
w_lapmat2 = [ones(x_size,1) ones(x_size,1) -4.*ones(x_size,1) ones(x_size,1)
ones(x_size,1)];
w_lapmat2(1,:) = 0;
w_lapmat2(end,:) = [1 0 -2 0 1];
w_lapmat3 = w_lapmat2;
w_lapmat3(left_ret+1:right_ret-1,:) = 0;
```

```
w_lapmat4 = w_lapmat2;
w_lapmat4(left_ret:right_ret,:) = 0;
w_lapfull2 = repmat(w_lapmat2./(dx^2),top_ret-2,1);
w_lapfull3 = w_lapmat3./(dx^2);
w_lapfull4 = repmat(w_lapmat4./(dx^2),y_size-top_ret-1,1);
w_lapfull = vertcat(w_lapmat1,w_lapfull2,w_lapfull3,w_lapfull4,w_lapmat1);
w_lapmat =
spdiags([w_lapfull(:,5),w_lapfull(:,4),w_lapfull(:,3),w_lapfull(:,2),w_lapfull(:,1)],
[-x_size,-1,0,1,x_size],lap_size,lap_size)';

w_lap_addx1 = [zeros(x_size,1) zeros(x_size,1) zeros(x_size,1) zeros(x_size,1)
zeros(x_size,1) zeros(x_size,1) zeros(x_size,1)];
w_lap_addx2 = w_lap_addx1;
w_lap_addx2(end,:) = [-1 4 -5 2 0 0 0];
w_lap_addx_full2 = repmat(w_lap_addx2./(dx^2),y_size-2,1);
w_lap_addx_full = vertcat(w_lap_addx1,w_lap_addx_full2,w_lap_addx1);
w_lap_addx_mat =
spdiags([w_lap_addx_full(:,7),w_lap_addx_full(:,6),w_lap_addx_full(:,5),w_lap_addx_fu
ll(:,4),w_lap_addx_full(:,3),w_lap_addx_full(:,2),w_lap_addx_full(:,1)],[-3,-2,-
1,0,1,2,3],lap_size,lap_size)';

w_y_coeff = zeros(y_size,x_size);
w_y_coeff(2:end-1,2:end) = 1;
w_y_coeff(top_ret+1:bot_ret,left_ret:right_ret) = 0;
w_y_coeff(top_ret,left_ret+1:right_ret-1) = 0;
w_y_coeff = reshape(w_y_coeff',[],1);
w_rsq = (1./y_coeff.^2).*w_y_coeff;
w_rsq_mat = spdiags(w_rsq,0,lap_size,lap_size)';

w_dy_mat1 = [zeros(x_size,1) zeros(x_size,1) zeros(x_size,1)];
w_dy_mat2 = [ones(x_size,1) zeros(x_size,1) -ones(x_size,1)];
w_dy_mat2(1,:) = 0;
w_dy_mat3 = w_dy_mat2;
w_dy_mat3(left_ret+1:right_ret-1,:) = 0;
w_dy_mat4 = w_dy_mat2;
w_dy_mat4(left_ret:right_ret,:) = 0;
w_dy_full2 = repmat(w_dy_mat2./(2*dx),top_ret-2,1);
w_dy_full3 = w_dy_mat3./(2*dx);
w_dy_full4 = repmat(w_dy_mat4./(2*dx),y_size-top_ret-1,1);
w_dy_full = vertcat(w_dy_mat1,w_dy_full2,w_dy_full3,w_dy_full4,w_dy_mat1);
w_dy_full = (1./y_coeff).*w_dy_full;
w_dy_mat = spdiags([w_dy_full(:,3),w_dy_full(:,2),w_dy_full(:,1)],[-
1,0,1].*x_size,lap_size,lap_size)';

w_ybnd1 = [zeros(x_size,1) zeros(x_size,1) 7.*ones(x_size,1) -8.*ones(x_size,1)
ones(x_size,1)];
w_ybnd2 = [zeros(x_size,1) zeros(x_size,1) zeros(x_size,1) zeros(x_size,1)
zeros(x_size,1)];
w_ybnd3 = w_ybnd2;
w_ybnd3(left_ret+1:right_ret-1,:) = repmat([1 -8 0 0 0],right_ret-left_ret-1,1);
w_ybnd_full1 = w_ybnd1./(2*dx^2);
w_ybnd_full2 = repmat(w_ybnd2,top_ret-2,1);
w_ybnd_full3 = w_ybnd3./(2*dx^2);
w_ybnd_full4 = repmat(w_ybnd2,y_size-top_ret,1);
```

