COMPUTATIONAL FLUID DYNAMICS: FINAL REPORT

Eric Weissman

Arizona State University

MAE 561

This paper outlines the numerical solution to the fluid flow in a mixing chamber as shown in figure 1. The chamber contains three velocity inlets and one outlet, with inlet 1 also introducing a chemical species. Within the domain there exists an immersed boundary condition modeling two chemical species absorbing disks which rotate sinusoidally. The objective of this paper is to determine the velocity fields as well as the chemical mass fraction within the computational domain. Then the amount of mixing that occurred within the chamber is quantified using the average kinetic energy and average scalar mixing within the chamber. The solutions are then verified using a GCI analysis for two different flow regimes.

Contents

Introduction and Governing Equations

The modeling of fluid flow has long been of interest to engineers and researchers as there exist countless practical applications including in medicine, construction, and transportation. Unfortunately, fluid flows have been shown to be modeled by a set of nonlinear PDEs containing both diffusive and convective terms. Thus far, no known general closed form solution to these PDEs exists except for in simplified and narrow cases. Therefore, with the advent of modern computational capabilities, numerical methods became the preferred method for modeling the fluid flow within a system. This paper will attempt to demonstrate a technique to model the flow within a computational domain using numerical methods.

The system concerning this paper is a mixing chamber as shown in figure 1. The chamber contains three velocity inlets and one outlet, with inlet 1 also introducing a chemical species. Within the domain there also exists an immersed boundary condition modeling two chemical species absorbing disks which rotate sinusoidally. This paper will determine the velocity fields as well as the chemical species' mass fraction within the computational domain and then the amount of mixing that occurred within the chamber will be quantified using the average kinetic energy and average scalar mixing within the chamber. The solutions will then be verified using a GCI analysis for two different flow regimes.

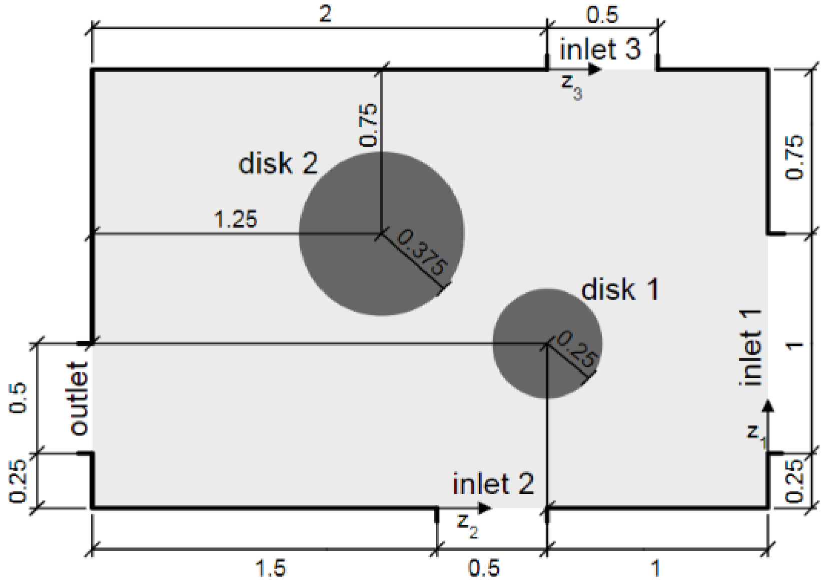

Figure 1: Mixing chamber computational domain.

The flow within the mixing chamber is modeled using equations 1 through 4, which respectively represent the nondimensional continuity equation, the 2D incompressible Navier-Stokes equation for the x and y directions, and the convective/diffusive transport equation for the mass fraction (Y) of the chemical species. The chemical species' diffusion is governed by the Schmidt number, representing a nondimensional ratio between the momentum diffusivity and the mass diffusivity. Likewise, the Navier-Stokes equations are characterized by the nondimensional Reynolds number, which represents the ratio between the viscosity of the fluid, and a characteristic length and velocity of the system.

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

Equation 1

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

Equation 2

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

Equation 3

$$where\ Re = \frac{u_{ref} L}{\nu}$$

$$\frac{\partial Y}{\partial t} + u\frac{\partial Y}{\partial x} + v\frac{\partial Y}{\partial y} = \frac{1}{ReSc}\left(\frac{\partial^2 Y}{\partial x^2} + \frac{\partial^2 Y}{\partial y^2}\right) + Q_Y$$

Equation 4

Notably, the Navier-Stokes equations contain parabolic and nonlinear hyperbolic terms, as well as a seemingly uncoupled pressure gradient term. Meanwhile the continuity equation appears to over-define the system. This paper will later discuss a method for enforcing the continuity equation while also including the effect of these pressure terms using an elliptical PDE. Likewise, the mass fraction equation also contains parabolic and nonlinear hyperbolic terms which are functions of the x and y velocities (u and v respectively), however with no pressure terms. Equations 2, 3, and 4 also contain source terms. These terms are used to introduce the immersed boundary discussed earlier.

The system is initially at rest with zero mass fraction within the system. Then the boundary conditions of the system are modeled as follows. For the walls, there is assumed to be zero flux of the mass fraction, in addition to the no slip condition as shown in equation 5. Likewise, equations 6 through 8 contain the boundary conditions for inlets 1 through 3 respectively. Z is a nondimensional coordinate from 0 to 1 along the length of the inlets.

$$u_w = 0;\ v_w = 0;\ \left.\frac{\partial Y}{\partial n}\right|_w = 0$$

Equation 5

Inlet 1: $Y = 1;\ u = \frac{3}{4}\cos(\pi)\, f(z_1);\ v = \frac{3}{4}\sin(\pi)\, f(z_1)$

Equation 6

Inlet 2: $Y = 0;\ u = 2\cos(\pi/3)\, f(z_2);\ v = 2\sin(\pi/3)\, f(z_2)$

Equation 7

$$\text{Inlet 3: } Y = 0;\ u = 3\cos(-\pi/6)\, f(z_3);\ v = 3\sin(-\pi/6)\, f(z_3)$$

Equation 8

Where:

$$f(z) = 6z(1 - z)$$

Equation 9

The outlet of the domain is then modeled by equation 10.

$$\left.\frac{\partial u}{\partial x}\right|_{out} = \left.\frac{\partial v}{\partial x}\right|_{out} = \left.\frac{\partial Y}{\partial x}\right|_{out} = 0$$

Equation 10

The performance of the mixing chamber is, as discussed above, quantified via two metrics. The first is by the time averaged kinetic energy, $\overline{K}$, within the system, as shown in equation 11. This is found by integrating the instantaneous kinetic energy within the system (which is found using equation 12) over a given period of time.

$$\overline{K} = \frac{1}{10}\int_0^{10} k(t)\, dt$$

Equation 11

$$k(t) = \int_0^{L_y}\int_0^{L_x} \frac{1}{2}(\hat{u}^2 + \hat{v}^2) dx dy$$

Equation 12

Where L_y is the height of the domain, L_x is the width, and where $\hat{v}$ and $\hat{u}$ are the velocities at cell center found from a 2nd order averaging. Likewise, the mixing performance is also quantified using the time averaged scalar mixing $\overline{R}$, calculated in a similar way to $\overline{K}$, as shown in in equations 13 and 14.

$$\overline{R} = \frac{1}{10}\int_0^{10} S(t)\, dt$$

Equation 13

$$S(t) = \frac{1}{L_x L_y}\int_0^{L_y}\int_0^{L_x} Y(1 - Y) dx dy$$

Equation 14

Numerical Methods

In order to solve the analytical equations described above numerically, the equations must be converted into finite difference equations. The finite difference equations are the Taylor-series approximation of the derivatives over some discretized space and time mesh, with the infinite series of terms truncate at some selected point. The point at which we truncate these terms determines the order of the method (a measure of how quickly the error of the method decreases as the mesh size decreases). A summary of the numerical methods used to model the mixing chamber can be found in table 1, which shows the name and the order of the methods. While the lowest order method used is the 1st order immersed boundary, the overall order of the solution method may not be first order if the dominant error is not from these immersed boundaries.

