MAE 561: Computational Fluid Dynamics

Professor Marcus Herrmann

Final Project Report

Chandler Hutchens

1216384494

Arizona State University

Table of Contents

ABSTRACT

For this investigation, the objective was to create a simulation of a mixing chamber by modeling the continuity and Navier-Stokes equations. Beginning with the problem statement, moving on to the governing equations used, an analysis of the numerical methods, and finally ending with the results and discussion. Using various MATLAB functions generated throughout this class, plots for the u velocity, v velocity, and mass fraction (Y) were generated, along with a video of the ten-second simulation. After completion of the many numerical methods, the result yields two additional graphs of the tangential force vs time, and the chemical species leaving the chamber vs time. Solution verification was conducted on the parameters of average tangential force and total chemical species leaving the chamber using the Richardson Extrapolation and Grid Conversion Index. Reviewing the results, it was found that the observed order was two, as expected for Task 1, and falls within the accepted values to be in the asymptotic region.

NOMENCLATURE

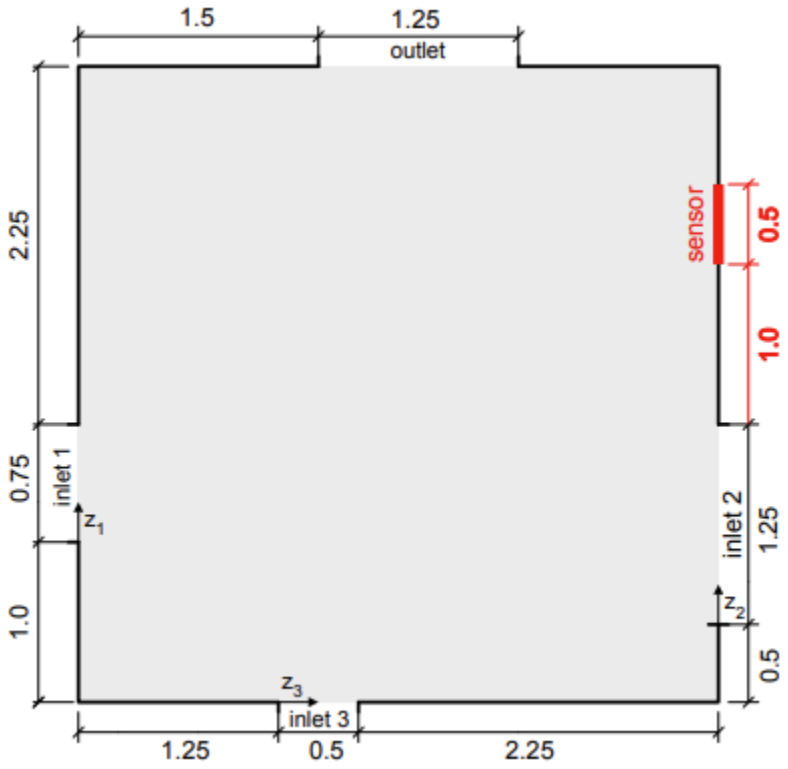
$\overline{F}$ = Average Tangential Force
$A(t)$ = Chemical Species Leaving Chamber
CFL = Courant–Friedrichs–Lewy
ρ = Density
GCI = Grid Conversion Index
Y = Mass Fraction
P = Pressure
Sc = Schmidt Number
Q = Source Term
Re = Reynolds Number
$F(t)$ = Tangential Force
B = Total Chemical Species Leaving Chamber
u = Velocity in the X-Direction
v = Velocity in the Y-Direction
υ = Viscosity

1. Introduction

1.1 Project Description

This project outlines a mixing chamber, shown in Figure 1-1-1, which has a non-dimensionalized length of X and Y both equal to four. Using the Navier-Stokes Equations, solve for the velocity in the u and v, the mass fraction (Y) at various times from zero to ten. Additionally, calculate the time evolution of the tangential force (F(t)) and the chemical species leaving the chamber (A(t)). Finally, calculate the values of the average tangential force ($\overline{F}$) and the total chemical species that left the chamber (B) with a given percentage of error and a solution verification table.

Figure 1-1-1 Mixing Chamber



1.2 Report Outline

This final project report will be organized into the following sections: Introduction (1), Governing Equations (2), Numerical Methods (3), Results: Task 1 (4), Results: Task 2 (5), Conclusion (6), and Appendix (7).

2. Governing Equations

2.1 Continuity Equation

When beginning to solve this mixing chamber problem, there are a variety of equations that will be needed to generate the required velocities, mass fractions, etc. Starting with Equation 1, the continuity equation can be seen which represents the conservation of mass. The continuity equation is shown here in vector form.

$$\nabla \cdot \hat{v} = 0 \tag{1}$$

2.2 X and Y Momentum Equation

Second to the continuity equation, the Navier-Stokes equation represents the conservation of momentum, shown in Equation 2. This also is shown in vector form. However, this form is not particularly useful for solving using finite differences, and therefore shown below is the transformed two-dimensional partial differential equation form. Equation 4 represents the X-momentum and Equation 5 represents the Y-momentum. It should be noted that the density and viscosity have been traded for the Reynolds number and that these equations are in conservative form.

$$\frac{\partial \hat{v}}{\partial t} + \nabla \cdot \left(\widehat{\widehat{vv}}\right) = -\frac{1}{\rho}\nabla p + \nu \nabla^2 \hat{v} \tag{2}$$

$$\frac{\partial u}{\partial x} + \frac{\partial v}{\partial y} = 0 \tag{3}$$

$$\frac{\partial u}{\partial t} + \frac{\partial uu}{\partial x} + \frac{\partial uv}{\partial y} = -\frac{\partial p}{\partial x} + \frac{1}{Re}\left(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2}\right) + Q_u(x, y, t) \tag{4}$$

$$\frac{\partial v}{\partial t} + \frac{\partial uv}{\partial x} + \frac{\partial vv}{\partial y} = -\frac{\partial p}{\partial y} + \frac{1}{Re}\left(\frac{\partial^2 v}{\partial x^2} + \frac{\partial^2 v}{\partial y^2}\right) + Q_v(x, y, t) \tag{5}$$

2.3 Mass Fraction Equation

Similar to the X and Y momentum equations, Equation 6 details a transport equation which models the chemical species, also in the conservative form.

$$\frac{\partial u}{\partial t} + u\frac{\partial Y}{\partial x} + v\frac{\partial Y}{\partial y} = \frac{1}{Re\,SC}\left(\frac{\partial^2 Y}{\partial x^2} + \frac{\partial^2 Y}{\partial y^2}\right) + Q_Y(x, y, t) \tag{6}$$

2.4 Inlet Boundary Conditions

Looking at the given equations in sections 2.1 to 2.3, they can only offer a limited solution without boundary conditions. Looking at Figure 1-1-1, the mixing chamber has three inlets, with each of these having a given x-velocity, y-velocity, and mass fraction. Inlet one is located on the left boundary of the chamber and the boundary conditions are shown in Equations 7, 8, and 9.

$$u_{in,1}\left(z_1,\ t\right) = U_1(t)\ f\left(z_1\right)\ cos\left(\alpha_1(t)\right) \tag{7}$$

$$v_{in,1}\left(z_1,\ t\right) = U_1(t)\ f\left(z_1\right)\ cos\left(\alpha_1(t)\right) \tag{8}$$

$$Y_{in,1}\left(z_1,\ t\right) = f_{Y_1}(t) \tag{9}$$

Inlet two is located on the right boundary of the chamber and the boundary conditions are shown in Equations 10, 11, and 12.

$$u_{in,2}\left(z_1,\ t\right) = U_2(t)\ f\left(z_2\right)\ cos\left(\alpha_2(t)\right) \tag{10}$$

$$v_{in,2}\left(z_2,\ t\right) = U_2(t)\ f\left(z_2\right)\ cos\left(\alpha_2(t)\right) \tag{11}$$

$$Y_{in,2}\left(z_2,\ t\right) = f_{Y_2}(t) \tag{12}$$

Inlet three is located on the bottom boundary of the chamber and the boundary conditions are shown in Equations 13, 14, and 15.

$$u_{in,3}\left(z_3,\ t\right) = U_3(t)\ f\left(z_3\right)\ cos\left(\alpha_3(t)\right) \tag{13}$$

$$v_{in,3}\left(z_3,\ t\right) = U_3(t)\ f\left(z_3\right)\ cos\left(\alpha_3(t)\right) \tag{14}$$

$$Y_{in,3}\left(z_3,\ t\right) = f_{Y_3}(t) \tag{15}$$

Where Equations 7 through 15 have embedded variables which are detailed in Equations 16 through 25.

$$f(z) = 6z(1 - z) \tag{16}$$

$$\alpha_1(t) = \frac{\pi}{3} sin(\pi t) \tag{17}$$

$$\alpha_2(t) = \frac{5\pi}{6} + \frac{\pi}{4} sin\left(\frac{3\pi}{5} t\right) \tag{18}$$

$$\alpha_3(t) = \frac{2\pi}{5} + \frac{\pi}{3} sin\left(\frac{2\pi}{5} t\right) \tag{19}$$

$$U_1(t) = \left(1 + sin\left(\frac{2\pi}{3} t + \frac{\pi}{4}\right)\right) \tag{20}$$

$$U_2(t) = \frac{13}{20}\left(1 + sin\left(\frac{2\pi}{5} t - \frac{\pi}{4}\right)\right) \tag{21}$$

$$U_3(t) = \frac{3}{2}\left(1 + sin\left(\frac{3\pi}{4} t + \frac{\pi}{5}\right)\right) \tag{22}$$

$$f_{Y_1}(t) = \begin{cases} 1, & \text{if mod}(t, 4) < 2 \\ 0, & \text{otherwise} \end{cases} \tag{23}$$

$$f_{Y_2}(t) = \begin{cases} 1, & \text{if mod}(t, 3) < 2 \\ 0, & \text{otherwise} \end{cases} \tag{24}$$

$$f_{Y_3}(t) = \begin{cases} 0, & \text{if mod}(t, 2) < 1.5 \\ 1, & \text{otherwise} \end{cases} \tag{25}$$

2.5 Outlet Boundary Conditions

Similar to the inlet, looking at Figure 1-1-1, the mixing chamber has one outlet with a given x-velocity, y-velocity, and mass fraction. Equations 26, 2 zero-Neumman boundary conditions.

$$\frac{\partial u}{\partial y}\Big|_{out} = 0 \tag{26}$$

$$\frac{\partial v}{\partial y}\Big|_{out} = 0 \tag{27}$$

$$\frac{\partial Y}{\partial y}\Big|_{out} = 0 \tag{28}$$

2.6 Wall Boundary Conditions

Moving to the walls, Equations 29 and 30 are no-slip conditions and Equation 31 states there are no fluxes of chemical species.

$$u_w = 0 \tag{29}$$

$$v_w = 0 \tag{30}$$

$$\frac{\partial Y}{\partial n}\Big|_w = 0 \tag{31}$$

2.7 F(t) Equation

When solving this mixing chamber there are a few calculations which are required to gain useful information. The first is the tangential force exerted by the flow on the sensor shown in Equation 32. To calculate this, the shear stress for the wall is needed and can be calculated in Equation 33.

$$F(t) = \int \tau_w (y, t)\, dy \tag{32}$$

$$\tau_w = -\frac{1}{Re}\frac{\partial v}{\partial x} \tag{33}$$

2.8 $\overline{F}$ Equation

After integrating Equation 32, shown in Equation 34, the average tangential force from 0 to 10 is shown.

$$\overline{F} = \frac{1}{10}\int_0^{10} F(t)\,dt \tag{34}$$

2.9 A(t) Equation

An additional calculation is to determine the amount of chemical species leaving the chamber at a given time. This can be seen in

$$A(t) = \int v\,Y\,dx \tag{35}$$

2.10 B Equation

Similar to the average tangential force, the total amount of chemical species having left the chamber can be seen in Equation 36 from integrating Equation 35.

$$B = \int_0^{10} A(t)\,dt \tag{36}$$

3. Numerical Methods

3.1 Navier-Stokes Overview

The Navier-Stokes equation can be classified as three different types of partial differential equations (PDEs): elliptical, parabolic, and hyperbolic. Each PDE is solved separately and combined together to create the full Navier-Stokes solution. In the end, the viscous terms in each PDE need to be solved, along with the convective teams of the momentum equations shown in Equations 4 and 5, the Poisson terms in those momentum equations, and the convective and diffusive terms in the mass fraction equation in Equation 6.

3.2 Viscous Terms (Parabolic)

Various methods are used and combined to solve the viscous terms. The overarching method is known as the Crank-Nicholson method, which is implicit and created by using the central differences to an intermediate time step of ½. To solve Crank-Nicholson, the use of the alternating direction implicit (ADI) method is needed. This takes an initial time step and solves in the X-direction and then takes another time step and solves in the Y-direction, reaching a full-time step! Additionally, since this is fundamentally from the central difference method, Crank-Nicholson is a second-order method in both time and space. Finally, it should be noted that this system is tridiagonal and can be solved with Gaussian elimination. Now, the Crank-Nicolson method is shown below in Equations 37 through 44. Equation 37 details step 1 for the u-velocity, while Equation 38 shows step 2 for the u-velocity.

$$\begin{aligned} &- d_1 u^{n+\frac{1}{2}}_{i+\frac{3}{2},j} + (1 + 2\,d_1)\, u^{n+\frac{1}{2}}_{i+\frac{1}{2},j} - d_1 u^{n+\frac{1}{2}}_{i-\frac{1}{2},j} \\ &= d_2 u^{n}_{i+\frac{1}{2},j+1} + (1 - 2\,d_2)\, u^{n}_{i+\frac{1}{2},j} + d_2\, u^{n}_{i+\frac{1}{2},j-1} \\ &\quad + \frac{dt}{2} Q_u\left(x_{i+\frac{1}{2}},\, y_j,\, t_n\right) \end{aligned} \tag{37}$$

$$\begin{aligned} &- d_2 u^{n+1}_{i+\frac{1}{2},j+1} + (1 + 2\,d_2)\, u^{n+1}_{i+\frac{1}{2},j} - d_2 u^{n+1}_{i+\frac{1}{2},j-1} \\ &= d_1 u^{n+\frac{1}{2}}_{i+\frac{3}{2},j} + (1 - 2\,d_1)\, u^{n+\frac{1}{2}}_{i+\frac{1}{2},j} + d_1\, u^{n+\frac{1}{2}}_{i-\frac{1}{2},j} \\ &\quad + \frac{dt}{2} Q_u\left(x_{i+\frac{1}{2}},\, y_j,\, t_{n+\frac{1}{2}}\right) \end{aligned} \tag{38}$$

Equation 39 details step 1 for the v-velocity, while Equation 40 shows step 2 for the v-velocity.

$$\begin{aligned} &- d_1 v^{n+\frac{1}{2}}_{i+1,j+\frac{1}{2}} + (1 + 2d_1) v^{n+\frac{1}{2}}_{i,j+\frac{1}{2}} - d_1 v^{n+\frac{1}{2}}_{i-1,j+\frac{1}{2}} \\ &= d_2 v^{n}_{i,j+\frac{3}{2}} + (1 - 2d_2) v^{n}_{i,j+\frac{1}{2}} + d_2 v^{n}_{i,j-\frac{1}{2}} \\ &\quad + \frac{dt}{2} Q_v\left(x_i, y_{j+\frac{1}{2}}, t_n\right) \end{aligned} \tag{39}$$

$$\begin{aligned} &- d_2 v^{n+1}_{i,j+\frac{3}{2}} + (1 + 2d_2) v^{n+1}_{i,j+\frac{1}{2}} - d_2 v^{n+1}_{i,j-\frac{1}{2}} \\ &= d_1 v^{n+\frac{1}{2}}_{i+1,j+\frac{3}{2}} + (1 - 2d_1) v^{n+\frac{1}{2}}_{i,j+\frac{1}{2}} + d_1 v^{n}_{i-1,j+\frac{1}{2}} \\ &\quad + \frac{dt}{2} Q_v\left(x_i, y_{j+\frac{1}{2}}, t_{n+\frac{1}{2}}\right) \end{aligned} \tag{40}$$

Equation 41 details step 1 for the mass fraction, while Equation 42 shows step 2 for the mass fraction.

$$\begin{aligned} &- d_1 Y^{n+\frac{1}{2}}_{i+1,j} + (1 + 2d_1) Y^{n+\frac{1}{2}}_{i,j} - d_1 Y^{n+\frac{1}{2}}_{i-1,j} \\ &= d_2 Y^{n}_{i,j+1} + (1 - 2d_2) Y^{n}_{i,j} + d_2 Y^{n}_{i,j-1} \\ &\quad + \frac{dt}{2} Q_Y\left(x_i, y_j, t_n\right) \end{aligned} \tag{41}$$

$$\begin{aligned} &- d_2 Y^{n+1}_{i,j+1} + (1 + 2d_2) Y^{n+1}_{i,j} - d_2 Y^{n+1}_{i,j-1} \\ &= d_1 Y^{n+\frac{1}{2}}_{i+1,j} + (1 - 2d_1) Y^{n+\frac{1}{2}}_{i,j} + d_1 Y^{n}_{i-1,j} \\ &\quad + \frac{dt}{2} Q_Y\left(x_i, y_j, t_{n+\frac{1}{2}}\right) \end{aligned} \tag{42}$$

Where d_1 and d_2 are shown below in Equation 43 for u and v, and Equation 44 for Y.

$$d_1 = d_2 = \frac{dt}{2 \cdot Re \cdot h^2} \tag{43}$$

$$d_1 = d_2 = \frac{dt}{2 \cdot Re \cdot Sc \cdot h^2} \tag{44}$$

3.3 CONVECTIVE TERMS (HYPERBOLIC)

Looking at the convective terms, these are used in the momentum equation. Similar to how the main method was Crank-Nicholson for viscous terms, the main method for convective is known as the total variation diminishing-Runge Kutta (TVD-RK3). This method uses ordinary differential equations ODEs to solve and has a third order of accuracy in time. However, TVD-RK3 can be combined with a weighted essentially non-oscillatory scheme (WENO-5), which has fifth–order accuracy in space, to completely solve the momentum equation. Equations 45 through 47 detail the three steps for the TVD-RK3 time integration.

$$Y_i^{(1)} = Y_i^n - a\left(x_i, t^n\right)\Delta t \left.\frac{\partial Y^n}{\partial x}\right|_i^{\pm} \tag{45}$$

$$\begin{aligned} Y_i^{(2)} = {} & Y_i^1 + \frac{3}{4}a\left(x_i, t^n\right)\Delta t \left.\frac{\partial Y^n}{\partial x}\right|_i^{\pm} \\ & - \frac{1}{4}a\left(x_i, t^n + \Delta t\right)\Delta t \left.\frac{\partial Y^{(1)}}{\partial x}\right|_i^{\pm} \end{aligned} \tag{46}$$

$$\begin{aligned} Y_i^{(n+1)} = {} & Y_i^2 + \frac{1}{12}a\left(x_i, t^n\right)\Delta t \left.\frac{\partial Y^n}{\partial x}\right|_i^{\pm} \\ & + \frac{1}{12}a\left(x_i, t^n + \Delta t\right)\Delta t \left.\frac{\partial Y^{(1)}}{\partial x}\right|_i^{\pm} \\ & - \frac{2}{3}a\left(x_i, t^n + \frac{1}{2}\Delta t\right)\Delta t \left.\frac{\partial Y^{(2)}}{\partial x}\right|_i^{\pm} \end{aligned} \tag{47}$$

The following equations are then a detailed representation of the WENO-5 method.