```
w_ybnd_full = vertcat(w_ybnd_full1,w_ybnd_full2,w_ybnd_full3,w_ybnd_full4);
w_ybnd_full = (1./y_coeff).*w_ybnd_full;
w_ybnd_mat =
spdiags([w_ybnd_full(:,5),w_ybnd_full(:,4),w_ybnd_full(:,3),w_ybnd_full(:,2),w_ybnd_f
ull(:,1)],[-2,-1,0,1,2].*x_size,lap_size,lap_size)';

w_xbnd1 = [zeros(x_size,1) zeros(x_size,1) zeros(x_size,1) zeros(x_size,1)
zeros(x_size,1)];
w_xbnd2 = w_xbnd1;
w_xbnd2(left_ret,:) = [1 -8 0 0 0];
w_xbnd2(right_ret,:) = [0 0 0 -8 1];
w_xbnd_full1 = repmat(w_xbnd1,top_ret,1);
w_xbnd_full2 = repmat(w_xbnd2./(2*dx^2),y_size-top_ret-1,1);
w_xbnd_full = vertcat(w_xbnd_full1,w_xbnd_full2,w_xbnd1);
w_xbnd_full = (1./y_coeff).*w_xbnd_full;
w_xbnd_mat =
spdiags([w_xbnd_full(:,5),w_xbnd_full(:,4),w_xbnd_full(:,3),w_xbnd_full(:,2),w_xbnd_f
ull(:,1)],[-2,-1,0,1,2],lap_size,lap_size)';

lap_mat1 = [zeros(x_size,1) zeros(x_size,1) zeros(x_size,1) zeros(x_size,1)
zeros(x_size,1)];
lap_mat2 = [ones(x_size,1) ones(x_size,1) -4.*ones(x_size,1) ones(x_size,1)
ones(x_size,1)];
lap_mat2(1,:) = 0;
lap_mat2(end,:) = [1 0 -2 0 1];
lap_mat3 = lap_mat2;
lap_mat3(left_ret:right_ret,:) = 0;
lap_full2 = repmat(lap_mat2./(dx^2),top_ret-2,1);
lap_full3 = repmat(lap_mat3./(dx^2),y_size-top_ret,1);
lap_full = vertcat(lap_mat1,lap_full2,lap_full3,lap_mat1);
lap_full = (1./y_coeff).*lap_full;
lap_mat =
spdiags([lap_full(:,5),lap_full(:,4),lap_full(:,3),lap_full(:,2),lap_full(:,1)],[-
x_size,-1,0,1,x_size],lap_size,lap_size)';

psi_lap_addx1 = [zeros(x_size,1) zeros(x_size,1) zeros(x_size,1) zeros(x_size,1)
zeros(x_size,1) zeros(x_size,1) zeros(x_size,1)];
psi_lap_addx2 = psi_lap_addx1;
psi_lap_addx2(end,:) = [-1 4 -5 2 0 0 0];
psi_lap_addx_full2 = repmat(psi_lap_addx2./(dx^2),y_size-2,1);
psi_lap_addx_full = vertcat(psi_lap_addx1,psi_lap_addx_full2,psi_lap_addx1);
psi_lap_addx_full = (1./y_coeff).*psi_lap_addx_full;
psi_lap_addx_mat =
spdiags([psi_lap_addx_full(:,7),psi_lap_addx_full(:,6),psi_lap_addx_full(:,5),psi_lap
_addx_full(:,4),psi_lap_addx_full(:,3),psi_lap_addx_full(:,2),psi_lap_addx_full(:,1)]
,[-3,-2,-1,0,1,2,3],lap_size,lap_size)';

dy_mat1 = [zeros(x_size,1) zeros(x_size,1) zeros(x_size,1)];
dy_mat2 = [ones(x_size,1) zeros(x_size,1) -ones(x_size,1)];
dy_mat2(1,:) = 0;
dy_mat3 = dy_mat2;
dy_mat3(left_ret:right_ret,:) = 0;
dy_full2 = repmat(dy_mat2./(2*dx),top_ret-2,1);
dy_full3 = repmat(dy_mat3./(2*dx),y_size-top_ret,1);
```