Table 1: Summary of numerical methods used.

	Convective Terms / Elliptical Terms			
	Name- time	**Order**	**Name- Space**	**Order**
u-equation	Adams-Bashforth	2nd	Central	2nd
v-equation	Adams-Bashforth	2nd	Central	2nd
Y-equation	TVD-RK3	3rd	WENO-5	5th

	Viscous / Diffusive Terms			
	Name- time	**Order**	**Name- Space**	**Order**
u-equation	Crank-Nicolson- ADI	2nd	Central	2nd
v-equation	Crank-Nicolson- ADI	2nd	Central	2nd
Y-equation	Crank-Nicolson- ADI	2nd	Central	2nd

	Boundary Conditions	**Immersed Boundary**	**Variable Location**
	Order- space	**Order- Space**	
u-equation	2nd	1st	Staggered Mesh ($u_{i+1/2,j}$)
v-equation	2nd	1st	Staggered Mesh ($v_{i,j+1/2}$)
Y-equation	2nd	1st	Cell Centered with 3 ghost layers ($Y_{i,j}$)

Along with the above methods, the 2D Navier-Stokes equations are solved using the fractional step method to enforce the continuity equation and to incorporate the pressure terms. This method generates a Poisson's equation, which is solved using the v-cycle multigrid with a 2nd-order Gauss Seidel method. The multigrid method is selected as the Gauss Seidel method converges quicker on coarser meshes.

u and v Equations

Expanding on the sections above, this section hopes to clearly express the finite difference equations, the solution methodology utilized, and the associated stability constraints of the u and v momentum equations (equations 2 and 3). To solve these equations, first the hyperbolic terms are calculated for a given time using a 2nd order central in space scheme, as shown in equations 15 through 18.

$$Right\ hand\ side\ v\ convective\ terms\ \equiv H_v = -\frac{\partial uv}{\partial x} - \frac{\partial v^2}{\partial y}$$

Equation 15

$$(H_v)^n_{i,j+\frac{1}{2}} = -\frac{\left(\left(v^n_{i,j+1}\right)^2 - \left(v^n_{i,j}\right)^2\right)}{h} - \frac{\left((uv)^n_{i+1/2\,,j+1/2} - \ (uv)^n_{i-1/2\,,j+1/2}\right)}{h}$$

Equation 16

$$Right\ hand\ side\ u\ convective\ terms\ \equiv H_u = -\frac{\partial u^2}{\partial x} - \frac{\partial uv}{\partial y}$$

Equation 17

$$(H_u)^n_{i+1/2,j} = -\frac{\left(\left(u^n_{i+1,j}\right)^2 - \left(u^n_{i,j}\right)^2\right)}{h} - \frac{\left((uv)^n_{i+1/2\,,j+1/2} - \ (uv)^n_{i+1/2\,,j-1/2}\right)}{h}$$

Equation 18

Where:

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

Equation 19

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

Equation 20

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

Equation 21

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

Equation 22

These hyperbolic terms are then incorporated into the source terms for the entirety of the time step using the Adams-Bashforth in time method. Then the diffusive terms are calculated using the implicit Crank-Nicholson with ADI in time with a central in space scheme. These finite differences can be seen in equations 23 and 24.

$$\frac{u^{n+1}_{i+\frac{1}{2},j} - u^n_{i+\frac{1}{2},j}}{\Delta t} = \frac{3}{2}{H_u}^n_{i+\frac{1}{2},j} - \frac{1}{2}{H_u}^{n-1}_{i+\frac{1}{2},j} + \frac{1}{2RE}\left(\left.\frac{\partial^2 u}{\partial x^2}\right|^n_{i+\frac{1}{2},j} + \left.\frac{\partial^2 u}{\partial x^2}\right|^{n+1}_{i+\frac{1}{2},j} + \left.\frac{\partial^2 u}{\partial y^2}\right|^n_{i+\frac{1}{2},j} + \left.\frac{\partial^2 u}{\partial y^2}\right|^{n+1}_{i+\frac{1}{2},j}\right) + Q_u$$

Equation 23

$$\frac{v_{i,j+\frac{1}{2}}^{n+1}-v_{i,j+\frac{1}{2}}^{n}}{\Delta t}=\frac{3}{2}H_{v}{}_{i,j+\frac{1}{2}}^{n}-\frac{1}{2}H_{v}{}_{i,j+\frac{1}{2}}^{n-1}+\frac{1}{2RE}\left(\left.\frac{\partial^2 v}{\partial x^2}\right|_{i,j+\frac{1}{2}}^{n}+\left.\frac{\partial^2 v}{\partial x^2}\right|_{i,j+\frac{1}{2}}^{n+1}+\left.\frac{\partial^2 v}{\partial y^2}\right|_{i,j+\frac{1}{2}}^{n}+\left.\frac{\partial^2 v}{\partial y^2}\right|_{i,j+\frac{1}{2}}^{n+1}\right)+Q_v$$

Equation 24

Where:

$$\left.\frac{\partial^2 v}{\partial x^2}\right|_{i,j+\frac{1}{2}}^{n}=\frac{v_{i,j+\frac{3}{2}}-2v_{i,j+\frac{1}{2}}+v_{i,j-\frac{1}{2}}}{h^2}$$

Equation 25

$$\left.\frac{\partial^2 v}{\partial y^2}\right|_{i,j+\frac{1}{2}}^{n}=\frac{v_{i+1,j+\frac{1}{2}}-2v_{i,j+\frac{1}{2}}+v_{i-1,j+\frac{1}{2}}}{h^2}$$

Equation 26

$$\left.\frac{\partial^2 u}{\partial x^2}\right|_{i+\frac{1}{2},j}^{n}=\frac{u_{i+\frac{3}{2},j}-2u_{i+\frac{1}{2},j}+u_{i-\frac{1}{2},j}}{h^2}$$

Equation 27

$$\left.\frac{\partial^2 u}{\partial y^2}\right|_{i+\frac{1}{2},j}^{n}=\frac{u_{i+\frac{1}{2},j+1}-2u_{i+\frac{1}{2},j}+u_{i+\frac{1}{2},j-1}}{h^2}$$

Equation 28

Solving for the diffusive and convective terms has thus far failed to enforce the continuity equation, nor the pressure terms (thus we call the resulting velocities at this point u* and v*). Therefore, the continuity equation is enforced using equation 29.

$$u_{\frac{3}{2},j}^{*}=u_{\frac{3}{2},j}^{*}-\frac{h}{L_{out}}\left(\sum_{j=2}^{N+1}\left(u_{\frac{3}{2},j}^{*}-u_{M+\frac{3}{2},j}^{*}\right)+\sum_{i=1}^{M+1}\left(v_{i,\frac{3}{2}}^{*}-v_{i,N+\frac{3}{2}}^{*}\right)\right)$$

Equation 29

Likewise, the pressure correction is then determined using the fractional step method, which generates the aforementioned pressure Poisson's equations as shown in equation 30 (finite difference form in equation 31).

$$\left.\frac{\partial^2 \varphi}{\partial x^2}\right|^{n+1}+\left.\frac{\partial^2 \varphi}{\partial y^2}\right|^{n+1}=\frac{1}{\Delta t}\left(\frac{\partial u^*}{\partial x}+\frac{\partial v^*}{\partial y}\right)$$

Equation 30

$$\frac{\varphi_{i+1,j}^{n+1} - 2\varphi_{i,j}^{n+1} + \varphi_{i-1,j}^{n+1}}{h^2} + \frac{\varphi_{i,j+1}^{n+1} - 2\varphi_{i,j}^{n+1} + \varphi_{i,j-1}^{n+1}}{h^2} = \frac{1}{\Delta t}\left(\frac{u_{i+\frac{1}{2},j}^{*} - u_{i-\frac{1}{2},j}^{*}}{h} + \frac{v_{i,j+\frac{1}{2}}^{*} - v_{i,j-\frac{1}{2}}^{*}}{h}\right)$$

Equation 31

The solution to the pressure Poisson's is then used to project the u and v functions back into the solenoidal function space, which enforces the pressure terms (see equations 32 to 34).