$$if\ a(x, t) > 0: \tag{48}$$

$$\begin{aligned} \left.\frac{\partial Y^n}{\partial x}\right|_i^{-} = {} & \frac{1}{12 \cdot \Delta x}\left(-\Delta^{+}Y_{i-2} + 7\Delta^{+}Y_{i-1} + 7\Delta^{+}Y_i - \Delta^{+}Y_{i+1}\right) \\ & - \psi_{WENO}\left(\frac{\Delta^{-}\Delta^{+}Y_{i-2}}{\Delta x}, \frac{\Delta^{-}\Delta^{+}Y_{i-1}}{\Delta x}, \frac{\Delta^{-}\Delta^{+}Y_i}{\Delta x}, \frac{\Delta^{-}\Delta^{+}Y_{i+1}}{\Delta x}\right) \end{aligned}$$

$$if\ a(x, t) < 0: \tag{49}$$

$$\begin{aligned} \left.\frac{\partial Y^n}{\partial x}\right|_i^{+} = {} & \frac{1}{12 \cdot \Delta x}\left(-\Delta^{+}Y_{i-2} + 7\Delta^{+}Y_{i-1} + 7\Delta^{+}Y_i - \Delta^{+}Y_{i+1}\right) \\ & - \psi_{WENO}\left(\frac{\Delta^{-}\Delta^{+}Y_{i+2}}{\Delta x}, \frac{\Delta^{-}\Delta^{+}Y_{i+1}}{\Delta x}, \frac{\Delta^{-}\Delta^{+}Y_i}{\Delta x}, \frac{\Delta^{-}\Delta^{+}Y_{i-1}}{\Delta x}\right) \end{aligned}$$

Where the following inputs to Equations 48 and 49, and ε is 10^{-6}.

$$\Delta^{+}Y_i = Y_{i+1} - Y_i \tag{50}$$

$$\Delta^{-}Y_i = Y_i - Y_{i-1} \tag{51}$$

$$\Delta^{+}\Delta^{-} = Y_{i+1} - 2\,Y_i + Y_{i-1} \tag{52}$$

$$\psi_{WENO}(a,\ b,\ c,\ d) = \frac{1}{3}\,\omega_0\,(a - 2b + c) + \frac{1}{6}\left(\omega_2 - \frac{1}{2}\right)(b - 2c + d) \tag{53}$$

$$\omega_0 = \frac{\alpha_0}{\alpha_0+\alpha_1+\alpha_2} \tag{54}$$

$$\omega_2 = \frac{\alpha_2}{\alpha_0+\alpha_1+\alpha_2} \tag{55}$$

$$\alpha_0 = \frac{1}{\left(\varepsilon+IS_0\right)^2} \tag{56}$$

$$\alpha_1 = \frac{6}{\left(\varepsilon+IS_1\right)^2} \tag{57}$$

$$\alpha_2 = \frac{3}{\left(\varepsilon+IS_2\right)^2} \tag{58}$$

$$IS_0 = 13\,(a - b)^2 + 3\,(a - 3b)^2 \tag{59}$$

$$IS_1 = 13\,(b - c)^2 + 3\,(b + c)^2 \tag{60}$$

$$IS_2 = 13\,(c - d)^2 + 3\,(3c - d)^2 \tag{61}$$

Now although we can compute ω_0 , ω_1 , and ω_2 with the above equations, since we have selected to perform WENO-5, ω_0 , ω_1 , and ω_2 are pre-calculated to be ω_0 = .1, ω_1 = .6 , and ω_2 = .3.

Additionally, both the u and v velocities have convective terms, each being second order in accuracy in space and time. They are shown below in Equations 62 through 77.

$$\frac{\partial uu}{\partial x}\Big|_{i+\frac{1}{2},j} = \frac{\left(u_{i+1,j}\right)^2 - \left(u_{i,j}\right)^2}{\Delta x} + O\left(\Delta x^2\right) \tag{62}$$

Where,

$$u_{i+1,j} = \frac{1}{2}\left(u_{i+\frac{3}{2},j} + u_{i+\frac{1}{2},j}\right) \tag{63}$$

$$u_{i,j} = \frac{1}{2}\left(u_{i+\frac{1}{2},j} + u_{i-\frac{1}{2},j}\right) \tag{64}$$

$$\frac{\partial uv}{\partial y}\Big|_{i+\frac{1}{2},j} = \frac{uv_{i+\frac{1}{2},j+\frac{1}{2}} - uv_{i+\frac{1}{2},j-\frac{1}{2}}}{\Delta y} + O\left(\Delta y^2\right) \tag{65}$$

Where,

$$u_{i+\frac{1}{2},j+\frac{1}{2}} = \frac{1}{2}\left(u_{i+\frac{1}{2},j} + u_{i+\frac{1}{2},j+1}\right) \tag{66}$$

$$v_{i+\frac{1}{2},j+\frac{1}{2}} = \frac{1}{2}\left(v_{i,j+\frac{1}{2}} + v_{i+1,j+\frac{1}{2}}\right) \tag{67}$$

$$u_{i+\frac{1}{2},j-\frac{1}{2}} = \frac{1}{2}\left(u_{i+\frac{1}{2},j-1} + u_{i+\frac{1}{2},j}\right) \tag{68}$$

$$v_{i+\frac{1}{2},j-\frac{1}{2}} = \frac{1}{2}\left(v_{i,j-\frac{1}{2}} + v_{i+1,j-\frac{1}{2}}\right) \tag{69}$$

$$\frac{\partial vv}{\partial y}\Big|_{i,j+\frac{1}{2}} = \frac{\left(v_{i,j+1}\right)^2-\left(v_{i,j}\right)^2}{\Delta y} + O\left(\Delta y^2\right) \tag{70}$$

Where,

$$v_{i,j+1} = \frac{1}{2}\left(v_{i,j+\frac{3}{2}} + v_{i,j+\frac{1}{2}}\right) \tag{71}$$

$$v_{i,j} = \frac{1}{2}\left(v_{i,j+\frac{1}{2}} + v_{i,j-\frac{1}{2}}\right) \tag{72}$$

$$\frac{\partial vu}{\partial x}\Big|_{i,j+\frac{1}{2}} = \frac{vu_{i+\frac{1}{2},j+\frac{1}{2}}-vu_{i-\frac{1}{2},j+\frac{1}{2}}}{\Delta x} + O\left(\Delta x^2\right) \tag{73}$$

Where,

$$u_{i+\frac{1}{2},j+\frac{1}{2}} = \frac{1}{2}\left(u_{i+\frac{1}{2},j} + u_{i+\frac{1}{2},j+1}\right) \tag{74}$$

$$v_{i+\frac{1}{2},j+\frac{1}{2}} = \frac{1}{2}\left(v_{i,j+\frac{1}{2}} + v_{i+1,j+\frac{1}{2}}\right) \tag{75}$$

$$u_{i-\frac{1}{2},j-\frac{1}{2}} = \frac{1}{2}\left(u_{i-\frac{1}{2},j} + u_{i-\frac{1}{2},j+1}\right) \tag{76}$$

$$v_{i-\frac{1}{2},j-\frac{1}{2}} = \frac{1}{2}\left(v_{i,j+\frac{1}{2}} + v_{i-1,j+\frac{1}{2}}\right) \tag{77}$$

3.4 Performance Metrics

Now looking at Equations 32 and 35, each is using 2D composite midpoint integration method which is considered to be a second order in accuracy in space and time. Additionally, Equations 34 and 36 are using the trapezoidal integration method, which is also second order in accuracy in space and time.

$$F(t) \approx \int -\frac{1}{Re}\left(\frac{v_{M+2,j}^{n+1} - v_{M+1,j}^{n+1}}{\Delta x}\right)dy \tag{78}$$

$$F(t) = \frac{\Delta x}{2}\left(\tau_w\left(x_0\right) + 2\,\tau_w\left(x_1\right) + \ \ldots\ + 2\,\tau_w\left(x_{n-1}\right) + \tau_w\left(x_n\right)\right) \tag{79}$$

$$\overline{F} = \frac{1}{10}\sum_{n=2}^{n_{max}}\left(\frac{F\left(t^{n-1}\right) + F\left(t^{n}\right)}{2}\right)\left(t^n - t^{n-1}\right) \tag{80}$$

$$A(t) = \Delta x\left(v_{i,N+1}\left(\frac{Y_{i+2,N+4} + Y_{i+2,N+3}}{2}\right)\right) \tag{81}$$

$$B = \sum_{n=2}^{n_{max}}\left(\frac{A\left(t^{n-1}\right) + A\left(t^{n}\right)}{2}\right)\left(t^n - t^{n-1}\right) \tag{82}$$

3.5 Boundary Conditions

Finally, the boundary conditions are based on the second order differences method. This then ensures ghost cells and boundaries are second order. This then coordinates with the second order of the methods above and makes the second order the lowest order observed in this solution. Both Dirichlet and Neumann boundary conditions are implemented depending on staggered velocity meshes or cell-centered mesh.

4. Results: Task 1

4.1 Solution Parameters

Looking at the results, it is important to understand what parameters were initialized which lead to the results below. As shown in Table 4-1-1, there were three different mesh sizes selected for simulation to be used for the solution verification. It should be noted only the finest mesh was selected for plotting in section 4.3. The three meshes selected were 64, 128, and 256. Constants such as Schmidt and Reynolds numbers were given in the problem statement and are listed in Table 4-1-1 as such. Finally, the last selection inputs were the CFL number which was selected to be .9 for stability reasons, and the Poisson convergence for acceptable error which was set to be 10^{-9}.

Table 4-1-1 Solution Parameters

Variable Name	Numerical Value
Mesh Size (Coarse)	64
Mesh Size (Medium)	128
Mesh Size (Fine)	256
Schmidt (Sc)	1.5
Reynolds (Re)	20
Courant–Friedrichs–Lewy (CFL)	.9
Poisson Convergence Criteria	10^{-9}

4.2 Solution Verification

After selecting the solution parameters and creating the correct solution code, shown in section 7.2, the solution verification can be conducted to determine if the resulting values of $\overline{F}$ and B makes practical sense and is correct. The standard verification process is to perform Richardson Extrapolation and then calculate the Grid Conversion Index (GCI). Richardson Extrapolation is a method of solution verification that uses extrapolation-based error estimation. Essentially, you run a simulation with different grids or meshes that vary from coarse to fine. The point of this extrapolation method is to determine the exact solution as it goes to infinity. The results of the extrapolation are a p-value for the least squares regressions and the exact value of $\overline{F}$ or B. The value of p can be shown below in Equation 83, and the exact solution in Equation 84. In these equations, f_3 is the result of $\overline{F}$ or B at mesh 1, and f_1 is the result $\overline{F}$ or B at mesh 3.

$$p = \frac{ln\left(\frac{|f_3 - f_2|}{|f_2 - f_1|}\right)}{ln(r)} \tag{83}$$

$$f_{exact} = f_1 - \frac{f_2 - f_1}{r^p - 1} \tag{84}$$

Moving on then, Grid Conversion Index (GCI) will compute how far the estimated solution from Richardson Extrapolation is from the asymptomatic value. This essentially creates the error bars for the solution and yields further validity to the solution itself. Based on the selection of three meshes (or simulations), GCI requires that a constant f_{sec} is equal to 1.25 which is used in the following equations. In Equations 85 and 86 GCI_{12} and GCI_{23} are calculated with the f-number system seen above for Richardson Extrapolation.

$$GCI_{12} = F_{sec} \cdot \frac{\left|\frac{f_1 - f_2}{f_1}\right|}{r^p - 1} \tag{85}$$

$$GCI_{23} = F_{sec} \cdot \frac{\left|\frac{f_2 - f_3}{f_2}\right|}{r^p - 1} \tag{86}$$

Upon calculating the Richardson Extrapolation and GCI values, the exact solution of $\overline{F}$ is equated to .007895 +/- .021296%, as shown in Table 4-2-1. Similarly, the exact solution of B is equated to 2.472568 +/- .050818%, as shown in Table 4-2-2.

Table 4-2-1 Solution Verification For $\overline{F}$

M	$\overline{F}$	p	GCI_{12}	GCI_{23}	ARC
64	0.0079532048	~	~	~	~
128	0.0079042018	~	~	~	~
256	0.0078967273	2.7128052707	0.0002129564	0.0013948121	1.000946540

Table 4-2-2 Solution Verification For B

M	B	p	GCI_{12}	GCI_{23}	ARC
64	2.493331412	~	~	~	~
128	2.477137645	~	~	~	~
256	2.473573823	2.183941398	0.0005081778	0.0023058036	1.001440758

4.3 Resulting Figures

After confirmation from the solution verification process, creating plots from the finest mesh for the velocities (u and v), mass fraction (Y), tangential force (F(t)), and then chemical species leaving the chamber (A(t)) is the next step. Starting with Figures 4-3-1, 4-3-2, and 4-3-3, the velocities and mass fraction will be plotted at six different time intervals (1, 3.25, 5, 6, 8, and 10). The purpose of this is to showcase the evolution of the fluid in the mixing chamber and how the inlets and outlets are all interacting. Therefore, beginning with Figure 4-3-1, the u velocity in the X-direction is showcased at each time interval. Moving on to Figure 4-3-2, the v velocity in the Y-direction is showcased at each time interval. Finally, in Figure 4-3-3 the mass fraction (Y) is plotted at each time interval.

Figure 4-3-1 U Velocity (X-Direction)

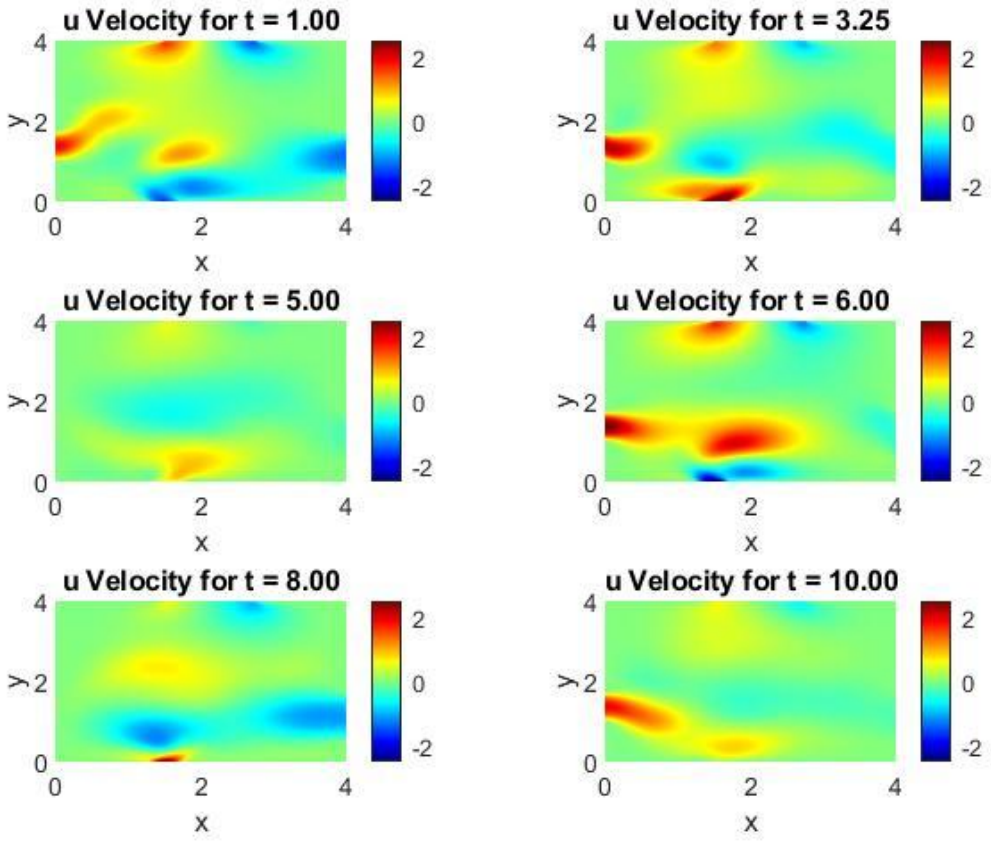


Figure 4-3-2 V Velocity (Y-Direction)

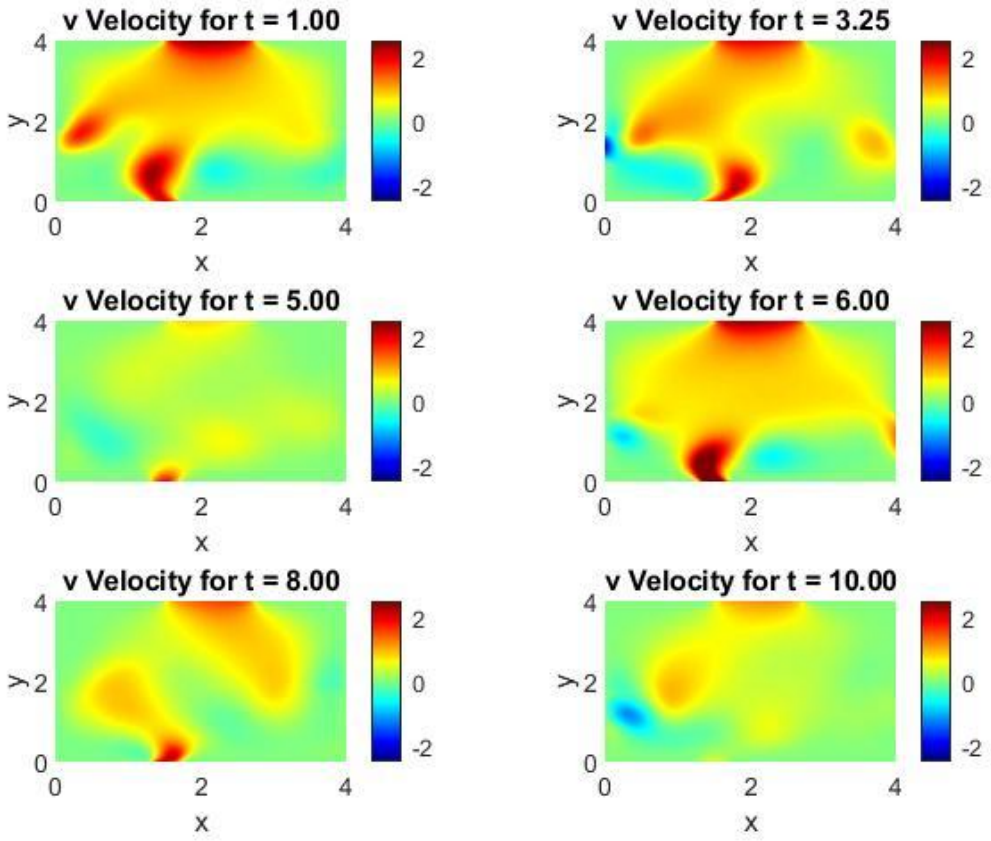
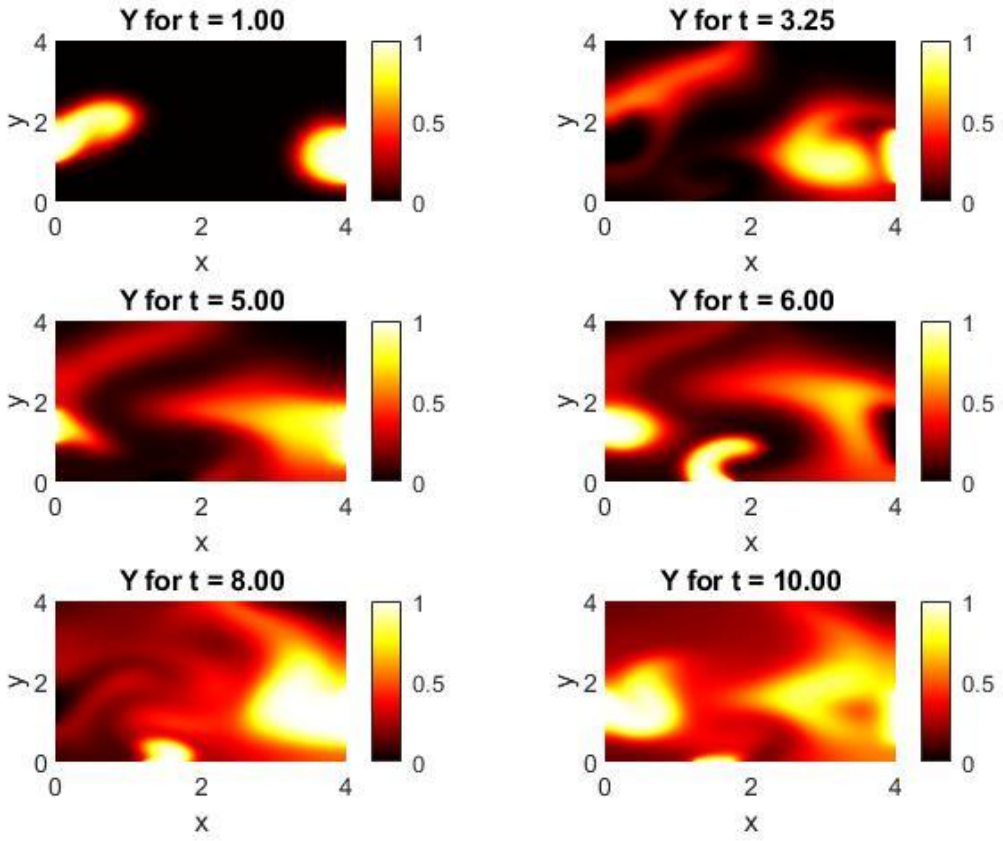


Figure 4-3-3 Mass Fraction (Y)



Continuing forward then, Figure 4-3-4 details the tangential force throughout the entire time using Equation 34 in section 2.8. Additionally, Figure 4-3-5 details the chemical species leaving the mixing chamber throughout the entire time using Equation 36 in section 2.10.