```
dy_full = vertcat(dy_mat1,dy_full2,dy_full3,dy_mat1);
dy_full = (1./y_coeff.^2).*dy_full;
dy_mat = spdiags([dy_full(:,3),dy_full(:,2),dy_full(:,1)],[-
1,0,1].*x_size,lap_size,lap_size)';

psi_id_diag = ones(y_size,x_size);
psi_id_diag(2:end-1,2:end) = 0;
psi_id_diag(top_ret:bot_ret,left_ret:right_ret) = 1;
psi_id_diag = reshape(psi_id_diag',[],1);
psi_id_mat = spdiags(psi_id_diag,0,lap_size,lap_size)';
psi_lapmat = lap_mat - dy_mat + psi_lap_addx_mat + psi_id_mat;

w_id_diag = ones(y_size,x_size);
w_id_diag(2:end-1,2:end) = 0;
w_id_diag(top_ret:bot_ret,left_ret:right_ret) = 1;
w_id_diag = reshape(w_id_diag',[],1);
w_id_mat = spdiags(w_id_diag,0,lap_size,lap_size)';
w_lapmat = w_lapmat + w_dy_mat - w_rsq_mat + w_lap_addx_mat + w_id_mat;

id_diag = zeros(y_size,x_size);
id_diag(2:end-1,2:end) = 1;
id_diag(top_ret:bot_ret,left_ret:right_ret) = 0;
id_diag = reshape(id_diag',[],1);
id_mat = spdiags(id_diag,0,lap_size,lap_size)';

w_bnd_mat = -w_ybnd_mat - w_xbnd_mat;

% Jacobian Matrix
J = [psi_lapmat id_mat; w_bnd_mat w_lapmat];

% Use NR method to find psi and w field
for iter = 1:5
    init_mat = J*val_col;
    psi_init = init_mat(1:lap_size);
    w_init = init_mat(lap_size+1:end);
    psi_init = reshape(psi_init,x_size,y_size)';
    psi_init(:,1) = 0;
    psi_init(1,:) = 0;
    psi_init(top_ret:bot_ret,left_ret:right_ret) = 0;
    psi_init(end,:) = 0;
    w_init = reshape(w_init,x_size,y_size)';
    w_init(end,:) = 0;
    w_init(:,1) = 0;
    psi_init = reshape(psi_init',[],1);
    w_init = reshape(w_init',[],1);
    init_mat = vertcat(psi_init,w_init);
    dval = mldivide(J,-init_mat);
    val_col = val_col + dval;
    max_error = max(abs(dval));
    if max_error < tol
        fprintf('tolerance satisfied after %d iterations\n',iter)
        break
    end
end
```

```
p_col = val_col(1:lap_size);
psi = reshape(p_col,x_size,y_size)';
w_col = val_col(lap_size+1:end);
w = reshape(w_col,x_size,y_size)';

% Solve u and v velocity field using the psi field
for j = 2:x_size
    for i = 2:y_size
        if j == x_size
            v(i,j) = (1/y_flipped(i))*(-1/(2*dx))*(psi(i,j) - psi(i,j-1));
        else
            v(i,j) = (1/y_flipped(i))*(-1/(2*dx))*(psi(i,j+1) - psi(i,j-1));
        end
        if i == y_size
            u(i,j) = (2/dx^2)*(psi(i-1,j));
        else
            u(i,j) = (1/y_flipped(i))*(1/(2*dx))*(psi(i-1,j) - psi(i+1,j));
        end
    end
end

u(top_ret:bot_ret,left_ret:right_ret) = 0;
v(top_ret:bot_ret,left_ret:right_ret) = 0;
u_sol = u(2:end-1,2:end-1);
v_sol = v(2:end-1,2:end-1);
vel_temp1 = sqrt(u_sol.^2 + v_sol.^2);
vel_temp2 = flip(vel_temp1);
vel_sol = cat(1,vel_temp1,vel_temp2);