$$\vec{v}^{n+1} = \vec{v}^{*} - \Delta t \nabla \varphi^{n+1}$$

Equation 32

$$u_{i+\frac{1}{2},j}^{n+1} = u_{i+\frac{1}{2},j}^{*} - \Delta t \frac{\varphi_{i+1,j}^{n+1} - \varphi_{i,j}^{n+1}}{h}$$

Equation 33

$$v_{i,j+\frac{1}{2}}^{n+1} = v_{i,j+\frac{1}{2}}^{*} - \Delta t \frac{\varphi_{i,j+1}^{n+1} - \varphi_{i,j}^{n+1}}{h}$$

Equation 34

Using this solution methodology results in a time step constraint as well as a cell Reyolds number constraint. These can be seen in equations 35 to 37.

$$\frac{max_{i+\frac{1}{2},j}\left(\left|u_{i+\frac{1}{2},j}^{n}\right|\right)\Delta t}{h} + \frac{max_{i,j+\frac{1}{2}}\left(\left|v_{i,j+\frac{1}{2}}^{n}\right|\right)\Delta t}{h} \leq 1$$

Equation 35

$$Re_{u,c} \leq Re\ h\left[2\ max_{i+\frac{1}{2},j}\left(\left|u_{i+\frac{1}{2},j}^{n}\right|\right)\Delta t + max_{i,j+\frac{1}{2}}\left(\left|v_{i,j+\frac{1}{2}}^{n}\right|\right)\Delta t\right]$$

Equation 36

$$Re_{v,c} \leq Re\ h\left[2max_{i+\frac{1}{2},j}\left(\left|u_{i+\frac{1}{2},j}^{n}\right|\right)\Delta t + 2\ max_{i,j+\frac{1}{2}}\left(\left|v_{i,j+\frac{1}{2}}^{n}\right|\right)\Delta t\right]$$

Equation 37

Y Equations

Similar to the above section, this section hopes to explain the methods and finite difference equations used to find the mass fraction within the system. Like the u and v momentum equations, the convective/diffusive transport equation for Y first calculates the nonlinear hyperbolic term shown in equation 38.

$$H_{Y\,i,j}^{\;n} = \frac{\left(Y_{s\,i,j}^{\;n} - Y_{i,j}^{n}\right)}{\Delta t}$$

Equation 38

Where Y_s is calculated using TVD-RK-3 WENO-5 which starts with a 3-step Rutta Kunga process shown in equations 39 through 41.

$$\varphi_{i,j}^{(1)} = Y_{i,j}^{n} - a_{0\,i,j}\Delta t \left.\frac{\partial Y^n}{\partial x}\right|_{i,j}^{\pm} - b_{0\,i,j}\Delta t \left.\frac{\partial Y^n}{\partial y}\right|_{i,j}^{\pm}$$

Equation 39

$$\varphi_{i,j}^{(2)} = \varphi_{i,j}^{(1)} + \frac{3}{4}\Delta t\left(a_{0\,i,j}\left.\frac{\partial Y^n}{\partial x}\right|_{i,j}^{\pm} + b_{0\,i,j}\left.\frac{\partial Y^n}{\partial y}\right|_{i,j}^{\pm}\right) - \frac{1}{4}\Delta t\left(a_{1\,i,j}\left.\frac{\partial Y^n}{\partial x}\right|_{i,j}^{\pm} + b_{1\,i,j}\left.\frac{\partial Y^n}{\partial y}\right|_{i,j}^{\pm}\right)$$

Equation 40

$$Y_s = \varphi_{i,j}^{(2)} + \frac{1}{12}\Delta t\left(a_{0_{i,j}}\left.\frac{\partial Y^n}{\partial x}\right|_{i,j}^{\pm} + b_{0\,i,j}\left.\frac{\partial Y^n}{\partial y}\right|_{i,j}^{\pm}\right) + \frac{1}{12}\Delta t\left(a_{1_{i,j}}\left.\frac{\partial Y^n}{\partial x}\right|_{i,j}^{\pm} + b_{1_{i,j}}\left.\frac{\partial Y^n}{\partial y}\right|_{i,j}^{\pm}\right) - \frac{2}{3}\Delta t\left(a_{2_{i,j}}\left.\frac{\partial Y^n}{\partial x}\right|_{i,j}^{\pm} + b_{2_{i,j}}\left.\frac{\partial Y^n}{\partial y}\right|_{i,j}^{\pm}\right)$$

Equation 41

The a and b values are calculated using a midpoint averaging for various points in time as shown in equations 42 and 43.

$$a_{0\,i,j} = \frac{u_{i+\frac{1}{2},j}^{n-1} + u_{i+\frac{3}{2},j}^{n-1}}{2};\; a_{1\,i,j} = \frac{u_{i+\frac{1}{2},j}^{n} + u_{i+\frac{3}{2},j}^{n}}{2};\; a_{2\,i,j} = \frac{a_{0\,i,j} + a_{1\,i,j}}{2}$$

Equation 42

$$b_{0\,i,j} = \frac{v_{i,j+\frac{1}{2}}^{n-1} + v_{i,j+\frac{3}{2}}^{n-1}}{2}; b_{1\,i,j} = \frac{v_{i,j+\frac{1}{2}}^{n} + v_{i,j+\frac{3}{2}}^{n}}{2}; b_{2\,i,j} = \frac{b_{0\,i,j} + b_{1\,i,j}}{2}$$

Equation 43

When $a_{i,j}$ is greater than 0, equation 44 is utilized to calculate the $\left.\frac{\partial Y^n}{\partial x}\right|_{i,j}^{\pm}$ term.

$$\left.\frac{\partial Y^n}{\partial x}\right|_{i,j}^{-} = \frac{1}{12h}\left(-\Delta^+ Y_{i-2,j}^{n} + 7\Delta^+ Y_{i-1,j}^{n} + 7\Delta^+ Y_{i,j}^{n} - \Delta^+ Y_{i+1,j}^{n}\right) - \psi_{weno}\left(\frac{\Delta^-\Delta^+ Y_{i-2,j}^{n}}{h}, \frac{\Delta^-\Delta^+ Y_{i-1,j}^{n}}{h}, \frac{\Delta^-\Delta^+ Y_{i,j}^{n}}{h}, \frac{\Delta^-\Delta^+ Y_{i+1,j}^{n}}{h}\right)$$

Equation 44

And when $a_{i,j}$ is less than 0, equation 45 is utilized to calculate the $\left.\frac{\partial Y^n}{\partial x}\right|_{i,j}^{\pm}$ term.

$$\left.\frac{\partial Y^n}{\partial x}\right|_{i,j}^{+} = \frac{1}{12h}\left(-\Delta^+ Y_{i-2,j}^n + 7\Delta^+ Y_{i-1,j}^n + 7\Delta^+ Y_{i,j}^n - \Delta^+ Y_{i+1,j}^n\right) + \psi_{weno}\left(\frac{\Delta^-\Delta^+ Y_{i+2,j}^n}{h}, \frac{\Delta^-\Delta^+ Y_{i+1,j}^n}{h}, \frac{\Delta^-\Delta^+ Y_{i,j}^n}{h}, \frac{\Delta^-\Delta^+ Y_{i-1,j}^n}{h}\right)$$

Equation 45

Where:

$$\Delta^+ Y_{i,j}^n = Y_{i+1,j}^n - Y_{i,j}^n$$

Equation 46

$$\Delta^-\Delta^+ Y_{i,j}^n = Y_{i+1,j}^n - 2Y_{i,j}^n + Y_{i-1,j}^n$$

Equation 47

When $b_{i,j}$ is greater than 0, equation 48 is utilized to calculate the $\left.\frac{\partial Y^n}{\partial y}\right|_{i,j}^{\pm}$ term.