Figure 4-3-4 Tangential Force (F(t))

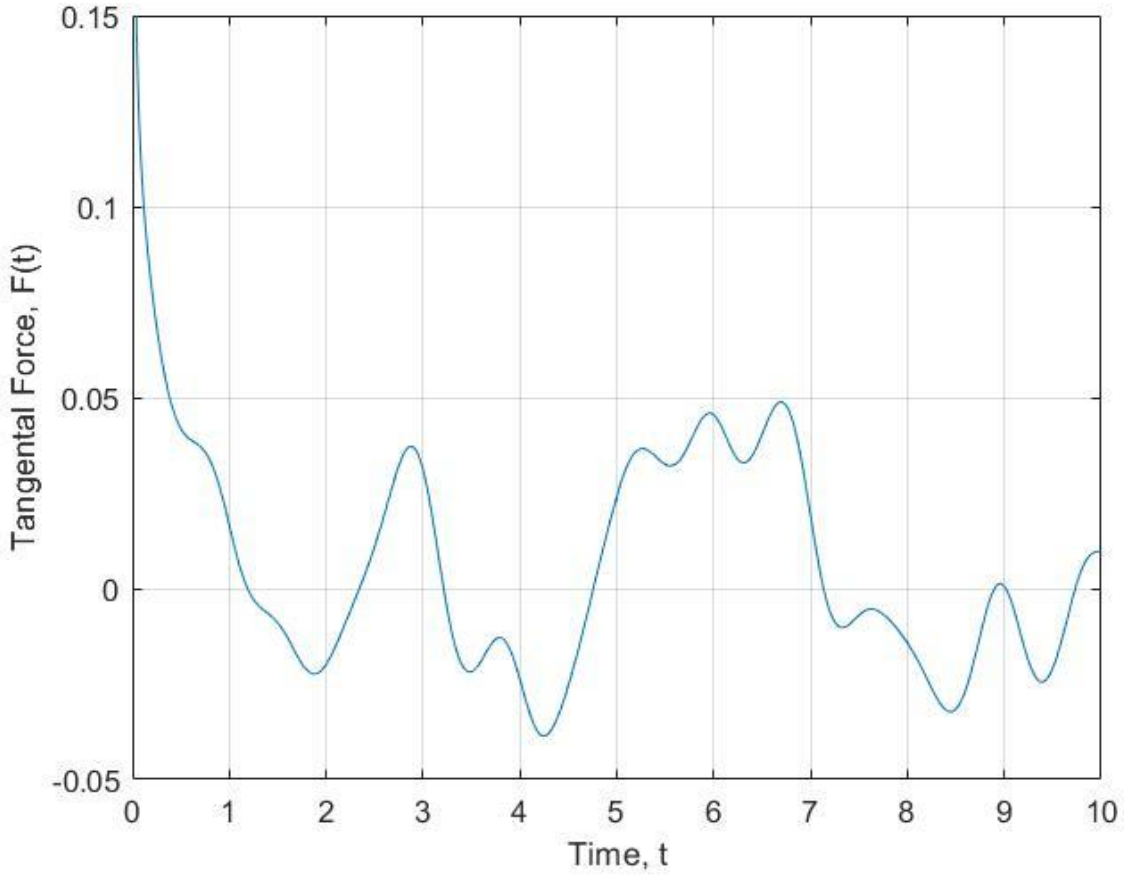
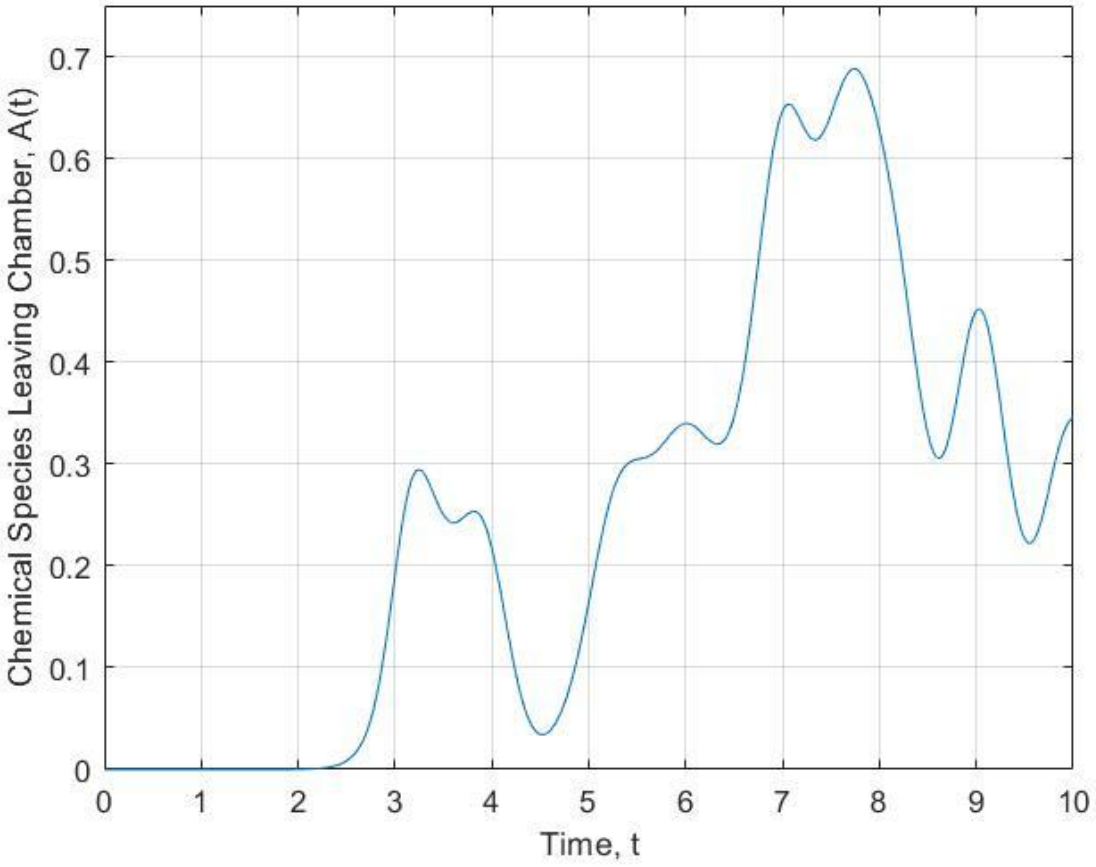


Figure 4-3-5 Chemical Species Leaving Chamber (A(t))



4.4 Discussion

Reviewing the results for Task 1, the first thing to discuss is the mesh values. The selected values are shown in Table 4-1-1 and were selected based on precision but also due to time constants. Upon removing 64 and adding 512, the time to run the program became too costly. Continuing with the initial parameters, the CFL number of .9 was selected based upon a trial and error, and previous experience from exam 9. Moving on to the solution verification, it appears that convergence is seen for the values of $\overline{F}$ and B. The observed order for $\overline{F}$ was 2.71, while the order for B was 2.18, both results are positive and within the acceptable range from one to three. Additionally, when reviewing section 3 the numerical methods used to calculate $\overline{F}$ and B were second order, while the other methods used for the velocities and mass fraction were higher. It should also be noted that the asymptotic range of convergence is shown to be very close to 1, which does confirm the accuracy of this result. Upon further mesh refinement, further research could be done to discover the true convergence and achieve an even more accurate solution for $\overline{F}$ and B.

Shifting focus from the solution verification to the flow plots and video for Task 1 Bonus, it can be seen that the flow is focused near the inlets. The velocity in the X-direction appears to show a meeting of the flow near the middle of the mixing chamber and avoiding the corners. Meanwhile, the velocity in the Y-direction shows a detailed view of the flow going from the inlets and moving toward the outlet. The flow appears to be building/colliding near the outlet when the three inlets converge. Looking at the mass fraction then, the flow first appears from inlet one and two, and then a new flow forms in the center from the combining. Vortices are very noticeable in the center of the chamber and indicate mixing is taking place! Some flow appears to be dissipated to the corners or outside of the main sections of the inlets and outlet once the mixing occurs and the vertices have finished.

5. Results: Task 2

5.1 Solution Parameters

Similar to section 4.1, it is important to understand what parameters were initialized which lead to the results below. As shown in Table 5-1-1, there were three different mesh sizes selected for simulation to be used for the solution verification. It should be noted only the finest mesh was selected for plotting in section 5.3, Resulting Figures. The three meshes selected were, again, 64, 128, and 256. Constants such as Schmidt and Reynolds numbers were given in the problem statement and are listed in Table 5-1-1 as such. Finally, the last selection inputs were the CFL number which was selected to be .9 for stability reasons, and the Poisson convergence for acceptable error which was set to be 10^{-9}.

Table 5-1-1 Solution Parameters

Variable Name	Numerical Value
Mesh Size (Coarse)	64
Mesh Size (Medium)	128
Mesh Size (Fine)	256
Schmidt (Sc)	2.5
Reynolds (Re)	75
Courant–Friedrichs–Lewy (CFL)	.9
Poisson Convergence Criteria	10^{-9}

5.2 Solution Verification

After selecting the solution parameters and creating the correct solution code, shown in section 7.3, the solution verification can be conducted to determine if the resulting values of $\overline{F}$ and B makes practical sense and is correct. The standard verification process is to perform Richardson Extrapolation and then calculate the Grid Conversion Index (GCI). The equations can be found in section 4.2 from Equations 83 to 86. Upon calculating the Richardson Extrapolation and GCI values, the exact solution for $\overline{F}$ is equated to .005665 +/- 1.705562%, as shown in Table 5-2-1. Similarly, the exact solution of B is equated to 2.770376 +/- 0.057545%, as shown in Table 5-2-2. It should be noted that in Tables 5-2-1 and 5-2-2 a mesh of 512 is present and will be explained in further detail in section 5.5.

Table 5-2-1 Solution Verification For $\overline{F}$

M	$\overline{F}$	p	GCI_{12}	GCI_{23}	ARC
64	0.005157282	~	~	~	~
128	0.005468181	~	~	~	~
256	0.005588677	1.36746367	0.01705572	0.044976310	0.978439306
512	0.005468181	1.57562182	0.00453239	0.013607155	0.992818368

Table 5-2-2 Solution Verification For B

M	B	p	GCI_{12}	GCI_{23}	ARC
64	2.691110572	~	~	~	~
128	2.781789198	~	~	~	~
256	2.771651816	3.161077343	0.000575446	0.005128592	1.00365752
512	2.781789198	-0.37504996	-0.02602418	-0.01997152	1.00476599

5.3 Resulting Figures

Similar to section 4, after confirmation from the solution verification process, creating plots from the finest mesh for the velocities (u and v), mass fraction (Y), tangential force (F(t)), and then chemical species leaving the chamber (A(t)) is the next step. Starting with Figures 5-3-1, 5-3-2, and 5-3-3, the velocities and mass fraction will be plotted at six different time intervals (1, 3.25, 5, 6, 8, and 10). The purpose of this is to showcase the evolution of the gas in the mixing chamber and how the inlets and outlets are all interacting. Therefore, beginning with Figure 5-3-1, the u velocity in the X-direction is showcased at each time interval. Moving on to Figure 5-3-2, the v velocity in the Y-direction is showcased at each time interval. Finally, in Figure 5-3-3 the mass fraction (Y) is plotted at each time interval.

Figure 5-3-1 U Velocity (X-Direction)

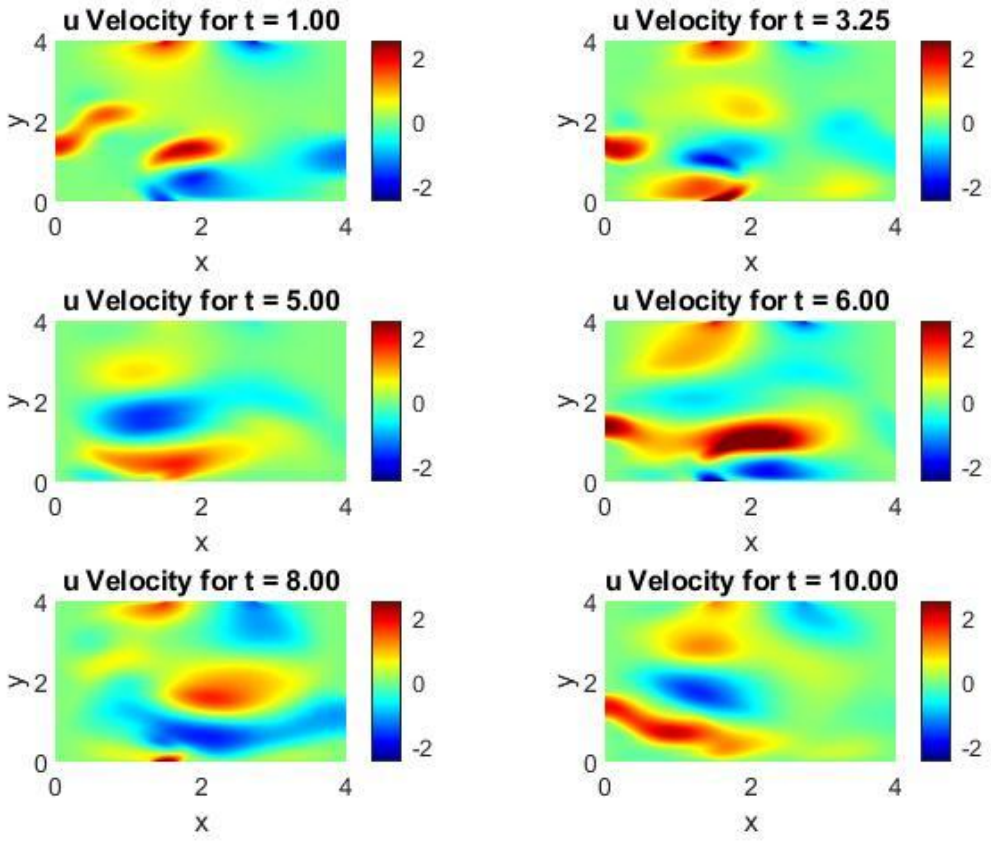


Figure 5-3-2 V Velocity (Y-Direction)

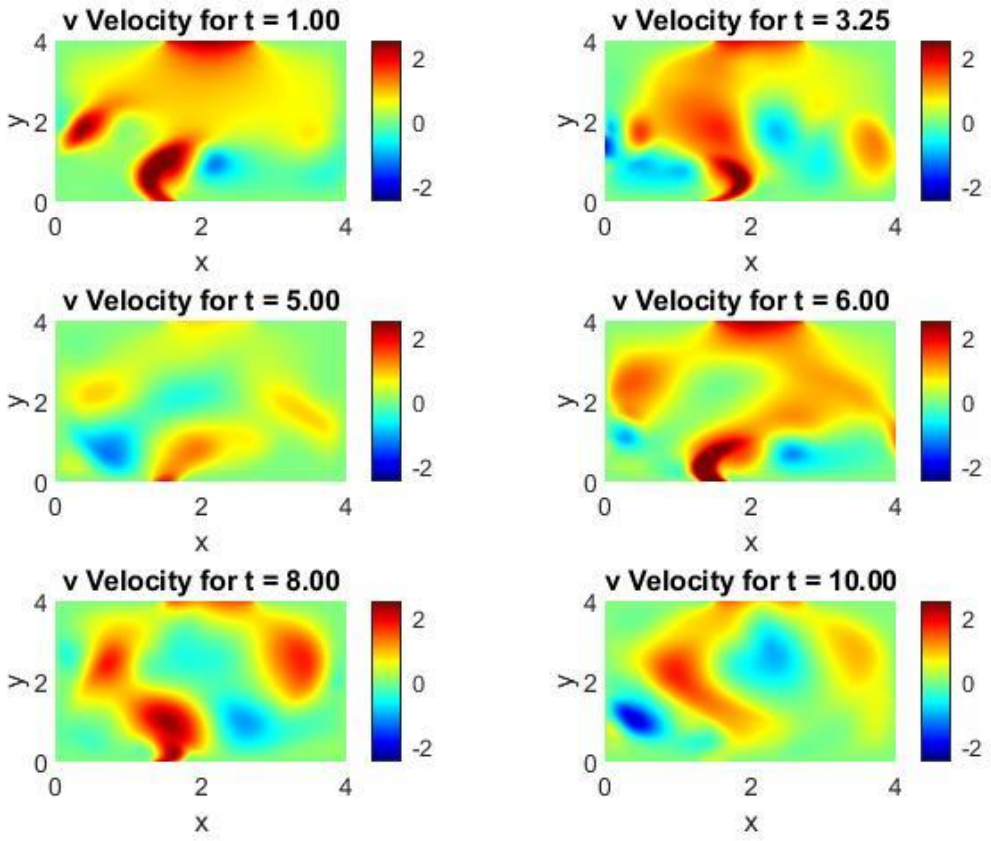
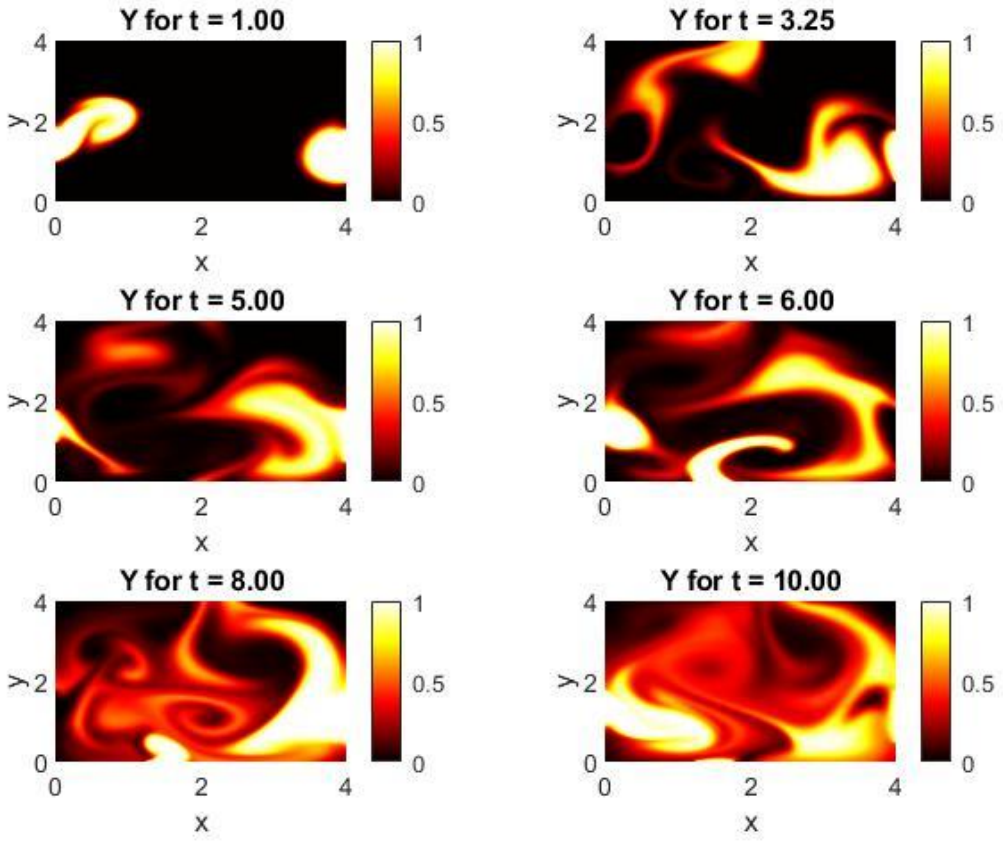


Figure 5-3-3 Mass Fraction (Y)



Continuing forward then, Figure 5-3-4 details the tangential force throughout the entire time using Equation 34 in section 2.8. Additionally, Figure 5-3-5 details the chemical species leaving the mixing chamber throughout the entire time using Equation 36 in section 2.10.