% Show graphs and figures
close all;
figure(1)
surf(vel_sol)
shading interp
colormap turbo

% Save u and v velocity
save('u_values_0t81.mat','u');
save('v_values_0t81.mat','v');
```

A.3 MATLAB CODE FOR CALCULATING THE 2D CONCENTRATION FIELD

```
clear all;
% NOTE: Make sure to run SolveVelocityField.m first and save the velocity
% field.
% Define input variables
dx = 0.005;                 % [mm] Grid spacing
xmax = 8;                   % [mm] Length of tube
ymax = 1.7;                 % [mm] Radius of tube
d = 0.00001;                % [mm] Thickness of isf layer
D = 3.0277*10^-3;           % [mm^2/s] Diffusion coefficient of oxygen in water
D_isf = 1*10^-4;            % [mm^2/s] Diffusion coefficient of oxygen in isf
Vmax = 0.002;               % [nmol/mm^3s] Maximum reaction rate of cell
Km = 5*10^-4;               % [nmol/mm^3] Michaelis Menten constant
all_conc = [2.17*10^(-1)];  % [nmol/mm^3] Inflow concentration
tol = 10^(-7);              % Required tolerance for NR method
left_ret = 350;             % Location of left face of tissue
right_ret = 450;            % Location of right face of tissue
top_ret = (0.2/dx) + 1;     % Location of top face of tissue
bot_ret = (ymax/dx) + 1;    % Location of bottom face of tissue

% Define constant variables
SA = (2*pi*1.5)*0.5 + 2*(pi/4)*(3)^2;   % [mm^2] Surface area of tissue
Vol = (pi/4)*(0.5)*(3)^2;   % [mm^3] Volume of tissue
V_dot = 0.383333;           % [mm^3/s] Volume flow rate of free stream
A_tube = (pi/4)*(3.4)^2;    % [mm^2] Cross sectional area of tube

% Define mesh grid and load velocity fields
x = 0:dx:xmax;
y = 0:dx:ymax;
x_size = length(x);
y_size = length(y);
lap_size = x_size*y_size;
[X,Y] = meshgrid(x,y);
init_oxy = 0;
u = load('u_values_8l.mat');
v = load('v_values_8l.mat');
u = cell2mat(struct2cell(u));
v = cell2mat(struct2cell(v));

m = length(all_conc);
all_Vmax = zeros(1,m);
all_mass_flow = zeros(1,m);
all_oxy = zeros(lap_size,m);
calculated_mass_flow = zeros(1,m);
calculated_Vmax = zeros(1,m);
res_error = zeros(1,m);

u(top_ret-1:bot_ret,left_ret-1:right_ret+1) = 0;
v(top_ret-1:bot_ret,left_ret-1:right_ret+1) = 0;
v(end,:) = 0;
```

```
u_col = reshape(u',[],1);
v_col  = reshape(v',[],1);

% Create matrix with appropriate governing equations and boundary conditions
dx_mat1 = [zeros(x_size,1) -ones(x_size,1) zeros(x_size,1) ones(x_size,1)
zeros(x_size,1)];
dx_mat1(1,:) = 0;
dx_mat1(end,:) = 0;
dx_mat2 = dx_mat1;
dx_mat2(left_ret:right_ret,:) = 0;
dx_mat3 = dx_mat1;
dx_mat3(left_ret-1:right_ret+1,:) = 0;
dx_full1 = repmat(dx_mat1./(2*dx),top_ret-2,1);
dx_full2 = dx_mat2./(2*dx);
dx_full3 = repmat(dx_mat3./(2*dx),y_size-top_ret+1,1);
dx_full = vertcat(dx_full1,dx_full2,dx_full3);
dx_full = u_col.*dx_full;
dx_mat =
spdiags([dx_full(:,5),dx_full(:,4),dx_full(:,3),dx_full(:,2),dx_full(:,1)],[-2,-
1,0,1,2],lap_size,lap_size)';