$$\left.\frac{\partial Y^n}{\partial y}\right|_{i,j}^{-} = \frac{1}{12h}\left(-\Delta^+ Y_{i,j-2}^n + 7\Delta^+ Y_{i,j-1}^n + 7\Delta^+ Y_{i,j}^n - \Delta^+ Y_{i,j+1}^n\right) - \psi_{weno}\left(\frac{\Delta^-\Delta^+ Y_{i,j-2}^n}{h}, \frac{\Delta^-\Delta^+ Y_{i,j-1}^n}{h}, \frac{\Delta^-\Delta^+ Y_{i,j}^n}{h}, \frac{\Delta^-\Delta^+ Y_{i,j+1}^n}{h}\right)$$

Equation 48

And when $b_{i,j}$ is less than 0, equation 49 is utilized to calculate the $\left.\frac{\partial Y^n}{\partial y}\right|_{i,j}^{\pm}$ term.

$$\left.\frac{\partial Y^n}{\partial y}\right|_{i,j}^{+} = \frac{1}{12h}\left(-\Delta^+ Y_{i,j-2}^n + 7\Delta^+ Y_{i,j-1}^n + 7\Delta^+ Y_{i,j}^n - \Delta^+ Y_{i,j+1}^n\right) + \psi_{weno}\left(\frac{\Delta^-\Delta^+ Y_{i,j+2}^n}{h}, \frac{\Delta^-\Delta^+ Y_{i,j+1}^n}{h}, \frac{\Delta^-\Delta^+ Y_{i,j}^n}{h}, \frac{\Delta^-\Delta^+ Y_{i,j-1}^n}{h}\right)$$

Equation 49

Where:

$$\Delta^+ Y_{i,j}^n = Y_{i,j+1}^n - Y_{i,j}^n$$

Equation 50

$$\Delta^-\Delta^+ Y_{i,j}^n = Y_{i,j+1}^n - 2Y_{i,j}^n + Y_{i,j-1}^n$$

Equation 51

Additionally, the ψ_{weno} term is found using equation 52.

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

Equation 52

Where:

$$\omega_0 = \frac{\alpha_0}{\alpha_0+\alpha_1+\alpha_2};\ \omega_2 = \frac{2}{\alpha_0+\alpha_1+\alpha_2}$$

Equation 53

Where:

$$\alpha_0 = \frac{1}{(\epsilon + IS_0)^2};\ \alpha_1 = \frac{6}{(\epsilon + IS_1)^2};\ \alpha_2 = \frac{3}{(\epsilon + IS_2)^2}$$

Where:

$$\epsilon = 10^{-6}$$

$$IS_0 = 13(a-b)^2 + 3(a-3b)^2;\ IS_1 = 13(b-c)^2 + 3(b+c)^2;\ IS_2 = 13(c-d)^2 + 3(3c-d)^2$$

Equation 54

Now with the hyperbolic terms calculated, similar to the u and v momentum equations, the hyperbolic terms are then incorporated into the source terms for the entirety of the time step. Then the diffusive terms are calculated using the implicit Crank-Nicholson with ADI in time with a central in space scheme. The finite difference equation can be seen in equations 55.

$$\frac{Y_{i,j}^{n+1} - Y_{i,j}^{n}}{\Delta t} = {H_Y}_{i,j}^{n} + \frac{1}{2 * ScRe}\left(\left.\frac{\partial^2 Y}{\partial x^2}\right|_{i,j+1}^{n} + \left.\frac{\partial^2 Y}{\partial x^2}\right|_{i,j+1}^{n+1} + \left.\frac{\partial^2 Y}{\partial y^2}\right|_{i,j+1}^{n} + \left.\frac{\partial^2 Y}{\partial y^2}\right|_{i,j+1}^{n+1}\right) + Q_Y$$

Equation 55

Finally, this method for calculating the mass fraction produces a hyperbolic time step restriction as shown in equation 56.

$$\frac{max_{i,j}\left(\left|u_{i,j}^{n}\right|\right)\Delta t}{h} + \frac{max_{i,j}\left(\left|v_{i,j}^{n}\right|\right)\Delta t}{h} \leq 1$$

Equation 56

Performance Metrics

The aforementioned performance metrics of the mixing chamber are also calculated using numerical methods. The time-averaged kinetic energy is calculated using a 2nd-order composite trapezoidal integration as shown in equation 57. Likewise, the time-averaged scalar mixing parameter is also determined using a 2nd-order composite trapezoidal integration as shown in equation 58.

$$\bar{K} = \frac{1}{10}\sum_{i=2}^{n}\frac{1}{2}(t_i - t_{i-1})\big(k(t_{i-1}) + k(t_i)\big)\ where\ n = the\ number\ of\ time\ increments$$

Equation 57

$$\bar{R} = \frac{1}{10}\sum_{i=2}^{n}\frac{1}{2}(t_i - t_{i-1})\big(S(t_{i-1}) + S(t_i)\big)\ where\ n = the\ number\ of\ time\ increments$$

Equation 58

The instantaneous measurements of the kinetic energy and the mixing parameter (k(t) and S(t)) are both calculated using a composite 2D midpoint integration which is a 2nd-order method and can be seen in equations 59 and 60.

$$k(t) = \frac{1}{2}h^2\left(\sum_{j=2}^{N+1}\sum_{i=1}^{M}\left(\frac{u_{i+\frac{3}{2},j} + u_{i+\frac{1}{2},j}}{2}\right)^2 + \sum_{j=1}^{N}\sum_{i=2}^{M+1}\left(\frac{v_{i,j+\frac{3}{2}} + v_{i,j+\frac{1}{2}}}{2}\right)^2\right)$$

Equation 59

$$S(t) = \frac{h^2}{L_x L_y}\sum_{j=4}^{N+3}\sum_{i=4}^{M+3} Y_{i,j}\big(1 - Y_{i,j}\big)$$

Equation 60

Results

Results for Re = 100 and Sc = 2

To validate the model a simulation was performed with the Re equal 100 and the Sc equal to 2. The solution parameters were set to provide accurate results in the GCI analysis (shown below), while minimizing computational time. Therefore, a CFL equal to 0.5 was selected to ensure numerical stability and to limit the truncation errors caused by the time discretization (as the GCI is analyzing the spatial discretization). Then the finest mesh used had an N = 512 and an M = 768. The Poisson solution controller was then selected to provide an error below the solution order's method. Therefore, the max number of iterations is set to 100 and epsilon is set to h/100. Next, the number of Gauss-Seidel iterations on coarsest mesh was 3. The result of this can be seen in figures 2 to 6. Additionally, in the supplementary materials (a.k.a. canvas) there are movies posted showing the flow evolution over the time domain.