Figure 5-3-4 Tangential Force (F(t))

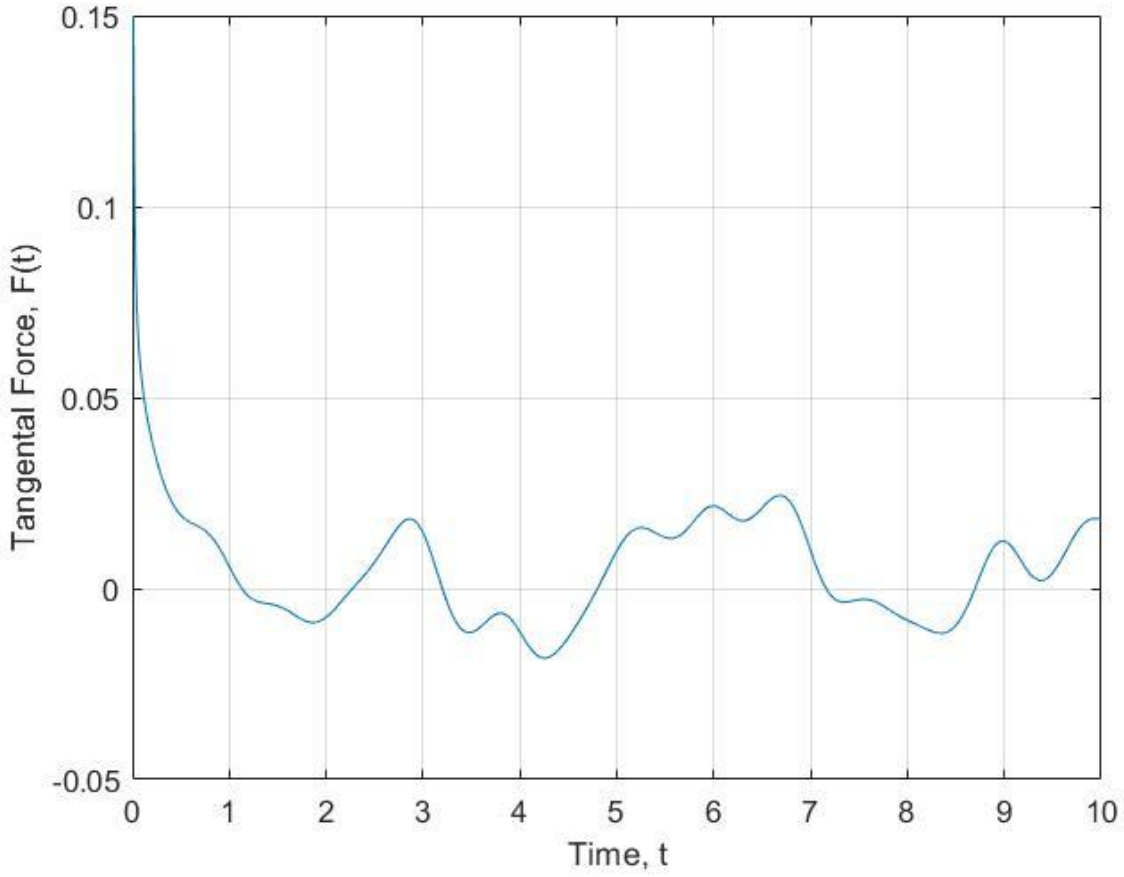
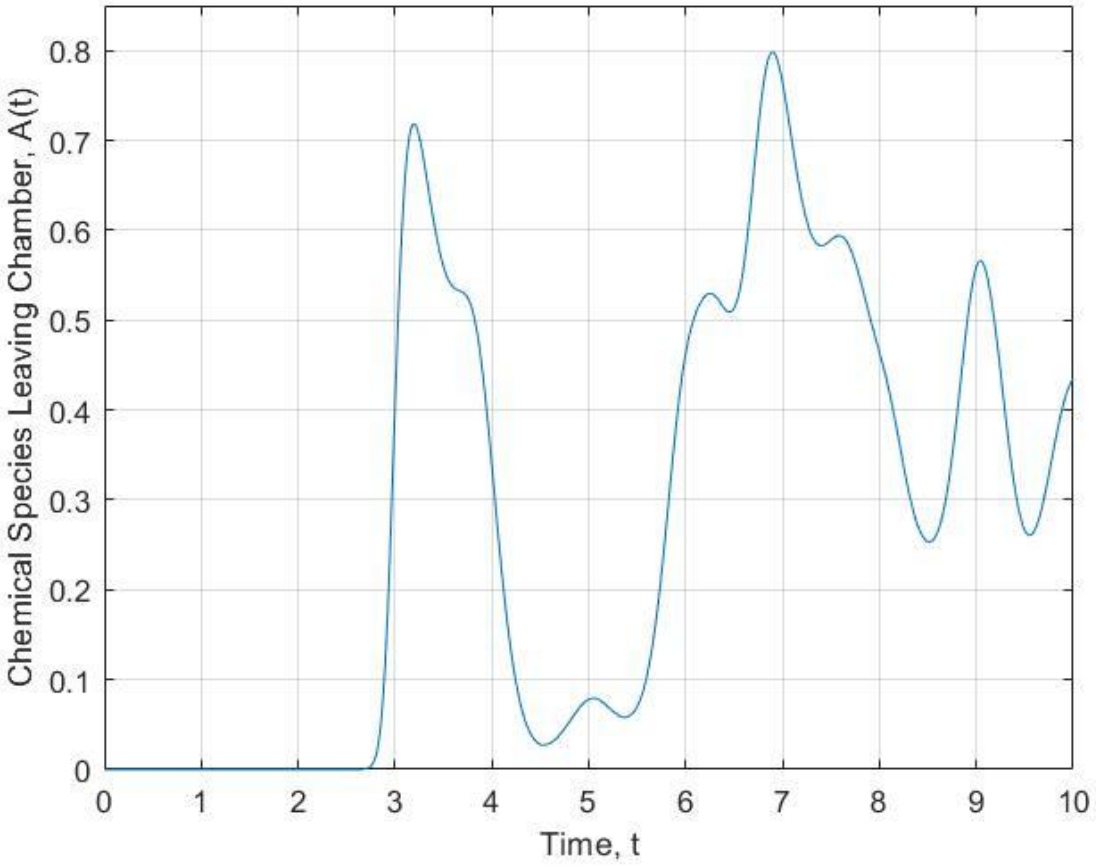


Figure 5-3-5 Chemical Species Leaving Chamber (A(t))



5.4 Discussion

Comparing this solution to Task 1, it is most notable that the Reynolds number and Schmidt numbers both increased. This would produce a faster and less viscous fluid than seen in Task 1. Now, the first thing to discuss is the mesh selection. The chosen selection, as discussed in Table 5-1-1 is 64, 128, and 256. However another simulation was attempted at a mesh of 512. However, this simulation was voided due to the observed orders of accuracy being 1.57 and -.37. Therefore, using the selected mesh values, the solution verification for $\overline{F}$ was 1.36, while the order for B was 3.16. Now, this obviously is different from what was observed in Task 1. It was expected to see a similar result of observed orders around 2, but it is clear something happens when the fluid properties are changed. Therefore, given further time running simulations at higher meshes, as well as changing the CFL number from .9 to maybe .6 and determining if any changes are observed.

Looking at the flow itself now, it shares similar effects as Task 1 with the one difference being the amount of flow. Throughout the same amount of time, there appears to be a higher amount of interactions and a greater volume of fluid entering the mixing chamber. Additionally, there appears to be a higher amount of vortices occurring, which most likely results in a better mixing.

6. Conclusion

In conclusion, this study was a success for modeling a 2D mixing chamber with the continuity and Navier Stokes equations. The flow in Task 1 showed convergence with expected orders of accuracy and the graphical representation was able to show detailed vortices and evidence of mixing. It could be concluded that this method may be inefficient due to it being simulated in MATLAB, but the premise is correct and creates a valid solution for low Reynolds numbers. As noted for Task 2, some issues begin to occur when the Reynolds number is increased and if it was increased further to turbulent flow, this method would be suboptimal. Looking at improvements for further research could be to increase the orders of every method used to be the same. As noted in section 3, we are using a variety of numerical methods of various orders of accuracy from two through five. If higher orders were used, the expected value would be easier to assume. Now for this to work, having access to more powerful computing, such as a supercomputer, may be required, as well as moving these scripts from MATLAB to a different language better equipped to handle a compiler or optimizer would greatly improve the results.

7. APPENDIX

7.1 BONUS TASK 1

```
%% MAE 561 ; Chandler Hutchens

% Bonus 1 of Final Project

format compact ;
format long ;
close all ;
clc ;

%% Givens

% Global Variables
global CFL Re Sc Lx Ly h t xc yf xf yc xc3 yc3

% Given Matrix
M = 256 ;
N = M ;

% Times for plotting
T = 0.1:0.1:10 ;

% Define Global Variables
CFL = .9 ;
Re = 20 ;
Sc = 1.5 ;
Lx = 4 ;
Ly = Lx ;
h = Lx/M ;
t = 0 ;
endTime = 10 ; % Allocated endtime
ifig = 1 ;  % Counter
I = 1 ; % Counter
nIterMax = 20 ;
epsilon = 1e-9 ;

% u
xf = linspace(0,Lx,M+1)' ;
yc = linspace(0-(h/2),Ly+(h/2),N+2)' ;

% v
xc = linspace(0-(h/2),Lx+(h/2),M+2)' ;
yf = linspace(0,Ly,N+1)' ;

% Y3
```

```
xc3 = linspace(0-5*(h/2),Lx+5*(h/2),M+6)' ;
yc3 = linspace(0-5*(h/2),Ly+5*(h/2),N+6)' ;

%% Calculations

% Generate Terms
u0 = zeros(M+1,N+2) ; Qu = u0 ;
v0 = zeros(M+2,N+1) ; Qv = v0 ;
Y = zeros(M+6,N+6) ; QY = Y ;
phi = zeros(M+2,N+2) ;

% Boundary Conditions
u0 = bc_u(u0,t) ;
v0 = bc_v(v0,t) ;
Y = bc_Y3(Y,t) ;

[u0,v0] = correctOutlet(u0,v0) ;
divV = calcDivV(u0,v0) ;
phi = myPoisson(phi,divV,h,nIterMax,epsilon) ;
[u0,v0] = projectV(u0,v0,phi,1) ;

u1 = u0 ;
v1 = v0 ;

a = VideoWriter('u','MPEG-4') ;
open(a) ;

b = VideoWriter('v','MPEG-4') ;
open(b) ;

c = VideoWriter('Y','MPEG-4') ;
open(c) ;

examFig1 = figure(1) ;
pcolor(xf,yc,u0') ;
shading interp  ;
colormap jet ;
colorbar ;
title(sprintf('u Velocity for t = %.2f',0))
xlabel('x')
ylabel('y')
xlim([0 Lx]) ;
ylim([0 Ly]) ;
caxis([-2.5 2.5]) ;
f1 = getframe(1) ;
writeVideo(a,f1) ;
```

```
examFig2 = figure(2) ;
pcolor(xc,yf,v0') ;
shading interp  ;
colormap jet ;
colorbar ;
title(sprintf('v Velocity for t = %.2f',0))
xlabel('x')
ylabel('y')
xlim([0 Lx]) ;
ylim([0 Ly]) ;
caxis([-2.5 2.5]) ;
f2 = getframe(2) ;
writeVideo(b,f2) ;

examFig3 = figure(3) ;
pcolor(xc3,yc3,Y') ;
shading interp  ;
colormap hot ;
colorbar ;
title(sprintf('Y for t = %.2f',0))
xlabel('x')
ylabel('y')
xlim([0 Lx]) ;
ylim([0 Ly]) ;
caxis([0 1]) ;
f3 = getframe(3) ;
writeVideo(c,f3) ;

while (t < endTime)

   outputTime = endTime ;
   if ifig <= length(T)
      outputTime = T(ifig) ;
   end

   [dt,outputFlag] = calcDt561(t,outputTime, u0, v0) ;

   [Hu0, Hv0] = hyperbolic_uv_2D(u0, v0) ;
   [Hu1, Hv1] = hyperbolic_uv_2D(u1, v1) ;

   [u1p] = parabolic_CN1_2D_u(u0, Qu+1.5*Hu0-.5*Hu1, dt) ;
   [u1p] = parabolic_CN2_2D_u(u1p, Qu+1.5*Hu0-.5*Hu1, dt) ;

   [v1p] = parabolic_CN1_2D_v(v0, Qv+1.5*Hv0-.5*Hv1, dt) ;
   [v1p] = parabolic_CN2_2D_v(v1p, Qv+1.5*Hv0-.5*Hv1, dt) ;

   [HY] = hyperbolic_Y_WENO_2D(Y, u0, v0, u1p, v1p, dt) ;
```

```
[Y] = parabolic_CN1_2D_Y3(Y, QY+HY, dt) ;
[Y] = parabolic_CN2_2D_Y3(Y, QY+HY, dt) ;

u1 = u0 ;
v1 = v0 ;

u0 = u1p ;
v0 = v1p ;

[u0,v0] = correctOutlet(u0,v0) ;
divV = calcDivV(u0,v0) ;
phi = myPoisson(phi,divV/dt,h,nIterMax,epsilon) ;
[u0,v0] = projectV(u0,v0,phi,dt) ;

%new time step
t = t + dt ;

if outputFlag == 1

   examFig1 = figure(1) ;
   pcolor(xf,yc,u0') ;
   shading interp  ;
   colormap jet ;
   colorbar ;
   title(sprintf('u Velocity for t = %.2f',T(ifig)))
   xlabel('x')
   ylabel('y')
   xlim([0 Lx]) ;
   ylim([0 Ly]) ;
   caxis([-2.5 2.5]) ;
   f1 = getframe(1) ;
   writeVideo(a,f1) ;

   examFig2 = figure(2) ;
   pcolor(xc,yf,v0') ;
   shading interp  ;
   colormap jet ;
   colorbar ;
   title(sprintf('v Velocity for t = %.2f',T(ifig)))
   xlabel('x')
   ylabel('y')
   xlim([0 Lx]) ;
   ylim([0 Ly]) ;
   caxis([-2.5 2.5]) ;
   f2 = getframe(2) ;
   writeVideo(b,f2) ;
```

```
    examFig3 = figure(3) ;
    pcolor(xc3,yc3,Y') ;
    shading interp  ;
    colormap hot ;
    colorbar ;
    title(sprintf('Y for t = %.2f',T(ifig)))
    xlabel('x')
    ylabel('y')
    xlim([0 Lx]) ;
    ylim([0 Ly]) ;
    caxis([0 1]) ;
    f3 = getframe(3) ;
    writeVideo(c,f3) ;

    ifig = ifig + 1 ;

  end

end

close(a) ;
close(b) ;
close(c) ;
```