dy_mat1 = [zeros(x_size,1) zeros(x_size,1) zeros(x_size,1) zeros(x_size,1)
zeros(x_size,1)];
dy_mat2 = [zeros(x_size,1) zeros(x_size,1) ones(x_size,1) -ones(x_size,1)
zeros(x_size,1)];
dy_mat2(1,:) = 0;
dy_mat3 = [zeros(x_size,1) ones(x_size,1) zeros(x_size,1) -ones(x_size,1)
zeros(x_size,1)];
dy_mat3(1,:) = 0;
dy_mat4 = dy_mat3;
dy_mat4(left_ret:right_ret,:) = 0;
dy_mat5 = dy_mat3;
dy_mat5(left_ret-1:right_ret+1,:) = 0;
dy_full2 = dy_mat2./(2*dx);
dy_full3 = repmat(dy_mat3./(2*dx),top_ret-3,1);
dy_full4 = dy_mat4./(2*dx);
dy_full5 = repmat(dy_mat5./(2*dx),y_size-top_ret,1);
dy_full_temp = vertcat(dy_full2,dy_full3,dy_full4,dy_full5,dy_mat1);
dy_full = v_col.*dy_full_temp;
dy_mat =
spdiags([dy_full(:,5),dy_full(:,4),dy_full(:,3),dy_full(:,2),dy_full(:,1)],[-2,-
1,0,1,2].*x_size,lap_size,lap_size)';

y_coeff = reshape(Y',[],1);
y_coeff(1:x_size) = 1;
y_coeff = flip(y_coeff);
dy_diff = (D./y_coeff).*dy_full_temp;
dy_diff_mat =
spdiags([dy_diff(:,5),dy_diff(:,4),dy_diff(:,3),dy_diff(:,2),dy_diff(:,1)],[-2,-
1,0,1,2].*x_size,lap_size,lap_size)';

lap_mat2 = [zeros(x_size,1) ones(x_size,1) -2.*ones(x_size,1) ones(x_size,1)
zeros(x_size,1)];
lap_mat2(1,:) = 0;
```

```
lap_mat2(end,:) = 0;
lap_mat3 = [ones(x_size,1) ones(x_size,1) -4.*ones(x_size,1) ones(x_size,1)
ones(x_size,1)];
lap_mat3(1,:) = 0;
lap_mat3(end,:) = [1 0 -2 0 1];
lap_mat4 = lap_mat3;
lap_mat4(left_ret:right_ret,:) = 0;
lap_mat5 = lap_mat3;
lap_mat5(left_ret-1:right_ret+1,:) = 0;
lap_mat6 = [4.*ones(x_size,1) ones(x_size,1) -6.*ones(x_size,1) ones(x_size,1)
zeros(x_size,1)];
lap_mat6(1,:) = 0;
lap_mat6(end,:) = [4 0 -4 0 0];
lap_mat6(left_ret-1:right_ret+1,:) = 0;
lap_full2 = lap_mat2./(dx^2);
lap_full3 = repmat(lap_mat3./(dx^2),top_ret-3,1);
lap_full4 = lap_mat4./(dx^2);
lap_full5 = repmat(lap_mat5./(dx^2),y_size-top_ret,1);
lap_full6 = lap_mat6./(dx^2);
lap_full = D.*vertcat(lap_full2,lap_full3,lap_full4,lap_full5,lap_full6);
lap_mat =
spdiags([lap_full(:,5),lap_full(:,4),lap_full(:,3),lap_full(:,2),lap_full(:,1)],[-
x_size,-1,0,1,x_size],lap_size,lap_size)';