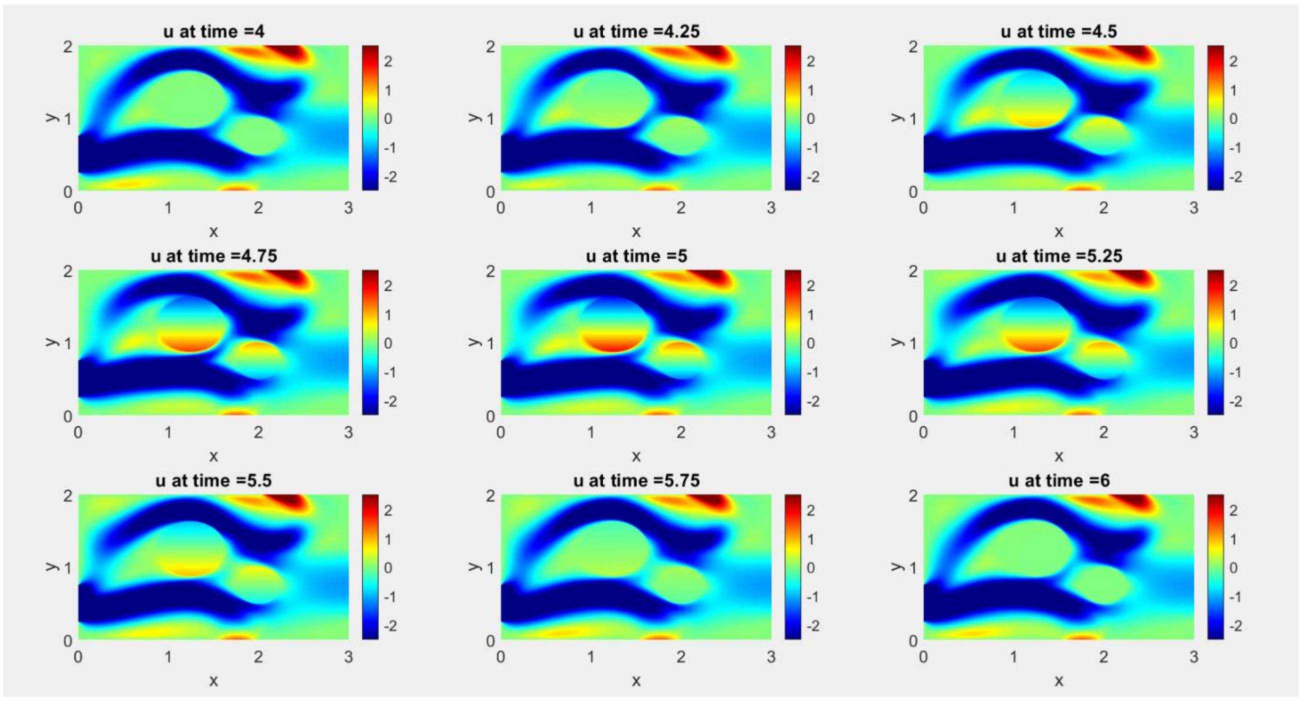

Figure 2: u velocity pcolors for t = 4 to t=6

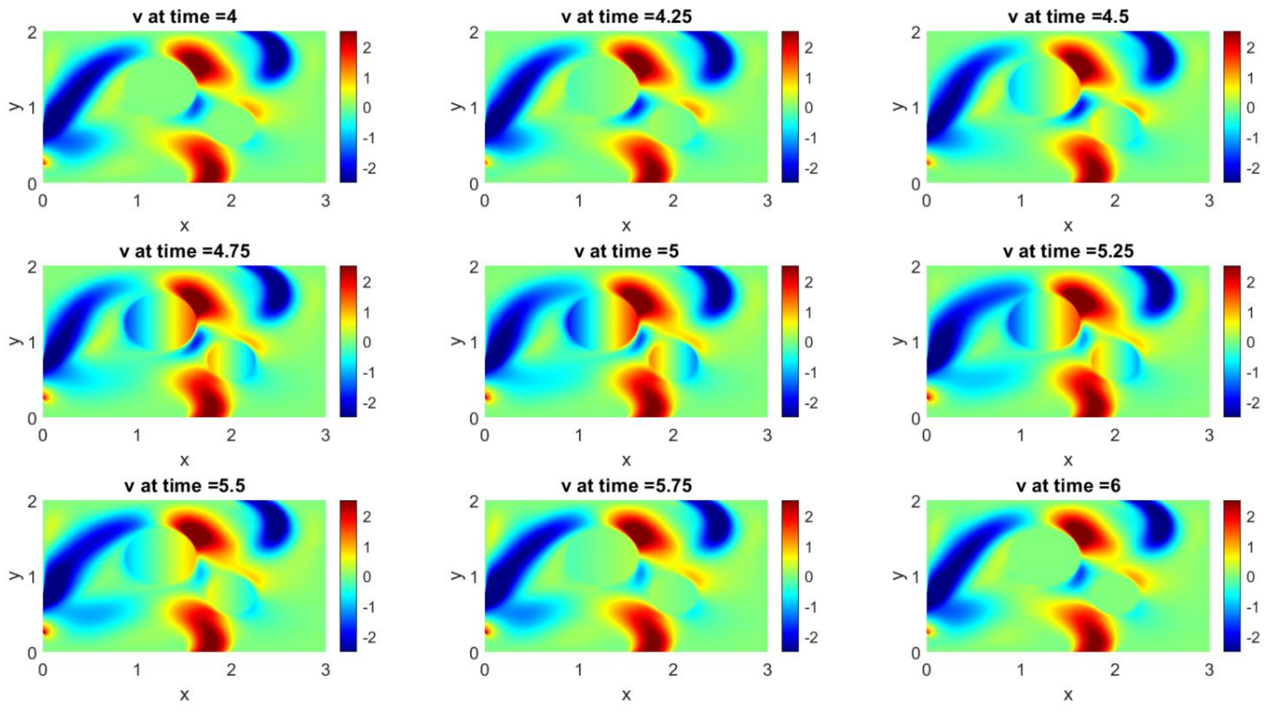

Figure 3:v velocity pcolors for t = 4 to t=6

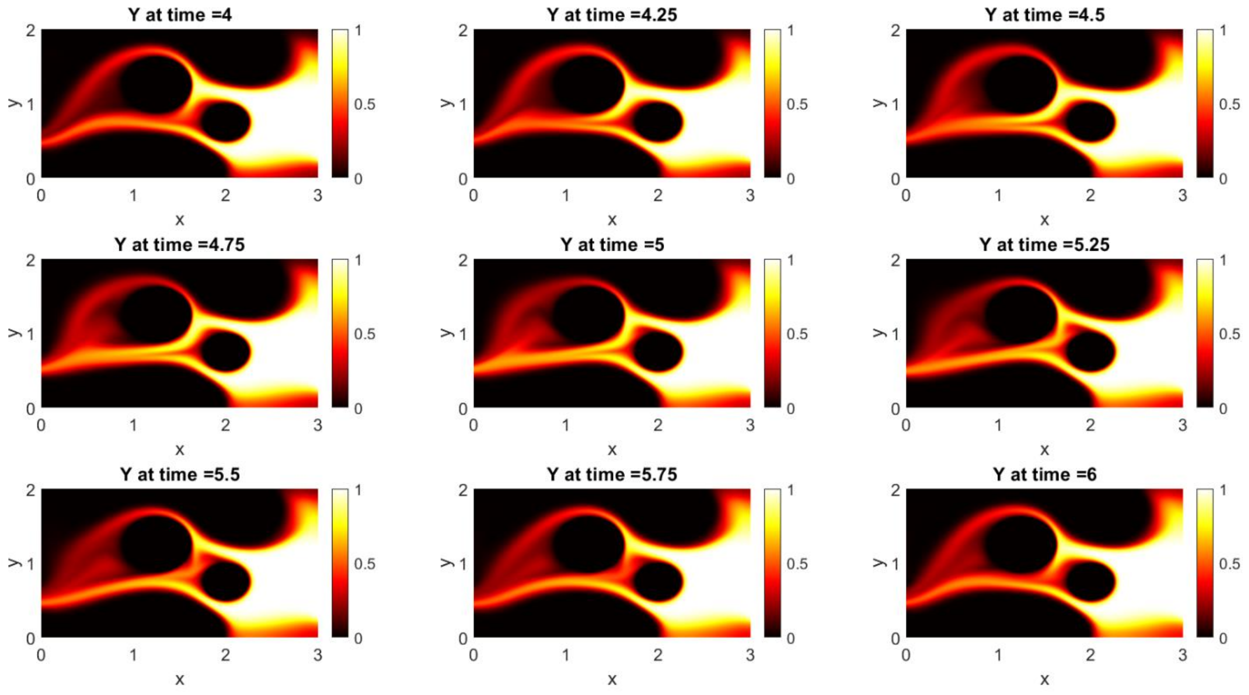

Figure 4: Y pcolors for t = 4 to t=6

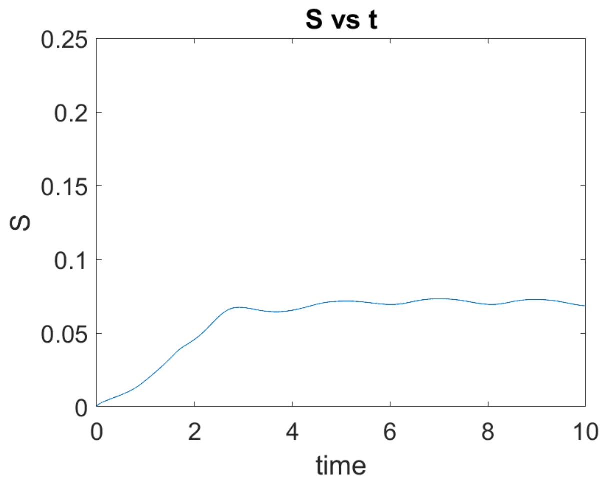

Figure 5: S(t) for Re = 100 and Sc = 2

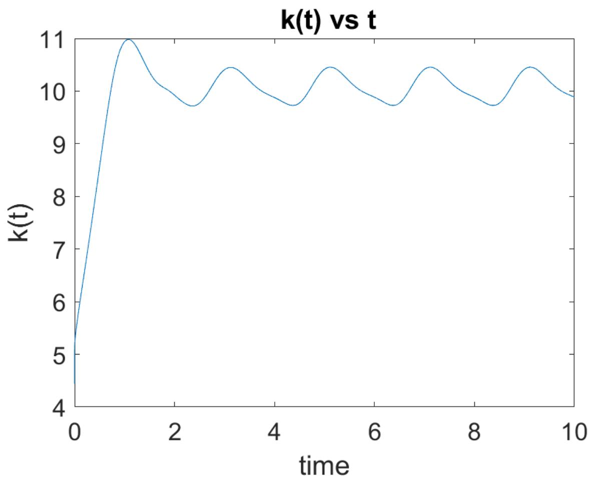

Figure 6: k(t) for Re = 100 and Sc = 2

Prior to looking at the GCI analysis, the above figures offer a qualitative understanding of the flow patterns within the mixing chamber. Notably, the u and v pcolor plots illustrate that as expected, the fluid enters the system at the inlets at the angles prescribed by the boundary conditions. Then the flow snakes around the disks and towards the outlet. When the disks rotate, the fluid flow direction between the disks can be seen to change its direction to match the disks. The results of the disks can also be seen in figure 6, showing that the average kinetic energy increases when the disks rotate.

In terms of mixing, it is clear from figures 4 and 5 that as the chemical species enters through the first inlet, it begins to diffuse through the fluid. Then as additional fluid is mixed in from the other inlet and as the species is removed by the disks, the concentration of the chemical species in fluid decreases. Interestingly, it is clear from figure 5 that as the disks rotate the mixing parameter increases. This is likely due to the rotation causing less of the mass fraction from being absorbed by the disk.

Solution Verification for RE = 100 and Sc = 2

GCI Analysis For $\overline{K}$

Presented in the table below is the results for the GCI analysis for $\overline{K}$. The analysis attempts to estimate the actual order of the method employed, and to use this information to extrapolate out the true value of the parameter of interest (in this case $\overline{K}$) as h goes to zero. The data presented below has the results for two different GCI analyses, which were performed as additional data is provided for finer meshes. Notably, in both analyses, the asymptotic range of convergence is very nearly equal to one, which suggests that the solutions are approaching a converged answer.

Table 2: GCI analysis for $\overline{K}$

M	N	$\overline{K}$	P	GCI12	GCI23	f_{h0}	Asymptotic Convergence
96	64	9.86798754568237	-	-	-	-	-
192	128	9.91136520322797	-	-	-	-	-
384	256	9.93797586818314	0.70494758	0.005312125	0.00868246	9.9802092	0.997322325
768	512	9.940716566947247	3.279388889	3.95697335e-05	3.8430612144e-04	9.9410312	0.999724295653