7.2 Solution Code: Task 1

```
%% MAE 561 ; Chandler Hutchens

% Task 1 of Final Project

format compact ;
format long ;
close all ;
clear all ;
clc ;

%% Givens

% Global Variables
global CFL Re Sc Lx Ly h t xc yf xf yc xc3 yc3

% Given Matrix
M = [64 128 256] ;
N = M ;
% Define Global Variables
CFL = .9 ;
Re = 20 ;
Sc = 1.5 ;
Lx = 4 ;
Ly = Lx ;

%% Step 1

r = 2 ;

for i = 1:length(M)

   h = Lx/M(i) ;

   u0 = zeros(M(i)+1,N(i)+2) ;
   v0 = zeros(M(i)+2,N(i)+1) ;
   Y = zeros(M(i)+6,N(i)+6) ;
   Qu = zeros(M(i)+1,N(i)+2) ;
   Qv = zeros(M(i)+2,N(i)+1) ;
   QY = zeros(M(i)+6,N(i)+6) ;
   phi = zeros(M(i)+2,N(i)+2) ;

   % u
   xf = linspace(0,Lx,M(i)+1)' ;
   yc = linspace(0-(h/2),Ly+(h/2),N(i)+2)' ;

   % v
```

```
xc = linspace(0-(h/2),Lx+(h/2),M(i)+2)' ;
yf = linspace(0,Ly,N(i)+1)' ;

% Y3
xc3 = linspace(0-5*(h/2),Lx+5*(h/2),M(i)+6)' ;
yc3 = linspace(0-5*(h/2),Ly+5*(h/2),N(i)+6)' ;

t = 0 ;

% Boundary Conditions
u0 = bc_u(u0,t) ;
v0 = bc_v(v0,t) ;
Y = bc_Y3(Y,t) ;

[u0,v0] = correctOutlet(u0,v0) ;

dt = 1 ;
nIterMax = 20 ;
epsilon = 1e-9 ;
divV = calcDivV(u0,v0) ;
f = divV/dt ;
phi = myPoisson(phi,f,h,nIterMax,epsilon) ;
[u0,v0] = projectV(u0,v0,phi,dt) ;

u1 = u0 ;
v1 = v0 ;

I = 1 ;
time(I) = t ;
F(I) = sensor(v0) ;
A(I) = sensorY3(v0,Y) ;

endTime = 10 ;
while (t < endTime)

   outputTime = endTime ;
   [dt,outputFlag] = calcDt561(t,outputTime, u0, v0) ;

   [Hu0, Hv0] = hyperbolic_uv_2D(u0, v0) ;
   [Hu1, Hv1] = hyperbolic_uv_2D(u1, v1) ;

   [u1p] = parabolic_CN1_2D_u(u0, Qu+1.5*Hu0-.5*Hu1, dt) ;
   [u1p] = parabolic_CN2_2D_u(u1p, Qu+1.5*Hu0-.5*Hu1, dt) ;

   [v1p] = parabolic_CN1_2D_v(v0, Qv+1.5*Hv0-.5*Hv1, dt) ;
   [v1p] = parabolic_CN2_2D_v(v1p, Qv+1.5*Hv0-.5*Hv1, dt) ;
```

```
        [HY] = hyperbolic_Y_WENO_2D(Y, u0, v0, u1p, v1p, dt) ;

        [Y] = parabolic_CN1_2D_Y3(Y, QY+HY, dt) ;
        [Y] = parabolic_CN2_2D_Y3(Y, QY+HY, dt) ;

        u1 = u0 ;
        v1 = v0 ;

        u0 = u1p ;
        v0 = v1p ;

        [u0,v0] = correctOutlet(u0,v0) ;

        nIterMax = 20 ;
        epsilon = 1e-9 ;
        divV = calcDivV(u0,v0) ;
        f = divV/dt ;
        phi = myPoisson(phi,f,h,nIterMax,epsilon) ;
        [u0,v0] = projectV(u0,v0,phi,dt) ;

        t = t + dt ;
        I = I + 1 ;

        time(I) = t ;
        F(I) = sensor(v0) ;
        A(I) = sensorY3(v0,Y) ;

    end

    Fb(i) = (1/10)*sum((F(1:end-1)+F(2:end))/2.*(time(2:end)-time(1:end-1))) ;

    B(i) = sum((A(1:end-1)+A(2:end))/2.*(time(2:end)-time(1:end-1))) ;

end

%% Step 2

f1 = Fb(3) ;
f2 = Fb(2) ;
f3 = Fb(1) ;

f4 = B(3) ;
f5 = B(2) ;
f6 = B(1) ;

%% Step 3
```

```
p1 = log((abs(f3-f2)/abs(f2-f1)))/log(r) ;
p2 = log((abs(f6-f5)/abs(f5-f4)))/log(r) ;

%% Step 4

fe1 = f1 + ((f1-f2)/(r^p1-1)) ;
fe2 = f4 + ((f4-f5)/(r^p2-1)) ;

%% Step 5

Fsec = 1.25 ; % 3 or more sims

GCI1_12 = Fsec * (abs((f1-f2)/f1))/(r^p1-1) ;

GCI1_23 = Fsec * (abs((f2-f3)/f2))/(r^p1-1) ;

GCI2_12 = Fsec * (abs((f4-f5)/f4))/(r^p2-1) ;

GCI2_23 = Fsec * (abs((f5-f6)/f5))/(r^p2-1) ;

%% Step 6

arc1 = (GCI1_12/GCI1_23)*(r^p1) ;
arc2 = (GCI2_12/GCI2_23)*(r^p2) ;

% Should be between .95 and 1.05
fprintf('Asymptotic range of convergence for Fb= %f \n', arc1)
fprintf('Asymptotic range of convergence for B = %f \n', arc2)

%% Step 7

fprintf('Solution for Fb = %f +/- %f \n', fe1, GCI1_12*100)

fprintf('Solution for B = %f +/- %f \n', fe2, GCI2_12*100)

%% Table

Fb = [f1 f2 f3] ;
pt1 = [p1 p1 p1] ;
GCI1_12t = [GCI1_12 GCI1_12 GCI1_12] ;
GCI1_23t = [GCI1_23 GCI1_23 GCI1_23] ;
arct1 = [arc1 arc1 arc1] ;

examTable1 = table(M',Fb',pt1',GCI1_12t',GCI1_23t',arct1','VariableNames',{'M','Fb','p','GCI_12','GCI_23','ARC'})

B = [f4 f5 f6] ;
pt2 = [p2 p2 p2] ;
```

```
GCI2_12t = [GCI2_12 GCI2_12 GCI2_12] ;
GCI2_23t = [GCI2_23 GCI2_23 GCI2_23] ;
arct2 = [arc2 arc2 arc2] ;

examTable2 = table(M',B',pt2',GCI2_12t',GCI2_23t',arct2','VariableNames',{'M','B','p','GCI_12','GCI_23','ARC'})

%% Plotting Best Mesh

% Given Matrix
M = 256 ;
N = M ;

% Times for plotting
T = [1,3.25,5,6,8,10] ;

% Define Global Variables
CFL = .9 ;
Re = 20 ;
Sc = 1.5 ;
Lx = 4 ;
Ly = Lx ;
h = Lx/M ;
t = 0 ;
endTime = 10 ; % Allocated endtime
ifig = 1 ;  % Counter
I = 1 ; % Counter
nIterMax = 20 ;
epsilon = 1e-9 ;

% u
xf = linspace(0,Lx,M+1)' ;
yc = linspace(0-(h/2),Ly+(h/2),N+2)' ;

% v
xc = linspace(0-(h/2),Lx+(h/2),M+2)' ;
yf = linspace(0,Ly,N+1)' ;

% Y3
xc3 = linspace(0-5*(h/2),Lx+5*(h/2),M+6)' ;
yc3 = linspace(0-5*(h/2),Ly+5*(h/2),N+6)' ;

% Calculations

% Generate Terms
u0 = zeros(M+1,N+2) ; Qu = u0 ;
v0 = zeros(M+2,N+1) ; Qv = v0 ;
Y = zeros(M+6,N+6) ; QY = Y ;
```

```
phi = zeros(M+2,N+2) ;

% Boundary Conditions
u0 = bc_u(u0,t) ;
v0 = bc_v(v0,t) ;
Y = bc_Y3(Y,t) ;

[u0,v0] = correctOutlet(u0,v0) ;
divV = calcDivV(u0,v0) ;
phi = myPoisson(phi,divV,h,nIterMax,epsilon) ;
[u0,v0] = projectV(u0,v0,phi,1) ;

u1 = u0 ;
v1 = v0 ;

while (t < endTime)

   outputTime = endTime ;
   if ifig <= length(T)
      outputTime = T(ifig) ;
   end

   [dt,outputFlag] = calcDt561(t,outputTime, u0, v0) ;

   [Hu0, Hv0] = hyperbolic_uv_2D(u0, v0) ;
   [Hu1, Hv1] = hyperbolic_uv_2D(u1, v1) ;

   [u1p] = parabolic_CN1_2D_u(u0, Qu+1.5*Hu0-.5*Hu1, dt) ;
   [u1p] = parabolic_CN2_2D_u(u1p, Qu+1.5*Hu0-.5*Hu1, dt) ;

   [v1p] = parabolic_CN1_2D_v(v0, Qv+1.5*Hv0-.5*Hv1, dt) ;
   [v1p] = parabolic_CN2_2D_v(v1p, Qv+1.5*Hv0-.5*Hv1, dt) ;

   [HY] = hyperbolic_Y_WENO_2D(Y, u0, v0, u1p, v1p, dt) ;

   [Y] = parabolic_CN1_2D_Y3(Y, QY+HY, dt) ;
   [Y] = parabolic_CN2_2D_Y3(Y, QY+HY, dt) ;

   u1 = u0 ;
   v1 = v0 ;

   u0 = u1p ;
   v0 = v1p ;

   [u0,v0] = correctOutlet(u0,v0) ;
   divV = calcDivV(u0,v0) ;
   phi = myPoisson(phi,divV/dt,h,nIterMax,epsilon) ;
```

```
[u0,v0] = projectV(u0,v0,phi,dt) ;

%new time step
t = t + dt ;
I = I + 1 ;

time(I) = t ;
F(I) = sensor(v0) ;
A(I) = sensorY3(v0,Y) ;

if outputFlag == 1

   examFig1 = figure(1) ;
   subplot(3,2,ifig);
   pcolor(xf,yc,u0') ;
   shading interp  ;
   colormap jet ;
   colorbar ;
   title(sprintf('u Velocity for t = %.2f',T(ifig)))
   xlabel('x')
   ylabel('y')
   xlim([0 Lx]) ;
   ylim([0 Ly]) ;
   caxis([-2.5 2.5]) ;

   examFig2 = figure(2) ;
   subplot(3,2,ifig);
   pcolor(xc,yf,v0') ;
   shading interp  ;
   colormap jet ;
   colorbar ;
   title(sprintf('v Velocity for t = %.2f',T(ifig)))
   xlabel('x')
   ylabel('y')
   xlim([0 Lx]) ;
   ylim([0 Ly]) ;
   caxis([-2.5 2.5]) ;

   examFig3 = figure(3) ;
   subplot(3,2,ifig);
   pcolor(xc3,yc3,Y') ;
   shading interp  ;
   colormap hot ;
   colorbar ;
   title(sprintf('Y for t = %.2f',T(ifig)))
   xlabel('x')
   ylabel('y')
```

```
        xlim([0 Lx]) ;
        ylim([0 Ly]) ;
        caxis([0 1]) ;

        ifig = ifig + 1 ;

    end

end

examFig4 = figure(4) ;
plot(time,F) ;
%title('Tangental Force vs time')
xlabel('Time, t')
ylabel('Tangental Force, F(t)')
xlim([0 10]) ;
ylim([-0.05 .15]) ;
grid on

examFig5 = figure(5) ;
plot(time,A) ;
%title('Chemical Species Leaving Chamber vs time')
xlabel('Time, t')
ylabel('Chemical Species Leaving Chamber, A(t)')
xlim([0 10]) ;
ylim([0 .75]) ;
grid on
```

7.3 Solution Code: Task 2

```
%% MAE 561 ; Chandler Hutchens

% Task 1 of Final Project

format compact ;
format long ;
close all ;
clear all ;
clc ;

%% Givens

% Global Variables
global CFL Re Sc Lx Ly h t xc yf xf yc xc3 yc3

% Given Matrix
M = [64 128 256] ;
N = M ;
% Define Global Variables
CFL = .9 ;
Re = 75 ;
Sc = 2.5 ;
Lx = 4 ;
Ly = Lx ;

%% Step 1

r = 2 ;

for i = 1:length(M)

   h = Lx/M(i) ;

   u0 = zeros(M(i)+1,N(i)+2) ;
   v0 = zeros(M(i)+2,N(i)+1) ;
   Y = zeros(M(i)+6,N(i)+6) ;
   Qu = zeros(M(i)+1,N(i)+2) ;
   Qv = zeros(M(i)+2,N(i)+1) ;
   QY = zeros(M(i)+6,N(i)+6) ;
   phi = zeros(M(i)+2,N(i)+2) ;

   % u
   xf = linspace(0,Lx,M(i)+1)' ;
   yc = linspace(0-(h/2),Ly+(h/2),N(i)+2)' ;

   % v
```

```
xc = linspace(0-(h/2),Lx+(h/2),M(i)+2)' ;
yf = linspace(0,Ly,N(i)+1)' ;

% Y3
xc3 = linspace(0-5*(h/2),Lx+5*(h/2),M(i)+6)' ;
yc3 = linspace(0-5*(h/2),Ly+5*(h/2),N(i)+6)' ;

t = 0 ;

% Boundary Conditions
u0 = bc_u(u0,t) ;
v0 = bc_v(v0,t) ;
Y = bc_Y3(Y,t) ;

[u0,v0] = correctOutlet(u0,v0) ;

dt = 1 ;
nIterMax = 20 ;
epsilon = 1e-9 ;
divV = calcDivV(u0,v0) ;
f = divV/dt ;
phi = myPoisson(phi,f,h,nIterMax,epsilon) ;
[u0,v0] = projectV(u0,v0,phi,dt) ;

u1 = u0 ;
v1 = v0 ;

I = 1 ;
time(I) = t ;
F(I) = sensor(v0) ;
A(I) = sensorY3(v0,Y) ;

endTime = 10 ;
while (t < endTime)

   outputTime = endTime ;
   [dt,outputFlag] = calcDt561(t,outputTime, u0, v0) ;

   [Hu0, Hv0] = hyperbolic_uv_2D(u0, v0) ;
   [Hu1, Hv1] = hyperbolic_uv_2D(u1, v1) ;

   [u1p] = parabolic_CN1_2D_u(u0, Qu+1.5*Hu0-.5*Hu1, dt) ;
   [u1p] = parabolic_CN2_2D_u(u1p, Qu+1.5*Hu0-.5*Hu1, dt) ;

   [v1p] = parabolic_CN1_2D_v(v0, Qv+1.5*Hv0-.5*Hv1, dt) ;
   [v1p] = parabolic_CN2_2D_v(v1p, Qv+1.5*Hv0-.5*Hv1, dt) ;
```

```
        [HY] = hyperbolic_Y_WENO_2D(Y, u0, v0, u1p, v1p, dt) ;

        [Y] = parabolic_CN1_2D_Y3(Y, QY+HY, dt) ;
        [Y] = parabolic_CN2_2D_Y3(Y, QY+HY, dt) ;

        u1 = u0 ;
        v1 = v0 ;

        u0 = u1p ;
        v0 = v1p ;

        [u0,v0] = correctOutlet(u0,v0) ;

        nIterMax = 20 ;
        epsilon = 1e-9 ;
        divV = calcDivV(u0,v0) ;
        f = divV/dt ;
        phi = myPoisson(phi,f,h,nIterMax,epsilon) ;
        [u0,v0] = projectV(u0,v0,phi,dt) ;

        t = t + dt ;
        I = I + 1 ;

        time(I) = t ;
        F(I) = sensor(v0) ;
        A(I) = sensorY3(v0,Y) ;

    end

    Fb(i) = (1/10)*sum((F(1:end-1)+F(2:end))/2.*(time(2:end)-time(1:end-1))) ;

    B(i) = sum((A(1:end-1)+A(2:end))/2.*(time(2:end)-time(1:end-1))) ;

end

%% Step 2

f1 = Fb(3) ;
f2 = Fb(2) ;
f3 = Fb(1) ;

f4 = B(3) ;
f5 = B(2) ;
f6 = B(1) ;

%% Step 3
```

```
p1 = log((abs(f3-f2)/abs(f2-f1)))/log(r) ;
p2 = log((abs(f6-f5)/abs(f5-f4)))/log(r) ;

%% Step 4

fe1 = f1 + ((f1-f2)/(r^p1-1)) ;
fe2 = f4 + ((f4-f5)/(r^p2-1)) ;

%% Step 5

Fsec = 1.25 ; % 3 or more sims

GCI1_12 = Fsec * (abs((f1-f2)/f1))/(r^p1-1) ;

GCI1_23 = Fsec * (abs((f2-f3)/f2))/(r^p1-1) ;

GCI2_12 = Fsec * (abs((f4-f5)/f4))/(r^p2-1) ;

GCI2_23 = Fsec * (abs((f5-f6)/f5))/(r^p2-1) ;

%% Step 6

arc1 = (GCI1_12/GCI1_23)*(r^p1) ;
arc2 = (GCI2_12/GCI2_23)*(r^p2) ;

% Should be between .95 and 1.05
fprintf('Asymptotic range of convergence for Fb= %f \n', arc1)
fprintf('Asymptotic range of convergence for B = %f \n', arc2)

%% Step 7

fprintf('Solution for Fb = %f +/- %f \n', fe1, GCI1_12*100)

fprintf('Solution for B = %f +/- %f \n', fe2, GCI2_12*100)

%% Table

Fb = [f1 f2 f3] ;
pt1 = [p1 p1 p1] ;
GCI1_12t = [GCI1_12 GCI1_12 GCI1_12] ;
GCI1_23t = [GCI1_23 GCI1_23 GCI1_23] ;
arct1 = [arc1 arc1 arc1] ;

examTable1 = table(M',Fb',pt1',GCI1_12t',GCI1_23t',arct1','VariableNames',{'M','Fb','p','GCI_12','GCI_23','ARC'})

B = [f4 f5 f6] ;
pt2 = [p2 p2 p2] ;
```

```
GCI2_12t = [GCI2_12 GCI2_12 GCI2_12] ;
GCI2_23t = [GCI2_23 GCI2_23 GCI2_23] ;
arct2 = [arc2 arc2 arc2] ;

examTable2 = table(M',B',pt2',GCI2_12t',GCI2_23t',arct2','VariableNames',{'M','B','p','GCI_12','GCI_23','ARC'})

%% Plotting Best Mesh

% Given Matrix
M = 256 ;
N = M ;

% Times for plotting
T = [1,3.25,5,6,8,10] ;

% Define Global Variables
CFL = .9 ;
Re = 75 ;
Sc = 2.5 ;
Lx = 4 ;
Ly = Lx ;
h = Lx/M ;
t = 0 ;
endTime = 10 ; % Allocated endtime
ifig = 1 ;  % Counter
I = 1 ; % Counter
nIterMax = 20 ;
epsilon = 1e-9 ;

% u
xf = linspace(0,Lx,M+1)' ;
yc = linspace(0-(h/2),Ly+(h/2),N+2)' ;

% v
xc = linspace(0-(h/2),Lx+(h/2),M+2)' ;
yf = linspace(0,Ly,N+1)' ;

% Y3
xc3 = linspace(0-5*(h/2),Lx+5*(h/2),M+6)' ;
yc3 = linspace(0-5*(h/2),Ly+5*(h/2),N+6)' ;

% Generate Terms
u0 = zeros(M+1,N+2) ; Qu = u0 ;
v0 = zeros(M+2,N+1) ; Qv = v0 ;
Y = zeros(M+6,N+6) ; QY = Y ;
phi = zeros(M+2,N+2) ;
```

```
% Boundary Conditions
u0 = bc_u(u0,t) ;
v0 = bc_v(v0,t) ;
Y = bc_Y3(Y,t) ;

[u0,v0] = correctOutlet(u0,v0) ;
divV = calcDivV(u0,v0) ;
phi = myPoisson(phi,divV,h,nIterMax,epsilon) ;
[u0,v0] = projectV(u0,v0,phi,1) ;

u1 = u0 ;
v1 = v0 ;

while (t < endTime)

   outputTime = endTime ;
   if ifig <= length(T)
      outputTime = T(ifig) ;
   end

   [dt,outputFlag] = calcDt561(t,outputTime, u0, v0) ;

   [Hu0, Hv0] = hyperbolic_uv_2D(u0, v0) ;
   [Hu1, Hv1] = hyperbolic_uv_2D(u1, v1) ;

   [u1p] = parabolic_CN1_2D_u(u0, Qu+1.5*Hu0-.5*Hu1, dt) ;
   [u1p] = parabolic_CN2_2D_u(u1p, Qu+1.5*Hu0-.5*Hu1, dt) ;

   [v1p] = parabolic_CN1_2D_v(v0, Qv+1.5*Hv0-.5*Hv1, dt) ;
   [v1p] = parabolic_CN2_2D_v(v1p, Qv+1.5*Hv0-.5*Hv1, dt) ;

   [HY] = hyperbolic_Y_WENO_2D(Y, u0, v0, u1p, v1p, dt) ;

   [Y] = parabolic_CN1_2D_Y3(Y, QY+HY, dt) ;
   [Y] = parabolic_CN2_2D_Y3(Y, QY+HY, dt) ;

   u1 = u0 ;
   v1 = v0 ;

   u0 = u1p ;
   v0 = v1p ;

   [u0,v0] = correctOutlet(u0,v0) ;
   divV = calcDivV(u0,v0) ;
   phi = myPoisson(phi,divV/dt,h,nIterMax,epsilon) ;
   [u0,v0] = projectV(u0,v0,phi,dt) ;
```

```
%new time step
t = t + dt ;
I = I + 1 ;

time(I) = t ;
F(I) = sensor(v0) ;
A(I) = sensorY3(v0,Y) ;

if outputFlag == 1

   examFig1 = figure(1) ;
   subplot(3,2,ifig);
   pcolor(xf,yc,u0') ;
   shading interp  ;
   colormap jet ;
   colorbar ;
   title(sprintf('u Velocity for t = %.2f',T(ifig)))
   xlabel('x')
   ylabel('y')
   xlim([0 Lx]) ;
   ylim([0 Ly]) ;
   caxis([-2.5 2.5]) ;

   examFig2 = figure(2) ;
   subplot(3,2,ifig);
   pcolor(xc,yf,v0') ;
   shading interp  ;
   colormap jet ;
   colorbar ;
   title(sprintf('v Velocity for t = %.2f',T(ifig)))
   xlabel('x')
   ylabel('y')
   xlim([0 Lx]) ;
   ylim([0 Ly]) ;
   caxis([-2.5 2.5]) ;

   examFig3 = figure(3) ;
   subplot(3,2,ifig);
   pcolor(xc3,yc3,Y') ;
   shading interp  ;
   colormap hot ;
   colorbar ;
   title(sprintf('Y for t = %.2f',T(ifig)))
   xlabel('x')
   ylabel('y')
   xlim([0 Lx]) ;
   ylim([0 Ly]) ;
```

```
        caxis([0 1]) ;

        ifig = ifig + 1 ;

    end

end

examFig4 = figure(4) ;
plot(time,F) ;
%title('Tangental Force vs time')
xlabel('Time, t')
ylabel('Tangental Force, F(t)')
xlim([0 10]) ;
ylim([-0.05 .15]) ;
grid on

examFig5 = figure(5) ;
plot(time,A) ;
%title('Chemical Species Leaving Chamber vs time')
xlabel('Time, t')
ylabel('Chemical Species Leaving Chamber, A(t)')
xlim([0 10]) ;
ylim([0 .85]) ;
grid on
```

7.4 Embedded Functions

Below list the numerous functions implemented in Task 1 and Task 2.