lap_add_dy1 = [zeros(x_size,1) zeros(x_size,1) zeros(x_size,1) zeros(x_size,1)
zeros(x_size,1)];
lap_add_dy2 = lap_add_dy1;
lap_add_dy2(left_ret:right_ret,:) = repmat((D/(2*dx)).*[-1 4 -3 0 0],right_ret-
left_ret+1,1);
lap_add_dy2(left_ret:right_ret,:) = lap_add_dy2(left_ret:right_ret,:) +
repmat((D_isf/d).*[0 0 -1 1 0],right_ret-left_ret+1,1);
lap_add_dy_full1 = repmat(lap_add_dy1,top_ret-2,1);
lap_add_dy_full2 = lap_add_dy2;
lap_add_dy_full3 = repmat(lap_add_dy1,y_size-top_ret+1,1);
lap_add_dy_full = vertcat(lap_add_dy_full1,lap_add_dy_full2,lap_add_dy_full3);
lap_add_dy_mat =
spdiags([lap_add_dy_full(:,5),lap_add_dy_full(:,4),lap_add_dy_full(:,3),lap_add_dy_fu
ll(:,2),lap_add_dy_full(:,1)],[-2,-1,0,1,2].*x_size,lap_size,lap_size)';

lap_add_dx1 = [zeros(x_size,1) zeros(x_size,1) zeros(x_size,1) zeros(x_size,1)
zeros(x_size,1)];
lap_add_dx2 = [zeros(x_size,1) zeros(x_size,1) zeros(x_size,1) zeros(x_size,1)
zeros(x_size,1)];
lap_add_dx2(left_ret-1,:) = (D/(2*dx)).*[-1 4 -3 0 0];
lap_add_dx2(left_ret-1,:) = lap_add_dx2(left_ret-1,:) + (D_isf/d).*[0 0 -1 1 0];
lap_add_dx2(right_ret+1,:) = (D/(2*dx)).*[0 0 -3 4 -1];
lap_add_dx2(right_ret+1,:) = lap_add_dx2(right_ret+1,:) + (D_isf/d).*[0 1 -1 0 0];
lap_add_dx_full1 = repmat(lap_add_dx1,top_ret-1,1);
lap_add_dx_full2 = repmat(lap_add_dx2,y_size-top_ret+1,1);
lap_add_dx_full = vertcat(lap_add_dx_full1,lap_add_dx_full2);
lap_add_dx_mat =
spdiags([lap_add_dx_full(:,5),lap_add_dx_full(:,4),lap_add_dx_full(:,3),lap_add_dx_fu
ll(:,2),lap_add_dx_full(:,1)],[-2,-1,0,1,2],lap_size,lap_size)';
```

```
dx_cell_mat1 = [zeros(x_size,1) zeros(x_size,1) zeros(x_size,1)];
dx_cell_mat2 = dx_cell_mat1;
dx_cell_mat2(left_ret,:) = [1 -1 0];
dx_cell_mat2(right_ret,:) = [0 -1 1];
dx_cell_full1 = repmat(dx_cell_mat1,top_ret-1,1);
dx_cell_full2 = repmat(dx_cell_mat2,y_size-top_ret+1,1);
dx_cell_full = (D_isf*SA/(d*Vol)).*vertcat(dx_cell_full1,dx_cell_full2);
dx_cell_mat = spdiags([dx_cell_full(:,3),dx_cell_full(:,2),dx_cell_full(:,1)],[-
1,0,1],lap_size,lap_size)';

dy_cell_mat1 = [zeros(x_size,1) zeros(x_size,1) zeros(x_size,1)];
dy_cell_mat2 = dy_cell_mat1;
dy_cell_mat2(left_ret+1:right_ret-1,:) = repmat([1 -1 0],right_ret-left_ret-1,1);
dy_cell_full1 = repmat(dy_cell_mat1,top_ret-1,1);
dy_cell_full2 = dy_cell_mat2;
dy_cell_full3 = repmat(dy_cell_mat1,y_size-top_ret,1);
dy_cell_full =
(D_isf*SA/(d*Vol)).*vertcat(dy_cell_full1,dy_cell_full2,dy_cell_full3);
dy_cell_mat = spdiags([dy_cell_full(:,3),dy_cell_full(:,2),dy_cell_full(:,1)],[-
1,0,1].*x_size,lap_size,lap_size)';