This analysis then permits us to make claims about the accuracy of our measurement. We can confidently assert (if our code is bug free and our inputs were infinitely precise) that:

$$\overline{K} = 9.941031248151880 \pm 3.956973354421057\text{e} - 03\%.$$

GCI Analysis For $\overline{R}$

The same analysis was then performed for the $\overline{R}$ solutions, which again verified that in both analyses, the asymptotic range of convergence is very nearly equal to one, suggesting that the solutions are approaching a converged answer.

Table 3: GCI analysis for $\overline{R}$

M	N	$\overline{R}$	P	GCI12	GCI23	f_{h0}	Asymptotic Convergence
96	64	0.0555571541442785	-	-	-	-	-
192	128	0.0582929740030818	-	-	-	-	-
384	256	0.0590709523646061	1.8141713	0.00654174	0.02331155	0.0593800	0.98682976
768	512	0.059081155470659	6.25264971	2.86874352e-06	2.187770931e-04	0.0590812	0.9998273

Again, using this information we can claim (if our code is bug free and our inputs were infinitely precise) that:

$$\overline{R} = 0.059081291061605 \pm 2.868743527827969\text{e} - 04\%$$

A note on the observed order

The observed order (p) is the rate at which the error is actually decreasing as h is decreased. Recalling table 1, we will note that the theoretical observed order for the method should be between 1 and 2 for u and v and between 1 and 5 for Y depending on source of the dominant error. However, the above analyses show that on the courser meshes we can get observed orders less than anticipated, while on the finer meshes, we can exceed the formal order. Discrepancies between the observed and formal orders are common and can be caused by numerous factors including large time discretization errors or other error sources other than truncation. However, in this case we are employing a mixed-order method (namely a method which applies methods of different orders). Thus, it is likely that the discrepancy of the formal and observed orders are the result of error cancellation and enhancement. What likely happened is that error was surreptitiously enhanced in one of the coarser meshes, meaning when the GCI analysis was performed, the observed order was less than expected. Then when the finer meshes were analyzed, because of the earlier high error, the method appeared to converge rapidly, indicating a higher order method than was truly displayed. This is evident from the observed order first being less than expected and then swinging to above the expectation.

Results for RE = 200 and Sc = 3

In the hopes of analyzing more turbulent and dispersive flows, shown here are the results of the numerical model of the mixing chamber at an Re = 200 and an Sc = 3 (all other parameters are the same to the previous analysis).