7.4.1 bc_u.m

```
function [u] = bc_u(u,t)
%
% bc_u applies the boundary conditions for the u velocity on a staggered mesh with at least second order accuracy
in space
%
% Input:
%   u: 2D array of staggered mesh values of u
%   t: time at which to apply the boundary conditions

% Ouput:
%   u: 2D array of staggered mesh values of u with boundary conditions applied

% Global Variables
global xf yc

U1 = (1+sin(2*pi/3*t+pi/4)) ;
U2 = 13/20*(1+sin(2*pi/5*t-pi/4)) ;
U3 = 3/2*(1+sin(3*pi/4*t+pi/5)) ;

alpha1 = pi/3*sin(pi*t) ;
alpha2 = 5*pi/6 + pi/4*sin(3*pi/5*t) ;
alpha3 = 2*pi/5 + pi/3*sin(2*pi/5*t) ;

z1 = @(y) (y-1)/.75 ;
z2 = @(y) (y-.5)/1.25 ;
z3 = @(x) (x-1.25)/.5 ;

f = @(z) 6*z*(1-z) ;

   for j = 1:length(yc)
      % Inlet 1
      if yc(j) >= 1 && yc(j) <= 1.75
         u(1,j) = U1*f(z1(yc(j)))*cos(alpha1) ;
      else
         u(1,j) = 0 ; % Left
      end
   end

   for j = 1:length(yc)
      % Inlet 2
      if yc(j) >= .5 && yc(j) <= 1.75
         u(end,j) = U2*f(z2(yc(j)))*cos(alpha2) ;
      else
```

```
        u(end,j) = 0 ; % Right
    end
  end

  for i = 1:length(xf)
    % Inlet 3
    if xf(i) >= 1.25 && xf(i) <= 1.75
      u(i,1) = 2*(U3*f(z3(xf(i))).*cos(alpha3))-u(i,2) ;
    else
      u(i,1) = -u(i,2) ; % Bottom
    end
  end

  for i = 1:length(xf)
    % Outlet
    if xf(i) >= 1.5 && xf(i) <= 2.75
      u(i,end) = u(i,end-1) ;
    else
      u(i,end) = -u(i,end-1) ; % Top
    end
  end

end
```

7.4.2 bc_v.m

```
function [v] = bc_v(v,t)
%
% bc_v applies the boundary conditions for the v velocity on a staggered mesh with at least second order accuracy in space
%
% Input:
%   v: 2D array of staggered mesh values of u
%   t: time at which to apply the boundary conditions

% Ouput:
%   v: 2D array of staggered mesh values of v with boundary conditions applied

% Global Variables
global xc yf


U1 = (1+sin(2*pi/3*t+pi/4)) ;
U2 = 13/20*(1+sin(2*pi/5*t-pi/4)) ;
U3 = 3/2*(1+sin(3*pi/4*t+pi/5)) ;

alpha1 = pi/3*sin(pi*t) ;
alpha2 = 5*pi/6 + pi/4*sin(3*pi/5*t) ;
alpha3 = 2*pi/5 + pi/3*sin(2*pi/5*t) ;

z1 = @(y) (y-1)/.75 ;
z2 = @(y) (y-.5)/1.25 ;
z3 = @(x) (x-1.25)/.5 ;

f = @(z) 6*z*(1-z) ;

   for j = 1:length(yf)
      % Inlet 1
      if yf(j) >= 1 && yf(j) <= 1.75
         v(1,j) = 2*U1*f(z1(yf(j)))*sin(alpha1) - v(2,j) ;
      else
         v(1,j) = -v(2,j) ;
      end
   end

   for j = 1:length(yf)
      % Inlet 2
      if yf(j) >= .5 && yf(j) <= 1.75
         v(end,j) = 2*U2*f(z2(yf(j)))*sin(alpha2) - v(end-1,j) ;
      else
         v(end,j) = -v(end-1,j) ; % Right
      end
```

```
    end

    for i = 1:length(xc)
        % Inlet 3
        if xc(i) >= 1.25 && xc(i) <= 1.75
            v(i,1) = U3*f(z3(xc(i))).*sin(alpha3) ;
        else
            v(i,1) = 0 ; % Bottom
        end
    end

    for i = 1:length(xc)
        % Outlet
        if xc(i) >= 1.5 && xc(i) <= 2.75
            v(i,end) = (4/3)*v(i,end-1)-(1/3)*v(i,end-2) ;
        else
            v(i,end) = 0 ; % Top
        end
    end

end
```

7.4.3 bc_Y3.m

```
function [Y] = bc_Y3(Y,t)

% applies the boundary conditions for the mass fraction Y on a cell centered mesh
% with 3 ghost cell layers using at least second order accuracy in space

% Inputs:
% Y: 2D array of cell centered values of Y
% t: time at which to apply the boundary conditions

% Outputs:
% Y: 2D array of cell centered values of Y with boundary conditions applied

global xc3 yc3

[M] = size(Y,1) - 6 ;
[N] = size(Y,2) - 6 ;

if mod(t,4) < 2
   f_Y1 = 1 ;
else
   f_Y1 = 0 ;
end

if mod(t,3) < 2
   f_Y2 = 1 ;
else
   f_Y2 = 0 ;
end

if mod(t,2) < 1.5
   f_Y3 = 0 ;
else
   f_Y3 = 1 ;
end

for j = 1:length(yc3)

   % Left Boundary
   if yc3(j) >= 1 && yc3(j)  <= 1.75
      Y(1,j) = 2*f_Y1 - Y(6,j) ;
      Y(2,j) = 2*f_Y1 - Y(5,j) ;
      Y(3,j) = 2*f_Y1 - Y(4,j) ;
   else
      Y(1,j) = Y(6,j) ;
      Y(2,j) = Y(5,j) ;
      Y(3,j) = Y(4,j) ;
```

```
    end


    % Right Boundary
    if yc3(j) >= .5 && yc3(j)  <= 1.75
        Y(M+6,j) = 2*f_Y2 - Y(M+1,j) ;
        Y(M+5,j) = 2*f_Y2 - Y(M+2,j) ;
        Y(M+4,j) = 2*f_Y2 - Y(M+3,j) ;
    else
        Y(M+6,j) = Y(M+1,j) ;
        Y(M+5,j) = Y(M+2,j) ;
        Y(M+4,j) = Y(M+3,j) ;

    end

end

for i = 1:length(xc3)

    % Top Boundary
    if xc3(i) >= 1.5 && xc3(i)  <= 2.75
        Y(i,N+6) = Y(i,N+1) ;
        Y(i,N+5) = Y(i,N+2) ;
        Y(i,N+4) = Y(i,N+3) ;
    else
        Y(i,N+6) = Y(i,N+1) ;
        Y(i,N+5) = Y(i,N+2) ;
        Y(i,N+4) = Y(i,N+3) ;
    end

    % Bottom Boundary
    if xc3(i) >= 1.25 && xc3(i)  <= 1.75
        Y(i,1) = 2*f_Y3 - Y(i,6) ;
        Y(i,2) = 2*f_Y3 - Y(i,5) ;
        Y(i,3) = 2*f_Y3 - Y(i,4) ;
    else
        Y(i,1) = Y(i,6) ;
        Y(i,2) = Y(i,5) ;
        Y(i,3) = Y(i,4) ;
    end

end

end
```

7.4.4 bcCN1_u.m

```
function [a,b,c,d] = bcCN1_u(a,b,c,d,t)
%
% bcCN1_u applies the boundary conditions for a, b, c, and/or d for step 1 of the ADI method for Eq. (1)
%       on a staggered mesh with at least second order accuracy in space
%
% Input:
%   a: 2D array of staggered mesh values of a for ADI step 1 of Eq. (1)
%   b: 2D array of staggered mesh values of b for ADI step 1 of Eq. (1)
%   c: 2D array of staggered mesh values of c for ADI step 1 of Eq. (1)
%   d: 2D array of staggered mesh values of d for ADI step 1 of Eq. (1)
%   t: time at which to apply the boundary conditions

% Ouput:
%   a: 2D array of staggered mesh values of a with boundary conditions applied for ADI step 1 of Eq. (1)
%   b: 2D array of staggered mesh values of b with boundary conditions applied for ADI step 1 of Eq. (1)
%   c: 2D array of staggered mesh values of c with boundary conditions applied for ADI step 1 of Eq. (1)
%   d: 2D array of staggered mesh values of d with boundary conditions applied for ADI step 1 of Eq. (1)

% Global Variables
global yc

U1 = (1+sin(2*pi/3*t+pi/4)) ;
alpha1 = pi/3*sin(pi*t) ;
z1 = @(y) (y-1)/.75 ;

U2 = 13/20*(1+sin(2*pi/5*t-pi/4)) ;
alpha2 = 5*pi/6 + pi/4*sin(3*pi/5*t) ;
z2 = @(y) (y-.5)/1.25 ;

f = @(z) 6*z*(1-z) ;

   for j = 1:length(yc)
      % Inlet 1
      if yc(j) >= 1 && yc(j) <= 1.75
         d(2,j) = d(2,j)-a(2,j)*U1*f(z1(yc(j)))*cos(alpha1) ;
         a(2,j) = 0 ;
      else
         a(2,j) = 0 ;
      end
   end

   for j = 1:length(yc)
      % Inlet 2
      if yc(j) >= .5 && yc(j) <= 1.75
         d(end-1,j) = d(end-1,j) - c(end-1,j)*U2*f(z2(yc(j)))*cos(alpha2) ;
         c(end-1,j) = 0 ;
```

```
      else
         c(end-1,j) = 0 ;
      end
   end

end
```

7.4.5 bcCN1_v.m

```
function [a,b,c,d] = bcCN1_v(a,b,c,d,t)
%
% bcCN1_v applies the boundary conditions for a, b, c, and/or d for step 1 of the ADI method for Eq. (1)
%       on a staggered mesh with at least second order accuracy in space
%
% Input:
%   a: 2D array of staggered mesh values of a for ADI step 1 of Eq. (1)
%   b: 2D array of staggered mesh values of b for ADI step 1 of Eq. (1)
%   c: 2D array of staggered mesh values of c for ADI step 1 of Eq. (1)
%   d: 2D array of staggered mesh values of d for ADI step 1 of Eq. (1)
%   t: time at which to apply the boundary conditions

% Ouput:
%   a: 2D array of staggered mesh values of a with boundary conditions applied for ADI step 1 of Eq. (1)
%   b: 2D array of staggered mesh values of b with boundary conditions applied for ADI step 1 of Eq. (1)
%   c: 2D array of staggered mesh values of c with boundary conditions applied for ADI step 1 of Eq. (1)
%   d: 2D array of staggered mesh values of d with boundary conditions applied for ADI step 1 of Eq. (1)

% Global Variables
global yf

U1 = (1+sin(2*pi/3*t+pi/4)) ;
alpha1 = pi/3*sin(pi*t) ;
z1 = @(y) (y-1)/.75 ;

U2 = 13/20*(1+sin(2*pi/5*t-pi/4)) ;
alpha2 = 5*pi/6 + pi/4*sin(3*pi/5*t) ;
z2 = @(y) (y-.5)/1.25 ;

f = @(z) 6*z*(1-z) ;

   for j = 1:length(yf)
      % Inlet 1
      if yf(j) >= 1 && yf(j) <= 1.75
         d(2,j) = d(2,j)-a(2,j)*2*U1*f(z1(yf(j)))*sin(alpha1) ;
         b(2,j) = b(2,j) - a(2,j) ;
         a(2,j) = 0 ;
      else
         b(2,j) = b(2,j) - a(2,j) ;
         a(2,j) = 0 ;
      end
   end

   for j = 1:length(yf)
      % Inlet 2
      if yf(j) >= .5 && yf(j) <= 1.75
```

```
        d(end-1,j) = d(end-1,j) - c(end-1,j)*2*U2*f(z2(yf(j)))*sin(alpha2) ;
        b(end-1,j) = b(end-1,j) - c(end-1,j) ;
        c(end-1,j) = 0 ;
      else
        b(end-1,j) = b(end-1,j) - c(end-1,j) ;
        c(end-1,j) = 0 ;
      end
    end

end
```

7.4.6 bcCN1_Y3.m

```
function [a,b,c,d] = bcCN1_Y3(a,b,c,d,t)

% applies the boundary conditions for a, b, c, and/or d for step 1 of the ADI
% method for Eq. (3) on a 3 ghost layer cell centered mesh with second order accuracy in space

% Inputs:
% a: 2D array of cell centered mesh values of a for ADI step 1 of Eq. (3)
% b: 2D array of cell centered mesh values of b for ADI step 1 of Eq. (3)
% c: 2D array of cell centered mesh values of c for ADI step 1 of Eq. (3)
% d: 2D array of cell centered mesh values of d for ADI step 1 of Eq. (3)
% t: time at which to apply the boundary conditions

% Outputs:
% a: 2D array of cell centered mesh values of a with boundary conditions applied for ADI step 1 of Eq. (3)
% b: 2D array of cell centered mesh values of b with boundary conditions applied for ADI step 1 of Eq. (3)
% c: 2D array of cell centered mesh values of c with boundary conditions applied for ADI step 1 of Eq. (3)
% d: 2D array of cell centered mesh values of d with boundary conditions applied for ADI step 1 of Eq. (3)

global yc3

[M] = size(a,1) - 6 ;

if mod(t,4) < 2
   f_Y1 = 1 ;
else
   f_Y1 = 0 ;
end

if mod(t,3) < 2
   f_Y2 = 1 ;
else
   f_Y2 = 0 ;
end

if mod(t,2) < 1.5
   f_Y3 = 0 ;
else
   f_Y3 = 1 ;

end

for j = 1:length(yc3)

   % Left Boundary
   if yc3(j) >= 1 && yc3(j)  <= 1.75
      d(4,j) = d(4,j) - a(4,j) * 2*f_Y1 ;
```

```
    b(4,j) = b(4,j) - a(4,j) ;
    a(4,j) = 0 ;
  else
    b(4,j) = b(4,j) + a(4,j) ;
    a(4,j) = 0 ;
  end

  % Right Boundary
  if yc3(j) >= .5 && yc3(j)  <= 1.75
    d(M+3,j) = d(M+3,j) - c(M+3,j) * 2 * f_Y2 ;
    b(M+3,j) = b(M+3,j) - c(M+3,j) ;
    c(M+3,j) = 0 ;
  else
    b(M+3,j) = b(M+3,j) + c(M+3,j) ;
    c(M+3,j) = 0 ;
  end

end
```

7.4.7 bcCN2_u.m

```
function [a,b,c,d] = bcCN2_u(a,b,c,d,t)
%
% bcCN2_u applies the boundary conditions for a, b, c, and/or d for step 2 of the ADI method for Eq. (1) on a staggered mesh with at
%       least second order accuracy in space
%
% Input:
%   a: 2D array of staggered mesh values of a for ADI step 2 of Eq. (1)
%   b: 2D array of staggered mesh values of b for ADI step 2 of Eq. (1)
%   c: 2D array of staggered mesh values of c for ADI step 2 of Eq. (1)
%   d: 2D array of staggered mesh values of d for ADI step 2 of Eq. (1)
%   t: time at which to apply the boundary conditions

% Ouput:
%   a: 2D array of staggered mesh values of a with boundary conditions applied for ADI step 2 of Eq. (1)
%   b: 2D array of staggered mesh values of b with boundary conditions applied for ADI step 2 of Eq. (1)
%   c: 2D array of staggered mesh values of c with boundary conditions applied for ADI step 2 of Eq. (1)
%   d: 2D array of staggered mesh values of d with boundary conditions applied for ADI step 2 of Eq. (1)

% Global Variables
global xf

U3 = 3/2*(1+sin(3*pi/4*t+pi/5)) ;
alpha3 = 2*pi/5 + pi/3*sin(2*pi/5*t) ;
z3 = @(x) (x-1.25)/.5 ;

f = @(z) 6*z*(1-z) ;

   for i = 1:length(xf)
      % Inlet 3
      if xf(i) >= 1.25 && xf(i) <= 1.75
         b(i,2) = b(i,2) - a(i,2) ;
         d(i,2) = d(i,2) - a(i,2)*2*U3*f(z3(xf(i)))*cos(alpha3) ;
         a(i,2) = 0 ;
      else
         b(i,2) = b(i,2) - a(i,2) ;
         a(i,2) = 0 ;
      end
   end

   for i = 1:length(xf)
      % Outlet
      if xf(i) >= 1.5 && xf(i) <= 2.75
         b(i,end-1) = c(i,end-1) + b(i,end-1) ;
         c(i,end-1) = 0 ;
      else
```

```
        b(i,end-1) = b(i,end-1) - c(i,end-1) ;
        c(i,end-1) = 0 ;
      end
    end

end
```

7.4.8 bcCN2_v.m

```
function [a,b,c,d] = bcCN2_v(a,b,c,d,t)
%
% bcCN2_v applies the boundary conditions for a, b, c, and/or d for step 2 of the ADI method for Eq. (1) on a staggered mesh with at
%       least second order accuracy in space
%
% Input:
%   a: 2D array of staggered mesh values of a for ADI step 2 of Eq. (1)
%   b: 2D array of staggered mesh values of b for ADI step 2 of Eq. (1)
%   c: 2D array of staggered mesh values of c for ADI step 2 of Eq. (1)
%   d: 2D array of staggered mesh values of d for ADI step 2 of Eq. (1)
%   t: time at which to apply the boundary conditions

% Ouput:
%   a: 2D array of staggered mesh values of a with boundary conditions applied for ADI step 2 of Eq. (1)
%   b: 2D array of staggered mesh values of b with boundary conditions applied for ADI step 2 of Eq. (1)
%   c: 2D array of staggered mesh values of c with boundary conditions applied for ADI step 2 of Eq. (1)
%   d: 2D array of staggered mesh values of d with boundary conditions applied for ADI step 2 of Eq. (1)

% Global Variables
global xc

U3 = 3/2*(1+sin(3*pi/4*t+pi/5)) ;
alpha3 = 2*pi/5 + pi/3*sin(2*pi/5*t) ;
z3 = @(x) (x-1.25)/.5 ;

f = @(z) 6*z*(1-z) ;

   for i = 1:length(xc)
      % Inlet 3
      if xc(i) >= 1.25 && xc(i) <= 1.75
         d(i,2) = d(i,2)-a(i,2)*U3*f(z3(xc(i))).*sin(alpha3) ;
         a(i,2) = 0 ;
      else
         a(i,2) = 0 ;
      end
   end

   for i = 1:length(xc)
      % Outlet
      if xc(i) >= 1.5 && xc(i) <= 2.75
         a(i,end-1) = a(i,end-1) - (1/3)*c(i,end-1) ;
         b(i,end-1) = b(i,end-1) + (4/3)*c(i,end-1) ;
         c(i,end-1) = 0 ;
      else
         c(i,end-1) = 0 ;
```

```
      end
   end

end
```