lap_end_dx1 = [zeros(x_size,1) zeros(x_size,1) zeros(x_size,1) zeros(x_size,1)
zeros(x_size,1) zeros(x_size,1) zeros(x_size,1)];
lap_end_dx1(end,:) = [-1 4 -5 2 0 0 0];
lap_end_dx_full = D.*repmat(lap_end_dx1./(dx^2),y_size,1);
lap_end_dx_mat =
spdiags([lap_end_dx_full(:,7),lap_end_dx_full(:,6),lap_end_dx_full(:,5),lap_end_dx_fu
ll(:,4),lap_end_dx_full(:,3),lap_end_dx_full(:,2),lap_end_dx_full(:,1)],[-3,-2,-
1,0,1,2,3],lap_size,lap_size)';

lap_wall_dy1 = [zeros(x_size,1) zeros(x_size,1) zeros(x_size,1) -7.*ones(x_size,1)
8.*ones(x_size,1) -ones(x_size,1) zeros(x_size,1)];
lap_wall_dy2 = [zeros(x_size,1) zeros(x_size,1) zeros(x_size,1) zeros(x_size,1)
zeros(x_size,1) zeros(x_size,1) zeros(x_size,1)];
lap_wall_dy_full1 = lap_wall_dy1./(2*dx^2);
lap_wall_dy_full2 = repmat(lap_wall_dy2,y_size-1,1);
lap_wall_dy_full = D.*vertcat(lap_wall_dy_full1,lap_wall_dy_full2);
lap_wall_dy_mat =
spdiags([lap_wall_dy_full(:,7),lap_wall_dy_full(:,6),lap_wall_dy_full(:,5),lap_wall_d
y_full(:,4),lap_wall_dy_full(:,3),lap_wall_dy_full(:,2),lap_wall_dy_full(:,1)],[-3,-
2,-1,0,1,2,3].*x_size,lap_size,lap_size)';

diag_full = ones(y_size,x_size);
diag_full(:,2:end) = 0;
diag_full(top_ret+1:bot_ret,left_ret+1:right_ret-1) = 1;
diag_full = reshape(diag_full',[],1);
diag_mat = spdiags(diag_full,0,lap_size,lap_size);

lap_mat = lap_mat + lap_add_dx_mat + lap_add_dy_mat + lap_end_dx_mat +
lap_wall_dy_mat;
cell_mat = dx_cell_mat + dy_cell_mat;

% Create Jacobian matrix
J =  lap_mat + dy_diff_mat - dx_mat - dy_mat + cell_mat + diag_mat;
```

```
r_height = flip(y);
r_tube = r_height';
r_top = 1.5;
r_cell = r_tube(top_ret:end);

loc_in = 2;
loc_out = x_size;
u_out = u(:,loc_out);
u_in = u(:,loc_in);
r_u_in = r_tube.*u_in;
r_u_out = r_tube.*u_out;

% Use NR method to find concentration field
for num = 1:m
    oxy = init_oxy.*ones(y_size,x_size);
    oxy(top_ret+1:bot_ret,left_ret+1:right_ret-1) = 0;
    oxy(:,1) = all_conc(num);
    oxy_disk = oxy(top_ret:bot_ret,left_ret:right_ret);
    oxy_col = reshape(oxy',[],1);
    for iter = 1:10
        [J_cell,F_cell] = get_J_F_cell(oxy_disk, Vmax, Km, x_size,y_size, top_ret,
bot_ret, left_ret, right_ret);
        J_mat = J + J_cell;
        F_mat = J*oxy_col + F_cell;
        F_mat = reshape(F_mat,x_size,y_size)';
        F_mat(:,1) = 0;
        F_mat(top_ret+1:bot_ret,left_ret+1:right_ret-1) = 0;
        F_mat = reshape(F_mat',[],1);
        doxy = mldivide(J_mat,-F_mat);
        oxy_col = oxy_col + doxy;
        oxy = reshape(oxy_col,x_size,y_size)';
        oxy_disk = oxy(top_ret:bot_ret,left_ret:right_ret);
        max_error = max(abs(doxy));
        if max_error < tol
            fprintf('tolerance satisfied after %d iterations for inflow concentration
of %.5f\n',iter,all_conc(num))
            break
        end
    end
    res_error(num) = max_error;
    conc_summation = (2*pi*dx).*r_tube.*oxy(:,end);
    all_oxy(:,num) = oxy_col;
    vel_summation = (2*pi*dx).*r_tube.*u(:,end).*oxy(:,end);
    [all_mass_flow(num),all_Vmax(num)] =
solve_surf_consumption(oxy_disk,dx,Vmax,Km,Vol,SA,r_cell);
    sum_mass_in = (2*pi*dx)*sum(r_u_in.*oxy(:,loc_in));
    sum_mass_out = (2*pi*dx)*sum(r_u_out.*oxy(:,loc_out));
    calculated_mass_flow(num) = sum_mass_in - sum_mass_out;
    calculated_Vmax(num) = calculated_mass_flow(num)/(Vol);
end
fprintf('Following results are for d = %.5f, Vmax = %.5f, Km = %.4f, D_isf
= %0.5f\n',d,Vmax,Km,D_isf)
fprintf('Normalized OCR (calculated using MM equation):\n')
```