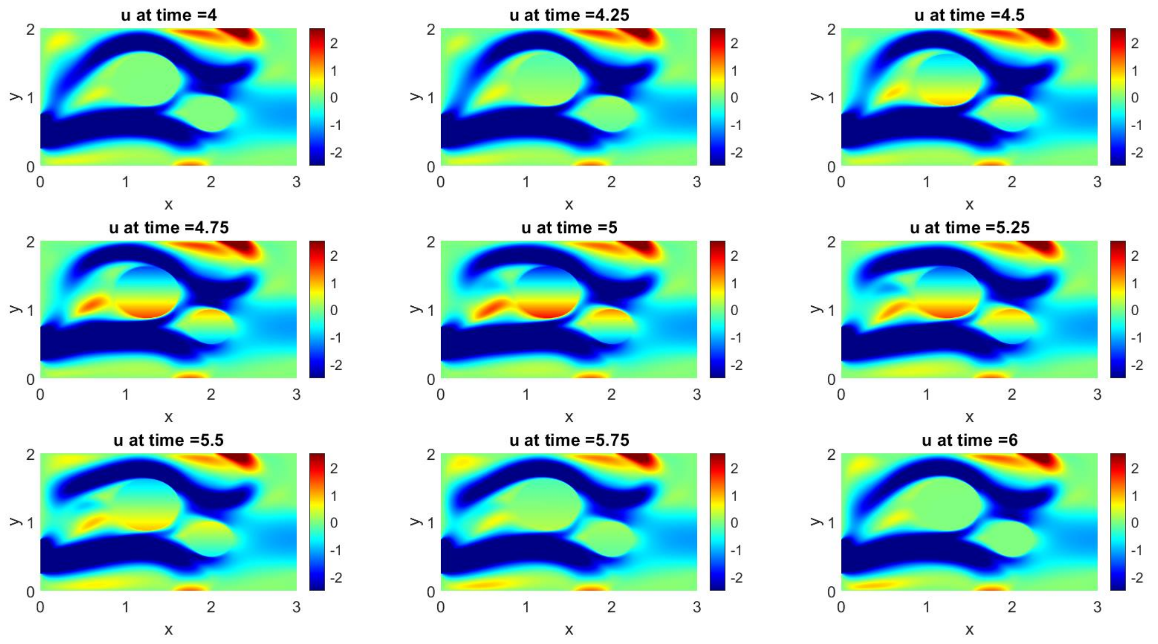

Figure 7: u velocity pcolors for t = 4 to t=6

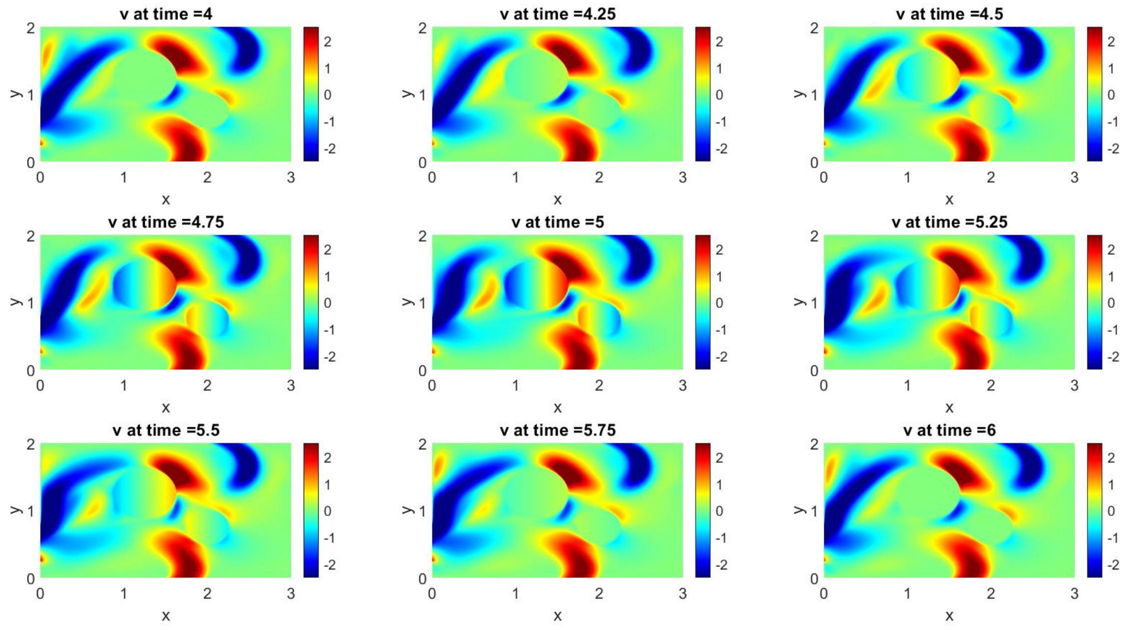

Figure 8:v velocity pcolors for t = 4 to t=6

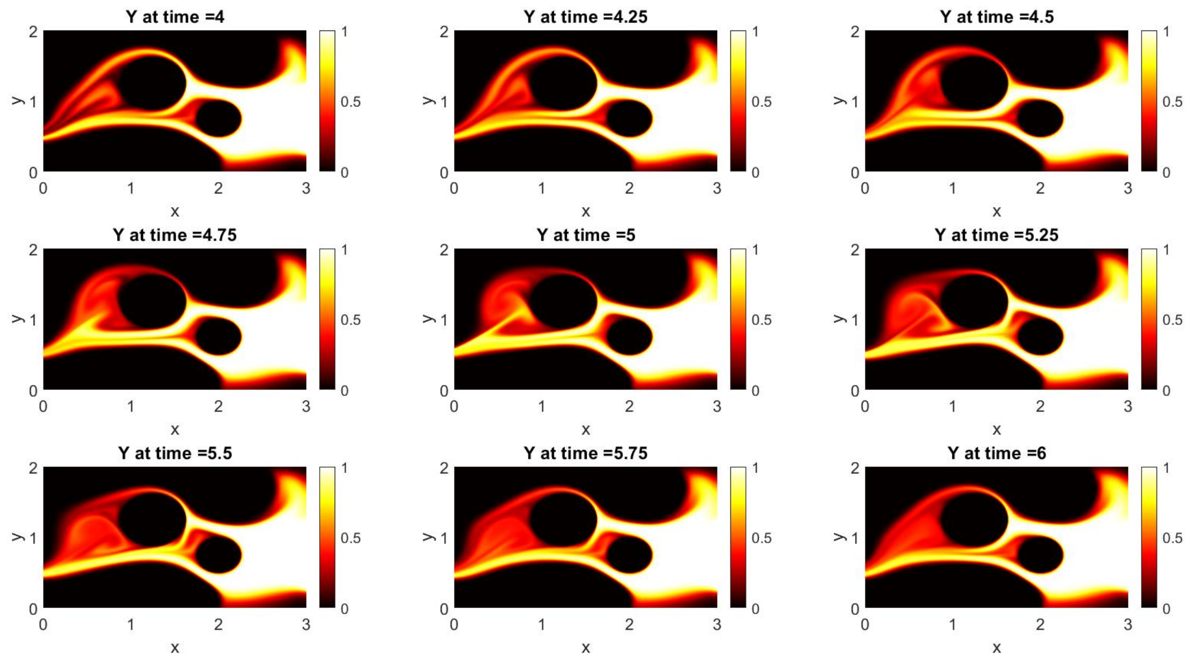

Figure 9 : Y pcolors for t = 4 to t=6

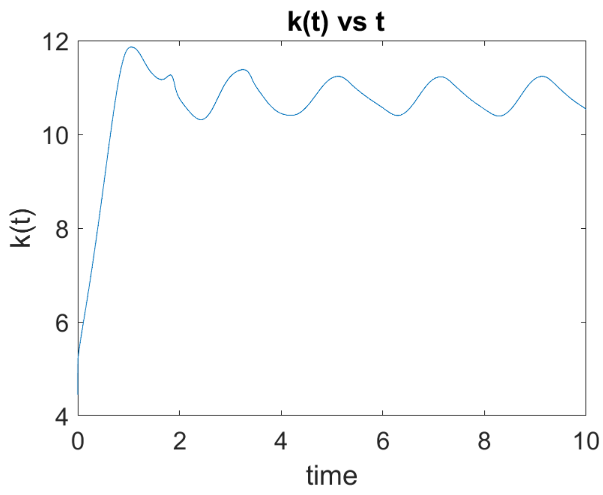

Figure 10: k(t) for Re = 200 and Sc = 3

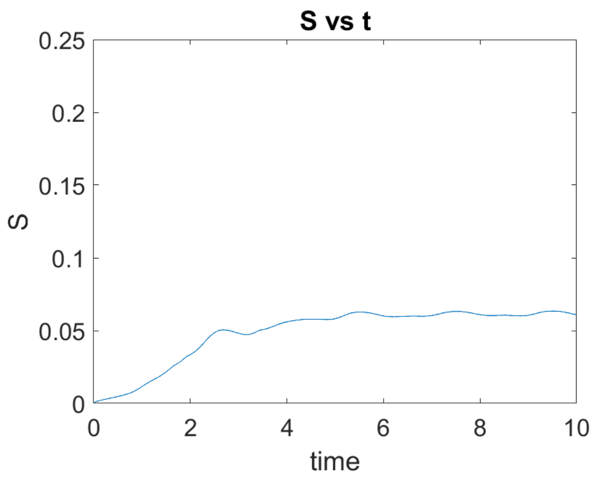

Figure 11: S(t) for Re = 200 and Sc = 3

The results indicated from figures 7 and 8 appear to mimic those of figures 2 and 3, with some minor differences that are difficult to distinguish qualitatively. That said, comparing figures 3 and 9, it is clear that the increase in the Re and the Sc resulted in a slightly more turbulent flow. This is especially evident in the wake (to the left of) the larger of the disks, where it appears that the beginning of vortex shedding can be observed in figure 9. The differences between the 2 systems are also evident in figures 10 and 11, which show that for the increased Re and Sc the instantaneous kinetic energy of the system has higher peak values. Likewise, the S(t) curve appears to have a less defined relationship to the disk motion, evident in the decrease in a sinusoidal pattern.

Solution Verification for RE = 200 and Sc = 3

Again, similar to the instance where Re = 100 and Sc = 2, a GCI analysis can be employed to verify our answers and to determine the error bars on our solution. The results of this analysis can be seen below.