7.4.9 bcCN2_Y3.m

```
function [a,b,c,d] = bcCN2_Y3(a,b,c,d,t)

% applies the boundary conditions for a, b, c, and/or d for step 2 of the ADI
% method for Eq. (3) on a 3 ghost layer cell centered with second order accuracy in space

% Inputs:
% a: 2D array of cell centered mesh values of a for ADI step 2 of Eq. (3)
% b: 2D array of cell centered mesh values of b for ADI step 2 of Eq. (3)
% c: 2D array of cell centered mesh values of c for ADI step 2 of Eq. (3)
% d: 2D array of cell centered mesh values of d for ADI
% t: time at which to apply the boundary conditions

% Outputs:
% a: 2D array of cell centered mesh values of a with boundary conditions applied for ADI step 2 of Eq. (3)
% b: 2D array of cell centered mesh values of b with boundary conditions applied for ADI step 2 of Eq. (3)
% c: 2D array of cell centered mesh values of c with boundary conditions applied for ADI step 2 of Eq. (3)
% d: 2D array of cell centered mesh values of d with boundary conditions applied for ADI step 2 of Eq. (3)

global xc3

[N] = size(a,2) - 6;

if mod(t,4) < 2
   f_Y1 = 1 ;
else
   f_Y1 = 0 ;
end

if mod(t,3) < 2
   f_Y2 = 1 ;
else
   f_Y2 = 0 ;
end

if mod(t,2) < 1.5
   f_Y3 = 0 ;
else
   f_Y3 = 1 ;
end

for i = 1:length(xc3)

   % Top Boundary
   if xc3(i) >= 1 && xc3(i)  <= 1.75
      b(i,N+3) = b(i,N+3) + c(i,N+3) ;
      c(i,N+3) = 0 ;
```

```
    else
        b(i,N+3) = b(i,N+3) + c(i,N+3) ;
        c(i,N+3) = 0 ;
    end


    % Bottom Boundary
    if xc3(i) >= 1.25 && xc3(i)  <= 1.75
        d(i,4) = d(i,4) - a(i,4) * 2 * f_Y3 ;
        b(i,4) = b(i,4) - a(i,4) ;
        a(i,4) = 0 ;
    else
        b(i,4) = b(i,4) + a(i,4) ;
        a(i,4) = 0 ;
    end

end
```

7.4.10 calc_adYdx_WENO_2D.m

```
function [f] = calc_adYdx_WENO_2D(Y,a)
%
% calc adYdx WENO 2D that calculates a ∂Y/∂x using the upwind biased WENO-5 method on a 2D cell centered
mesh with 3 ghost layers
%
% Input:
%   Y: cell centered Y incl. 3 ghost layers
%   a: cell centered a incl. 3 ghost layers

% Ouput:
%   f: cell centered a ∂Y/∂x calculated using the WENO-5 method incl. 3 ghost layers

global h

% delta^+ u_i = u_i+1 - u_i
% delta^- u_i = u_i - u_i-1
% delta^- delta^+ u_i = u_i+1 - 2*u_i + u_i-1

[M,N] = size(Y) ;

M = M - 6 ;
N = N - 6 ;

f = zeros(M+6,N+6) ;

for j = 4:N+3

   for i = 4:M+3

      if a(i,j) > 0

         a1 = (Y(i-1,j)-2*Y(i-2,j)+Y(i-3,j))/h ;
         b = (Y(i,j)-2*Y(i-1,j)+Y(i-2,j))/h ;
         c = (Y(i+1,j)-2*Y(i,j)+Y(i-1,j))/h ;
         d = (Y(i+2,j)-2*Y(i+1,j)+Y(i,j))/h ;
         psi = psiWENO(a1,b,c,d) ;

         f(i,j) = a(i,j) * ((1/(12*h)) * ( -1*(Y(i-1,j)-Y(i-2,j)) + 7*(Y(i,j)-Y(i-1,j)) + 7*(Y(i+1,j)-Y(i,j)) -
(Y(i+2,j)-Y(i+1,j)) ) - psi) ;

      else

         a1 = (Y(i+3,j)-2*Y(i+2,j)+Y(i+1,j))/h ;
         b = (Y(i+2,j)-2*Y(i+1,j)+Y(i,j))/h ;
         c = (Y(i+1,j)-2*Y(i,j)+Y(i-1,j))/h ;
         d = (Y(i,j)-2*Y(i-1,j)+Y(i-2,j))/h ;
```

```
        psi = psiWENO(a1,b,c,d) ;

        f(i,j) = a(i,j) * ((1/(12*h)) * ( -1*(Y(i-1,j)-Y(i-2,j)) + 7*(Y(i,j)-Y(i-1,j)) + 7*(Y(i+1,j)-Y(i,j)) -
(Y(i+2,j)-Y(i+1,j)) ) + psi) ;

      end

   end

end

end
```

7.4.11 calc_bdYdy_WENO_2D.m

```
function [f] = calc_bdYdy_WENO_2D(Y,b)
%
% calc adYdx WENO 2D that calculates a ∂Y/∂y using the upwind biased WENO-5 method on a 2D cell centered
mesh with 3 ghost layers
%
% Input:
%   Y: cell centered Y incl. 3 ghost layers
%   b: cell centered b incl. 3 ghost layers

% Ouput:
%   f: cell centered a ∂Y/∂x calculated using the WENO-5 method incl. 3 ghost layers

global h

% delta^+ u_i = u_i+1 - u_i
% delta^- u_i = u_i - u_i-1
% delta^- delta^+ u_i = u_i+1 - 2*u_i + u_i-1

[M,N] = size(Y) ;

M = M - 6 ;
N = N - 6 ;

f = zeros(M+6,N+6) ;

for j = 4:N+3

   for i = 4:M+3

      if b(i,j) > 0

         a = (Y(i,j-1)-2*Y(i,j-2)+Y(i,j-3))/h ;
         b1 = (Y(i,j)-2*Y(i,j-1)+Y(i,j-2))/h ;
         c = (Y(i,j+1)-2*Y(i,j)+Y(i,j-1))/h ;
         d = (Y(i,j+2)-2*Y(i,j+1)+Y(i,j))/h ;
         psi = psiWENO(a,b1,c,d) ;

         f(i,j) = b(i,j) * ((1/(12*h)) * ( -1*(Y(i,j-1)-Y(i,j-2)) + 7*(Y(i,j)-Y(i,j-1)) + 7*(Y(i,j+1)-Y(i,j)) -
(Y(i,j+2)-Y(i,j+1)) ) - psi) ;

      else

         a = (Y(i,j+3)-2*Y(i,j+2)+Y(i,j+1))/h ;
         b1 = (Y(i,j+2)-2*Y(i,j+1)+Y(i,j))/h ;
         c = (Y(i,j+1)-2*Y(i,j)+Y(i,j-1))/h ;
         d = (Y(i,j)-2*Y(i,j-1)+Y(i,j-2))/h ;
```

```
        psi = psiWENO(a,b1,c,d) ;

        f(i,j) = b(i,j) * ((1/(12*h)) * ( -1*(Y(i,j-1)-Y(i,j-2)) + 7*(Y(i,j)-Y(i,j-1)) + 7*(Y(i,j+1)-Y(i,j)) -
(Y(i,j+2)-Y(i,j+1)) ) + psi) ;

      end

   end

end

end
```

7.4.12 calcDivV.m

```
function [divV] = calcDivV(u,v)
%
% calcDivV calculates the divergence of a staggered velocity field ∇ · ~v in the interior of
% a cell centered mesh with 1 ghost layer
%
% Input:
%   u: staggered velocity u* with outlet correction applied
%   v: staggered velocity v* with outlet correction applied

% Ouput:
%   divV: velocity divergence on a cell centered mesh with 1 ghost layer

% Global Variables
global h

M = size(u,1)-1 ;
N = size(u,2)-2 ;

divV = zeros(M+2, N+2) ;

% Calculate velocity divergence in interior cells
for j = 2:N+1
   for i = 2:M+1
      divV(i,j) = (u(i,j)-u(i-1,j))/h + (v(i,j)-v(i,j-1))/h ;
   end
end

end
```

7.4.13 calcDt561.m

```
function [dt,outputFlag] = calcDt561(t,outputTime,u,v)
%
% calcDT561 calculates the time step Δt defined as Δt = CF L· Δtmax, potentially adjusted
% to reach a requested output time tout. Here Δtmax is the maximum stable time step of the FTCS
method
%
% Input:
%   t: current time t^n
%   outputTime: next requested time for output tout
%   u: u velocity on a staggered mesh at t^n
%   v: v velocity on a staggered mesh at t^n

% Ouput:
%   dt: time step ΔT to perform
%   outputFlag: flag, set to 1 if dt was adjusted to reach outputTime, else set to 0

% Global Variables
global h CFL

a = max(max(abs(u))) ;
b = max(max(abs(v))) ;
c = a ;
d = b ;

if a > 0
  %dtu = (h./(2*a+b)) ; % hyperbolic for u
  dtu = (h./(a+b)) ; % hyperbolic for u
end

if a < 0
  %dtu = (h./(2*a+b)) ; % hyperbolic for u
  dtu = (h./(a+b)) ; % hyperbolic for u
end

if b > 0
  %dtv = (h./(c+2*d)) ; % hyperbolic for v
  dtv = (h./(c+d)) ; % hyperbolic for v
end

if b < 0
  %dtv = (h./(c+2*d)) ; % hyperbolic for
  dtv = (h./(c+d)) ; % hyperbolic for v
```

```
end

dt = min(CFL*dtu,CFL*dtv) ; % min of hyperbolic

% Check if adjusted time step is needed to reach output time
   if (t < outputTime) && (t + dt >= outputTime)
      dt = outputTime - t ;
      outputFlag = 1 ;
   else
      outputFlag = 0 ;
   end

end
```

7.4.14 correctOutlet.m

```
function [u,v] = correctOutlet(u,v)
%
% correctOutlet calculates the outlet correction velocity ucorr and
% corrects the outlet velocities to enforce global mass conservation
%
% Input:
%   u: staggered velocity u*
%   v: staggered velocity v*

% Ouput:
%   u: staggered velocity u* with potential outlet correction applied
%   v: staggered velocity v* with potential outlet correction applied

% Global Variables
global h xc

M = size(u,1) - 1 ;
N = size(u,2) - 2 ;

   ucorr = (sum(u(1,2:N+1)*h)-sum(u(M+1,2:N+1)*h)+sum(v(2:M+1,1)*h)-sum(v(2:M+1,N+1)*h))/1.25
;

   for i = 1:length(xc)
      % Outlet
      if xc(i) >= 1.5 && xc(i) <= 2.75
         v(i,end) = v(i,end) + ucorr ;
      end
   end

end
```

7.4.15 hyperbolic_uv_2D.m

```
function [Hu,Hv] = hyperbolic_uv_2D(u,v)
%
% hyperbolic_uv_2D that calculates the hyperbolic terms of the Burgers  Eqs. (1) and (2) (highlighted in red) using second-order central differences
% on a staggered mesh as discussed in class when moved to the right hand side of the equations
%
% Input:
%   u: u velocity on a staggered mesh at time t^n
%   v: v velocity on a staggered mesh at time t^n

% Ouput:
%   Hu: hyperbolic terms of Eq. 1 on a staggered mesh
%   Hv: hyperbolic terms of Eq. 2 on a staggered mesh

% Global Variables
global h

% Number of nodes in mesh
[M,N] = size(u) ;

M = M-1 ;
N = N-2 ;

Hu = zeros(M+1,N+2) ;

for j = 2:N+1

   for i = 2:M

      Hu(i,j) = -(((.5*(u(i+1,j) + u(i,j)))^2 - (.5*(u(i,j) + u(i-1,j)))^2 ) /h + (((.5*(u(i,j) + u(i,j+1))) * (.5*( v(i,j) + v(i+1,j)))) - ((.5*(u(i,j-1) + u(i,j))) * (.5*( v(i,j-1) + v(i+1,j-1))))) /h) ;

   end

end

Hv = zeros(M+2,N+1) ;

for j = 2:N

   for i = 2:M+1

      Hv(i,j) = -(((.5*(v(i,j+1) + v(i,j)))^2 - (.5*(v(i,j)+v(i,j-1)))^2) /h + (((.5*(u(i,j) + u(i,j+1))) * (.5*( v(i,j) + v(i+1,j)))) - ((.5*(u(i-1,j) + u(i-1,j+1))) * (.5*( v(i,j) + v(i-1,j))))) /h) ;

   end
```

```
end

end
```

7.4.16 hyperbolic_Y_WENO_2D.m

```
function [HY] = hyperbolic_Y_WENO_2D(Y,u0,v0,u1,v1,dt)
%
% hyperbolic_uv_2D that calculates the hyperbolic terms of the Burgers  Eqs. (1) and (2) (highlighted in red) using second-order central differences
% on a staggered mesh as discussed in class when moved to the right hand side of the equations
%
% Input:
%   Y: mass fraction Y on a cell centered mesh with 3 ghost layers at time t^n
%   u0: u velocity on a staggered mesh at time t^n
%   v0: v velocity on a staggered mesh at time t^n
%   u1: u velocity on a staggered mesh at time t^n+1
%   v1: v velocity on a stag

% Ouput:
%   HY: hyperbolic terms HY of Eq. (3) on a cell centered mesh with 3 ghost layers

% Global Variables
global t

% u0 = u^n ; v0 = v^n
% u1 = u^n+1 ; v1 = v^n+1
% u12 = u^n+1/2 ; v12 = v^n+1/2

% Initalize
M = size(Y,1) - 6 ;
N = size(Y,2) - 6 ;

a0 = zeros(M+6,N+6) ;
b0 = zeros(M+6,N+6) ;
a1 = zeros(M+6,N+6) ;
b1 = zeros(M+6,N+6) ;
a2 = zeros(M+6,N+6) ;
b2 = zeros(M+6,N+6) ;

% u0 and v0
a0(4:M+3,3:N+4) = .5*(u0(1:M,:) + u0(2:M+1,:)) ;
b0(3:M+4,4:N+3) = .5*(v0(:,1:N) + v0(:,2:N+1)) ;

f0u = calc_adYdx_WENO_2D(Y,a0) ;
f0v = calc_bdYdy_WENO_2D(Y,b0) ;

% Inital BC at t
Y = bc_Y3(Y,t) ;

% Solve for Y1
Y1 = Y-(f0u + f0v)*dt ;
```

```
% BC for Y1 at t + dt
Y1 = bc_Y3(Y1,t+dt) ;

% u1 and v1
a1(4:M+3,3:N+4) = .5*(u1(1:M,:) + u1(2:M+1,:)) ;
b1(3:M+4,4:N+3) = .5*(v1(:,1:N) + v1(:,2:N+1)) ;

f1u = calc_adYdx_WENO_2D(Y1,a1) ;
f1v = calc_bdYdy_WENO_2D(Y1,b1) ;

% Solve for Y2
Y2 = Y1 + (3/4)*(f0u + f0v)*dt - (1/4)*(f1u + f1v)*dt ;

% BC for Y2 at t + dt/2
Y2 = bc_Y3(Y2,t+.5*dt) ;

% u12 and v12
a2(4:M+3,3:N+4) = .5*(.5*(u0(1:M,:) + u0(2:M+1,:)) + .5*(u1(1:M,:) + u1(2:M+1,:))) ;
b2(3:M+4,4:N+3) = .5*(.5*(v0(:,1:N) + v0(:,2:N+1)) + .5*(v1(:,1:N) + v1(:,2:N+1))) ;

f2u = calc_adYdx_WENO_2D(Y2,a2) ;
f2v = calc_bdYdy_WENO_2D(Y2,b2) ;

% Solve for Ys
Ys = Y2 + (1/12)*(f0u + f0v)*dt + (1/12)*(f1u + f1v)*dt  - (2/3)*(f2u + f2v)*dt ;

% BC for Ys at t + dt
Ys = bc_Y3(Ys,t+dt) ;

% Calculate HY
HY = (Ys - Y)/dt ;

end
```

7.4.17 myGaussSeidel.m

```
function [phi] = myGaussSeidel(phi,f,h,niter)
%
% myGaussSeidel solves the two-dimensional Poisson Equation
%
% Input:
%   phi: 2D array of cell centered PDE dsolution variable including ghost cells
%   f: 2D array of cell centered PDE right hand side values including
%      ghost cells (even though these are not necessarily defined)
%   h: Mesh spacing (the equidistant mesh spacing in the x and y-direction is h)
%   niter: Integer number of Gauss Seidel iterations to be performed
% Ouput:
%   phi: 2D array of cell centered solution variable including ghost cells,
%        containing the solution after niter iterations of Gauss Seidel

[M,N] = size(phi) ; % Yield size of M and N

% Loop over an integer for Gauss Seidel
for k = 1:niter

   phi = bcGS(phi) ; % BC

   % Loop over Y Values after BC
   for j = 2:(N-1)

      % Loop over X Values after BC
      for i = 2:(M-1)

         phi(i,j) = (phi(i-1,j) + phi(i+1,j) + phi(i,j-1) + phi(i,j+1))/4 - ((h^2)*f(i,j))/4 ;

      end

   end

   phi = bcGS(phi) ; % Update BC

end

end
```

7.4.18 myMultigrid.m

```
function [phi] = myMultigrid(phi,f,h)
%
% myMultigrid
%
% Input:
%   phi: 2D array of cell centered PDE dsolution variable including ghost cells
%   f: 2D array of cell centered PDE right hand side values including
%      ghost cells (even though these are not necessarily defined)
%   h: Mesh spacing (the equidistant mesh spacing in the x and y-direction is h)
% Ouput:
%   phi: 2D array of cell centered PDE dsolution variable including updated ghost cells

   % Yield size of M and N
   M = size(phi,1)-2 ;
   N = size(phi,2)-2 ;

   phi = myGaussSeidel(phi,f,h,1) ;

   if mod(M,2) == 0 && mod(N,2) == 0
      rh = myResidual(phi,f,h) ;
      r2h = myRestrict(rh) ;
      e2h = zeros(M/2+2,N/2+2) ;
      e2h = myMultigrid(e2h,r2h,2*h) ;
      eh = myProlong(e2h) ;
      phi = phi + eh ;
      phi = bcGS(phi) ;
      phi = myGaussSeidel(phi,f,h,1) ;
   else
      phi = myGaussSeidel(phi,f,h,3) ;
   end

end
```

7.4.19 myPoisson.m

```
function [phi] = myPoisson(phi,f,h,nIterMax,epsilon)
%
% myMultigrid
%
% Input:
%   phi: 2D array of cell centered PDE dsolution variable including ghost cells
%   f: 2D array of cell centered PDE right hand side values including
%      ghost cells (even though these are not necessarily defined)
%   h: Mesh spacing (the equidistant mesh spacing in the x and y-direction is h)
%   nIterMax: integer number of maximum V-cycle iterations to be performed
%   epsilon: convergence threshold for infinity norm of the residual
% Ouput:
%   phi: 2D array of cell centered PDE dsolution variable including updated ghost cells

   for i = 1:nIterMax

      [phi] = myMultigrid(phi,f,h) ;
      [r] = myResidual(phi,f,h) ;
      % Yield size of M and N
      M = size(phi,1)-2 ; N = size(phi,2)-2 ;
      Linf = max(max(abs(r(2:M+1,2:N+1)))) ; % Infite Error Norm

      if Linf < epsilon
         phi = phi ;
         break ;
      end

   end

end
```

7.4.20 myProlong.m

```
function [eh] = myProlong(e2h)
%
% myProlong
%
% Input:
%   e2h: Error on a coarse cell centered mesh
% Ouput:
%    eh: Prolonged error on a fine cell centered mesh

   % Yield size of M and N
   M = size(e2h,1)-2 ;
   N = size(e2h,2)-2 ;

   % Increase the size of the eh vector to be fine
   eh = zeros(2*M+2,2*N+2) ;

   for j = 2:(N)+1

      for i = 2:(M)+1

         eh(2*i-2,2*j-2) = e2h(i,j) ;
         eh(2*i-1,2*j-2) = e2h(i,j) ;
         eh(2*i-2,2*j-1) = e2h(i,j) ;
         eh(2*i-1,2*j-1) = e2h(i,j) ;

      end

   end

   eh = bcGS(eh) ;

end
```