```
fprintf('           %.15f',all_Vmax./Vmax)
fprintf('\n')
fprintf('Normalized OCR (calculated using inflow and outflow concentration):\n')
fprintf('           %.15f',calculated_Vmax./Vmax)
fprintf('\n')

% Show graphs and figures
close all;
oxy = reshape(oxy_col,x_size,y_size);
oxy = oxy';
oxy_sol1 = oxy(2:end-1,2:end-1);
oxy_sol2 = flip(oxy_sol1);
oxy_sol = cat(1,oxy_sol1,oxy_sol2);
figure(1)
surf(oxy_sol)
shading interp
colormap turbo

% Function for solving the matrix for the MM equation
function [J_cell,F_cell] = get_J_F_cell(oxy_disk, Vmax, Km, x_size,y_size, top_ret,
bot_ret, left_ret, right_ret)
    J_oxy_temp = zeros(y_size,x_size);
    F_oxy_temp = zeros(y_size,x_size);
    J_cell_temp = -(Vmax*Km)./((Km + oxy_disk).^2);
    F_cell_temp = -(Vmax.*oxy_disk)./(Km + oxy_disk);
    J_oxy_temp(top_ret:bot_ret,left_ret:right_ret) = J_cell_temp;
    J_oxy_temp(top_ret+1:bot_ret,left_ret+1:right_ret-1) = 0;
    F_oxy_temp(top_ret:bot_ret,left_ret:right_ret) = F_cell_temp;
    F_oxy_temp(top_ret+1:bot_ret,left_ret+1:right_ret-1) = 0;
    F_oxy_temp(top_ret,left_ret) = 0;
    F_oxy_temp(top_ret,right_ret) = 0;
    F_oxy_temp(bot_ret,left_ret) = 0;
    F_oxy_temp(bot_ret,right_ret) = 0;
    J_oxy_temp(top_ret,left_ret) = 0;
    J_oxy_temp(top_ret,right_ret) = 0;
    J_oxy_temp(bot_ret,left_ret) = 0;
    J_oxy_temp(bot_ret,right_ret) = 0;
    J_oxy_temp = reshape(J_oxy_temp',[],1);
    J_cell = spdiags(J_oxy_temp,0,x_size*y_size,x_size*y_size);
    F_cell = reshape(F_oxy_temp',[],1);
end

% Function for solving the oxygen consumption rate on the surface of cell
function [tot_mass_flow,tot_consumption] = solve_surf_consumption(oxy_disk, dx, Vmax,
Km, Vol, SA, r_cell)
    oxy_temp = (Vmax.*oxy_disk)./(Km + oxy_disk);
    oxy_temp = (2*pi*dx).*r_cell.*oxy_temp;
    oxy_temp(1,1) = 0;
    oxy_temp(end,1) = 0;
    oxy_temp(end,end) = 0;
    oxy_temp(1,end) = 0;
    oxy_temp2 = oxy_temp.*(Vol/SA);
    oxy_temp2(2:end,2:end-1) = 0;
    tot_mass_flow = sum(sum(oxy_temp2));
```

```
    oxy_temp = oxy_temp2./(Vol);
    tot_consumption = sum(sum(oxy_temp));
end
```