7.4.21 myResidual.m

```
function [r] = myResidual(phi,f,h)
%
% myResidual calculates the residual r for the two-dimensional Poisson equation
%
% Input:
%   phi: 2D array of cell centered PDE dsolution variable including ghost cells
%   f: 2D array of cell centered PDE right hand side values including
%      ghost cells (even though these are not necessarily defined)
%   h: Mesh spacing (the equidistant mesh spacing in the x and y-direction is h)
% Ouput:
%   r: 2D array of residual calculated on cell centered mesh including ghost cells,
%      even though the ghost cell values need not be set and can be left

[M,N] = size(phi) ; % Yield size of M and N

r = zeros(M,N) ;

for j = 2:(N-1)

   % Loop over X Values after BC
   for i = 2:(M-1)

      r(i,j) =  f(i,j) - ((phi(i+1,j) - 2*phi(i,j) + phi(i-1,j))/h^2 + (phi(i,j+1) - 2*phi(i,j) + phi(i,j-1))/h^2) ;

   end

end

end
```

7.4.22 myRestrict.m

```
function [r2h] = myRestrict(rh)
%
% myRestrict
%
% Input:
%    rh: Residual h on a fine cell centered mesh
% Ouput:
%   r2h: Residual 2h on a coarse cell centered mesh

   % Yield size of M and N
   M = size(rh,1)-2 ;
   N = size(rh,2)-2 ;

   % Decrease the size of the r2h vector to be coarse
   r2h = zeros((M/2)+2,(N/2)+2) ;

   % Average fine mesh values to coarse mesh
   for j = 2:(N/2)+1

      for i = 2:(M/2)+1

         r2h(i,j) = .25*(rh(2*i-2,2*j-2)+rh(2*i-1,2*j-2)+rh(2*i-2,2*j-1)+rh(2*i-1,2*j-1)) ;

      end

   end

end
```

7.4.23 mySolveTriDiag.m

```
function [x] = mySolveTriDiag(a,b,c,d)
%
% mySolveTriDiag solves a tridiagonal linear
% system using Gaussian elimination
%
% Input:
%   a: Lower diagonal (1D Column Vector)
%   b: Main diagonal (1D Column Vector)
%   c: Upper diagonal (1D Column Vector)
%   d: Right hand side (1D Column Vector)
% Ouput:
%   x: Solution (1D Column Vector)

% check that length of all vectors are equal
if length(a) ~= length(b) || length(a) ~= length(c) || length(a) ~= length(d)
   error('Vectors lengths must be the same')
end

P = length(b) ;

% Perform elimination from rows 2 to P
for i = 2:P
   b(i) = b(i) - c(i-1)*a(i)/(b(i-1)) ;
   d(i) = d(i) - d(i-1)*a(i)/(b(i-1)) ;
end

% Last row back Substituion
x = zeros(P,1) ; % Initalize vector
x(P) = d(P)/b(P) ;

% Back Substituion from P-1 to 1
for i = (P-1):-1:1
  x(i) = (d(i) - c(i)*x(i+1))/b(i) ;
end

end
```

7.4.24 parabolic_CN1_2D_u.m

```
function [u] = parabolic_CN1_2D_u(u,Qu,dt)
%
% parbolic_CN1_2D_u performs step 1 of the Crank-Nicolson ADI method for Eq. (1) on a staggered mesh
%
% Input:
%   u: 2D array of staggered mesh values of u^n
%   Qu: 2D array of staggered mesh values of Qu
%   dt: overall time step Δt

% Ouput:
%   u: 2D array of staggered mesh values of u^n+1/2 with boundary conditions applied

% Global Variables
global Re t h a b c d xf yc

% Number of nodes in mesh
M = length(xf)-1 ;
N = length(yc)-2 ;

dx = (1/Re)*(dt/(h^2)) ;
dy = dx ;

% Compute coefficients
a = -(dx/2)*ones(M+1,N+2) ;
b = (1+2*(dx/2))*ones(M+1,N+2) ;
c = -(dx/2)*ones(M+1,N+2) ;

d = zeros(M+1,N+2) ;

u_old = u ;

for j=2:N+1
   for i = 2:M

      d(i,j) = (dy/2)*u_old(i,j+1) + (1-2*(dy/2))*u_old(i,j) + (dy/2)*u_old(i,j-1) + (dt/2)*Qu(i,j) ;

   end

end

[a,b,c,d] = bcCN1_u(a,b,c,d,t+(dt/2)) ;

for j=2:N+1

   u(2:M,j) = mySolveTriDiag(a(2:M,j),b(2:M,j),c(2:M,j),d(2:M,j)) ;
```

```
end

[u] = bc_u(u,t+dt/2) ;

end
```

7.4.25 parabolic_CN1_2D_v.m

```
function [v] = parabolic_CN1_2D_v(v,Qv,dt)
%
% parbolic_CN1_2D_v performs step 1 of the Crank-Nicolson ADI method for Eq. (2) on a staggered mesh
%
% Input:
%   v: 2D array of staggered mesh values of v^n
%   Qv: 2D array of staggered mesh values of Qv
%   dt: overall time step Δt

% Ouput:
%   v: 2D array of staggered mesh values of u^n+1/2 with boundary conditions applied

% Global Variables
global Re t h a b c d xc yf

% Number of nodes in mesh
M = length(xc)-1 ;
N = length(yf)-2 ;

dx = (1/Re)*(dt/(h^2)) ;
dy = dx ;

% Compute coefficients
a = -(dx/2)*ones(M+1,N+2) ;
b = (1+2*(dx/2))*ones(M+1,N+2) ;
c = -(dx/2)*ones(M+1,N+2) ;

d = zeros(M+1,N+2) ;

v_old = v ;

for j=2:N+1
   for i = 2:M

      d(i,j) = (dy/2)*v_old(i,j+1) + (1-2*(dy/2))*v_old(i,j) + (dy/2)*v_old(i,j-1) + (dt/2)*Qv(i,j) ;

   end

end

[a,b,c,d] = bcCN1_v(a,b,c,d,t+(dt/2)) ;

for j=2:N+1

   v(2:M,j) = mySolveTriDiag(a(2:M,j),b(2:M,j),c(2:M,j),d(2:M,j)) ;
```

```
end

[v] = bc_v(v,t+dt/2) ;

end
```

7.4.26 parabolic_CN1_2D_Y3.m

```
function [Y] = parabolic_CN1_2D_Y3(Y,QY,dt)

% performs step 1 of the Crank-Nicolson ADI method for Eq. (3)
% on a cell centered mesh with 3 ghost cell layers

% Inputs:
% Y: 2D array of cell centered mesh values of Y^n
% QY: 2D array of cell centered mesh values of QY
% dt: overall time step Δt

% Outputs:
% Y: 2D array of cell centered mesh values of Y^n+1/2 with boundary conditions applied

global Re Sc t h a b c d

[M] = size(Y,1) - 6 ;
[N] = size(Y,2) - 6 ;

dx = (1/(Re*Sc)) * (dt/(h^2)) ;
dy = (1/(Re*Sc)) * (dt/(h^2)) ;
d1 = dx/2 ;
d2 = dy/2 ;

a = -d1 * ones(M+6,N+6) ;
b = (1+2*d1) * ones(M+6,N+6) ;
c = -d1 * ones(M+6,N+6) ;
d = zeros(M+6,N+6) ;

for j = 4:(N+3)
   for i = 4:(M+3)

      d(i,j) = d2*Y(i,j+1) + (1-2*d2)*Y(i,j) + d2*Y(i,j-1) + .5*dt*QY(i,j) ;

   end
end

[a,b,c,d] = bcCN1_Y3(a,b,c,d,t+.5*dt) ;

for j = 4:(N+3)

   Y(4:M+3,j) = mySolveTriDiag(a(4:M+3,j),b(4:M+3,j),c(4:M+3,j),d(4:M+3,j));

end

[Y] = bc_Y3(Y,t+.5*dt);
```

```
end
```

7.4.27 parapolic_CN2_2D_u.m

```
function [u] = parabolic_CN2_2D_u(u,Qu,dt)
%
% parbolic_CN1_2D_u performs step 2 of the Crank-Nicolson ADI method for Eq. (1) on a staggered mesh
%
% Input:
%   u: 2D array of staggered mesh values of u^n
%   Qu: 2D array of staggered mesh values of Qu
%   dt: overall time step Δt

% Ouput:
%   u: 2D array of staggered mesh values of u^n+1 with boundary conditions applied

% Global Variables
global Re t h a b c d xf yc

% Number of nodes in mesh
M = length(xf)-1 ;
N = length(yc)-2 ;

dx = (1/Re)*(dt/(h^2)) ;
dy = dx ;

% Compute coefficients
a = -(dy/2)*ones(M+1,N+2) ;
b = (1+2*(dy/2))*ones(M+1,N+2) ;
c = -(dy/2)*ones(M+1,N+2) ;

d = zeros(M+1,N+2) ;

u_old = u ;

for j=2:N+1
   for i = 2:M

      d(i,j) = (dx/2)*u_old(i+1,j) + (1-2*(dx/2))*u_old(i,j) + (dx/2)*u_old(i-1,j) + (dt/2)*Qu(i,j) ;

   end

end

[a,b,c,d] = bcCN2_u(a,b,c,d,t+dt) ;

for i=2:M

   u(i,2:N+1) = mySolveTriDiag(a(i,2:N+1),b(i,2:N+1),c(i,2:N+1),d(i,2:N+1)) ;
```

```
end

[u] = bc_u(u,t+dt) ;

end
```

7.4.28 parapolic_CN2_2D_v.m

```
function [v] = parabolic_CN2_2D_v(v,Qu,dt)
%
% parbolic_CN1_2D_v performs step 2 of the Crank-Nicolson ADI method for Eq. (2) on a staggered mesh
%
% Input:
%   v: 2D array of staggered mesh values of v^n
%   Qu: 2D array of staggered mesh values of Qu
%   dt: overall time step Δt

% Ouput:
%   v: 2D array of staggered mesh values of u^n+1 with boundary conditions applied

% Global Variables
global Re t h a b c d xc yf

% Number of nodes in mesh
M = length(xc)-1 ;
N = length(yf)-2 ;

dx = (1/Re)*(dt/(h^2)) ;
dy = dx ;

% Compute coefficients
a = -(dy/2)*ones(M+1,N+2) ;
b = (1+2*(dy/2))*ones(M+1,N+2) ;
c = -(dy/2)*ones(M+1,N+2) ;

d = zeros(M+1,N+2) ;

v_old = v ;

for j=2:N+1
   for i = 2:M

      d(i,j) = (dx/2)*v_old(i+1,j) + (1-2*(dx/2))*v_old(i,j) + (dx/2)*v_old(i-1,j) + (dt/2)*Qu(i,j) ;

   end

end

[a,b,c,d] = bcCN2_v(a,b,c,d,t+dt) ;

for i=2:M

   v(i,2:N+1) = mySolveTriDiag(a(i,2:N+1),b(i,2:N+1),c(i,2:N+1),d(i,2:N+1)) ;
```

```
end

[v] = bc_v(v,t+dt) ;

end
```

7.4.29 parapolic_CN2_2D_Y3.m

```
function [Y] = parabolic_CN2_2D_Y3(Y,QY,dt)

% performs step 2 of the Crank-Nicolson ADI method for Eq. (3)
% on a cell centered mesh with 3 ghost cell layers

% Inputs:
% Y: 2D array of cell centered mesh values of Y^n+1/2
% QY: 2D array of cell centered mesh values of QY
% dt: overall time step Δt

% Outputs:
% Y: 2D array of cell centered mesh values of Y^n+1 with boundary conditions applied

global Re Sc t h a b c d

[M] = size(Y,1) - 6 ;
[N] = size(Y,2) - 6 ;

dx = (1/(Re*Sc)) * (dt/(h^2)) ;
dy = (1/(Re*Sc)) * (dt/(h^2)) ;
d1 = dx/2 ;
d2 = dy/2 ;

a = -d2 * ones(M+6,N+6) ;
b = (1+2*d2) * ones(M+6,N+6) ;
c = -d2 * ones(M+6,N+6) ;
d = zeros(M+6,N+6) ;

%for interior points
for j = 4:N+3
   for i = 4:M+3

      d(i,j) = d1*Y(i+1,j) + (1-2*d1)*Y(i,j) + d1*Y(i-1,j) + .5*dt*QY(i,j) ;

   end
end

[a,b,c,d] = bcCN2_Y3(a,b,c,d,t+dt) ;

for i = 4:M+3

   Y(i,2:N+3) = mySolveTriDiag(a(i,2:N+3),b(i,2:N+3),c(i,2:N+3),d(i,2:N+3)) ;

end

[Y] = bc_Y3(Y,t+dt) ;
```

end

7.4.30 projectV.m

```
function [u,v] = projectV(u,v,phi,dt)
%
% projectV that projects the staggered velocity field into the sub-space of divergence-free velocity fields
%
% Input:
%   u: staggered velocity u*
%   v: staggered velocity v*
% phi: cell centered Lagrange multiplier with 1 ghost layer φ
%  dt: time step size Δt

% Ouput:
%    u: divergence free staggered velocity u^(n+1)
%    v: divergence free staggered velocity v^(n+1)

% Global Variables
global t h

M = size(phi,1)-1 ;
N = size(phi,2)-2 ;

for j = 2:N+1
   for i = 2:M

      u(i,j) = u(i,j) - dt * ((phi(i+1,j)-phi(i,j))/h) ;

   end
end

for j = 2:N
   for i = 2:M+1

      v(i,j) = v(i,j) - dt * ((phi(i,j+1)-phi(i,j))/h) ;

   end
end

u = bcGhost_u(u,t+dt) ;
v = bcGhost_v(v,t+dt) ;

end
```

7.4.31 psiWENO.m

```
function [psi] = psiWENO(a,b,c,d)
%
% psiWENO that calculates the value of the ψ(a, b, c, d) function of the WENO-5 method
%
% Input:
%   a: first argument in the ψ function of the WENO-5 method (this is not a from the problem statement!)
%   b: second argument in the ψ function of the WENO-5 method
%   c: third argument in the ψ function of the WENO-5 method
%   d: fourth argument in the ψ function of the WENO-5 method

% Ouput:
%   psi: value of ψ(a, b, c, d) of the WENO-5 method

epsilon = 1e-6 ;

IS0 = 13*(a-b)^2+3*(a-3*b)^2 ;
IS1 = 13*(b-c)^2+3*(b+c)^2 ;
IS2 = 13*(c-d)^2+3*(3*c-d)^2 ;
alpha0 = 1/(epsilon+IS0)^2 ;
alpha1 = 6/(epsilon+IS1)^2 ;
alpha2 = 3/(epsilon+IS2)^2 ;
omega0 = alpha0/(alpha0+alpha1+alpha2) ;
omega2 = alpha2/(alpha0+alpha1+alpha2) ;

psi = (1/3)*omega0*(a-2*b+c)+(1/6)*(omega2-(1/2))*(b-2*c+d) ;

end
```

7.4.32 sensor.m

```
function [F] = sensor(v)
%
% sensor that evaluates F(t), Eq. (18), using composite trapezoidal integration with τw evaluated using second-order finite differences
%
% Input:
%   v: 2D array of staggered velocity v

% Ouput:
%    F: value of F(t)

% Global Variables
global h Re yf

N = size(v,2) - 2 ;
iter = 1 ;

for j = 1:N
   if yf(j) >= 2.75 && yf(j) <= 3.25

      tw(iter) = -(1/Re)*(v(end,j)-v(end-1,j))/h ;
      iter = iter + 1 ;
   end
end

   F = (h/2)*(tw(1)+tw(end)+2*sum(tw(2:end-1))) ;

end
```

7.4.34 sensorY3.m

```
function [A] = sensorY3(v,Y)
%
% sensorY3 that evaluates with second order accuracy A(t) , Eq. (19), using composite midpoint integration.
%
% Input:
%   v: 2D array of staggered velocity v
%   Y: 2D array of cell centered mass fraction Y with 3 ghost layers

% Ouput:
%    A: value of A(t)

% Global Variables
global h xc

% Initalize
M = size(Y,1) - 6 ;
N = size(Y,2) - 6 ;

A = [] ;

for i = 1:length(xc)
   if xc(i) >= 1.5 && xc(i) <= 2.75

      A(end+1) =  v(i,N+1)*.5*(Y(i+2,N+4)+Y(i+2,N+3)) ;

   end
end

A = h*(sum(A)) ;

end
```

7.4.35 bcGhost_u.m

```
function [u] = bcGhost_u(u,t)
%
% bcGhost_u that applies only the ghost cell boundary conditions for the u velocity on a
% staggered mesh with at least second order accuracy in space
%
% Input:
%   u: 2D array of staggered mesh values of u
%   t: time at which to apply the boundary conditions

% Ouput:
%    u: 2D array of staggered mesh values of u with ghost cell boundary conditions applied

% Global Variables
global xf

U3 = 3/2*(1+sin(3*pi/4*t+pi/5)) ;
alpha3 = 2*pi/5 + pi/3*sin(2*pi/5*t) ;

z3 = @(x) (x-1.25)/.5 ;

f = @(z) 6*z*(1-z) ;

   for i = 1:length(xf)
      % Inlet 3
      if xf(i) >= 1.25 && xf(i) <= 1.75
         u(i,1) = 2*(U3*f(z3(xf(i))).*cos(alpha3))-u(i,2) ;
      else
         u(i,1) = -u(i,2) ; % Bottom
      end
   end

   for i = 1:length(xf)
      % Outlet
      if xf(i) >= 1.5 && xf(i) <= 2.75
         u(i,end) = u(i,end-1) ;
      else
         u(i,end) = -u(i,end-1) ; % Top
      end
   end


end
```

7.4.36 bcGhost_v.m

```
function [v] = bcGhost_v(v,t)
%
% bcGhost_v that applies only the ghost cell boundary conditions for the v velocity on a
% staggered mesh with at least second order accuracy in space
%
% Input:
%   v: 2D array of staggered mesh values of v
%   t: time at which to apply the boundary conditions

% Ouput:
%    v: 2D array of staggered mesh values of v with ghost cell boundary conditions applied

% Global Variables
global yf

U1 = (1+sin(2*pi/3*t+pi/4)) ;
U2 = 13/20*(1+sin(2*pi/5*t-pi/4)) ;

alpha1 = pi/3*sin(pi*t) ;
alpha2 = 5*pi/6 + pi/4*sin(3*pi/5*t) ;

z1 = @(y) (y-1)/.75 ;
z2 = @(y) (y-.5)/1.25 ;

f = @(z) 6*z*(1-z) ;

   for j = 1:length(yf)
      % Inlet 1
      if yf(j) >= 1 && yf(j) <= 1.75
         v(1,j) = 2*U1*f(z1(yf(j)))*sin(alpha1) - v(2,j) ;
      else
         v(1,j) = -v(2,j) ;
      end
   end

   for j = 1:length(yf)
      % Inlet 2
      if yf(j) >= .5 && yf(j) <= 1.75
         v(end,j) = 2*U2*f(z2(yf(j)))*sin(alpha2) - v(end-1,j) ;
      else
         v(end,j) = -v(end-1,j) ; % Right
      end
   end

end
```

7.4.37 bcGS.m

```
function [phi] = bcGS(phi)
%
% bcGS applies the given boundary conditions
%
% Input:
%   phi: 2D array of cell centered PDE dsolution variable (either phi or e) including ghost cells
% Ouput:
%   phi: 2D array of cell centered PDE solution variable (either phi or e) including ghost cells with
%        boundary conditions applied to the ghost cells.

global x y

   M = size(phi,1)-2 ;
   N = size(phi,2)-2 ;

      % Neumann Boundary Conditions
       phi(:,1) = phi(:,2) ; % x = 0
       phi(1,:) = phi(2,:) ; % y = 0
       phi(:,N+2) = phi(:,N+1) ; % y = 4
      % Dirichlet Boundary Conditions
      phi(M+2,:) = phi(M+1,:)  ; % (x = 4)

end
```