Table 4: $\overline{K}$ and $\overline{R}$ for various mesh sizes

M	N	$\overline{K}$	$\overline{R}$
192	128	10.6383284508859	0.0474547197933723
384	256	10.6625187387031	0.0487132899829918
768	512	10.6598689868027	0.048916308577989

GCI Analysis For $\overline{K}$

Assuming infinite precision on the input parameters:

GCI Analysis For $\overline{K}$	GCI Analysis For $\overline{R}$
P = 3.190498744889875	P = 2.632101898141499
f_{h0} = 10.659543034599759	f_{h0} = 0.048955355982327
GCI_{12} = 3.822188191421249e-05	GCI_{12} = 9.978115037979859e-04
GCI_{23} = 3.488509816802560e-04	GCI_{23} = 0.006211497915149
Asymptotic Convergence = 1.000248572651664	Asymptotic Convergence = 0.995849674660673

Therefore,

$$\overline{K} = 10.659543034599759 \pm 3.822188191421249\text{e} - 03\%$$

$$\overline{R} = 0.048955355982327 \pm 9.978115037979859\text{e} - 02\ \%$$

For both of the above GCI analyses, the asymptotic range of convergence being near to 1 indicated that that the solutions are approaching a converged answer. Additionally, for $\overline{R}$ the observed order is within the value ranges we would expect given the formal order and indicates that the TVD-RK3 method could be the source of the leading error term. For the $\overline{K}$ analysis, the order is above the value we would expect from the formal orders. As described before, this is probably due to error cancellation and enhancement.

Conclusion and Discussion

This paper was able to demonstrate that the nonlinear convective/ diffusive equations modeling a mixing chamber with a chemical species could be modeled and simulated using numerical approaches for a range of flow conditions. This use case represents a hypothetical application that an engineer might apply CFD to gain understanding or to design a system. The numerical methods used included the Crank-Nicholson ADI with a central in space scheme for solving the parabolic equations, the fractional step method for solving the pressure Poisson, and the Adams-Bashforth with central in space / TVD-RK3 WENO-5 methods for solving the hyperbolic terms. These methods represent a robust basis of tools for solving CFD problems or to understand more advanced modern methods used to solve modern CFD problems. The results from this investigation indicate that increasing the Re and the Sc can lead to a higher average kinetic energy, as well as a lower average mixing parameter. Qualitatively, this analysis indicated how increasing the Re and the Sc can encourage a more turbulent flow and the onset of vortex shedding. Finally, a GCI analysis was successfully performed to verify the solution and to discuss a method for engineers to estimate the error on their solutions.

Appendix

Code:

MAE 561-- Final Project

```
clear
clear all


clf reset
format long


global h CFL Re Sc xf xc yf yc yc3 xc3 Lx Ly t;


%Declare some constants
M=96;
N=64;
CFL = 0.5;
Re = 400;
Sc = 10;
Lx=3;
Ly=2;


times_of_interest = [4:.25:6, 10];


x_range = [0,Lx];
y_range = [0,Ly];


%h is equal in both x and y
h = (x_range(2)-x_range(1))/(M);


%poissons solver controls:
nIterMax = 100;
epsilon = (h)/100; %Q is first order in time, so epsilon needs only be less than h


%lets define xf, yf, yc, xc, sc3 and yc3;
xf = linspace(x_range(1),x_range(2),M+1)';
yf = linspace(y_range(1),y_range(2),N+1)';
xc = linspace(x_range(1)-h/2,x_range(1)+(x_range(2)-x_range(1))+h/2,M+2)';
yc = linspace(y_range(1)-h/2,y_range(1)+(y_range(2)-y_range(1))+h/2,N+2)';
yc3 = linspace(y_range(1)-5*h/2,y_range(1)+(y_range(2)-y_range(1))+5*h/2,N+6)';
xc3 = linspace(x_range(1)-5*h/2,x_range(1)+(x_range(2)-x_range(1))+5*h/2,M+6)';


%lets define some variables we will be storing
%we do not know the length of the vectore before hand. Thus we will
%preallocate memory in blocks and add blocks as needed
BlockSize = 1000;% size increments added to vector length
currentSize = BlockSize; %current length of vectors
k_values = zeros(BlockSize,1);
S_values = zeros(BlockSize,1);
Time_values = zeros(BlockSize,1);


%declare t
t = 0;
```

```
%applying initial conditons at t=0-------------------------
% 1. declare u, v and Y  --> set IC to 0
u = zeros(M+1, N+2);
v = zeros(M+2, N+1);
Y = zeros(M+6, N+6);


% 2. apply BCs to u, v and Y at T=0
u = bc_u(u, t);
v = bc_v(v, t);
Y = bc_Y3(Y, t);


% 3. correct outlet velocities to ensure volume conservation
[u, v] = correctOutlet(u,v);


% 4. calculate right hand side of Poisson equation using Δt = 1
dt = 1;
f = (1/dt)*calcDivV(u, v);


% 5. solve Poisson equation for Lagrange multiplier using Neumann boundary conditions
phi = ones(M+2, N+2); %inialized phi = 1
phi = bcGS(phi);
[phi, Linf, iter] = myPoisson(phi, f, h, nIterMax, epsilon);


% 6. project velocities using Lagrange multiplier using Δt = 1
[u, v]= projectV(u, v, phi, dt);


% 7.apply boundary conditions to ghost cell velocities only using initial time
u = bcGhost_u(u, t);
v = bcGhost_v(v, t);


%Adams-bashforth requires hyperbolic terms from previous step -->
%initialize these previous terms for n-1 using values at n
dt  = calcDtBurgers561(t, times_of_interest(2), u, v);
HY_prev = hyperbolic_Y_WENO_2D(Y, u, v, u, v, dt);
[Hu_prev,Hv_prev] = hyperbolic_uv_2D(u,v);


% 8. Start time loop------------------------------------


ifig = 1; % figure counter
numTimeSteps =1; %number of time steps counter
outputFlag = 0;


%calculate initial k and S
%calculate k and s
k_values(1) = calck(u, v);
S_values(1) = calcS3(Y);
Time_values(1) = t;
```

```
for k = 1:length(times_of_interest)

    while(~outputFlag)
    %1. find dt using Adam bashforth limitation
    [dt, outputFlag]  = calcDtBurgers561(t, times_of_interest(k), u, v);

    %2. use CN to solve Stokes Momentum eq w/o pressure
        % we will first use adam-bashforth to calculate the hyperbolic
        % terms for u and v, add those to the souce terms and then use CN
        % to find u and v. Then we will repeat this process, but for the Y
        % terms


        %record previous u and v terms
        u_prev = u;
        v_prev = v;


        %Find q at t_n
        [Qu, Qv, QY_n] = calcSourceIBFinal(u,v,Y,t,dt);


        %Hyperbolic terms will be added to source terms to "freeze the hyperbolic term in time during ADI"
        [Hu,Hv] = hyperbolic_uv_2D(u,v);
        Qu = Qu + (1.5)*Hu - (0.5)*Hu_prev;
        Qv = Qv + (1.5)*Hv - (0.5)*Hv_prev;

        %perform step one of ADI CN
        u = parabolic_CN1_2D_u(u, Qu, dt);
        v = parabolic_CN1_2D_v(v, Qv, dt);


        %Find q at t_n+1/2
        [Qu, Qv, QY_n12] = calcSourceIBFinal(u,v,Y,t+dt/2,dt);
        Qu = Qu + (1.5)*Hu - (0.5)*Hu_prev;
        Qv = Qv + (1.5)*Hv - (0.5)*Hv_prev;


        %perform step 2 of ADI CN
        u = parabolic_CN2_2D_u(u, Qu, dt);
        v = parabolic_CN2_2D_v(v, Qv, dt);

        %%calculate Y parabolic terms
        HY = hyperbolic_Y_WENO_2D(Y, u_prev, v_prev, u, v, dt);
        QY = QY_n + HY;

        Y = parabolic_CN1_2D_Y3(Y, QY, dt);

        QY = QY_n12 + HY;


        Y = parabolic_CN2_2D_Y3(Y, QY, dt);

        %increment t by half time step
        t = t+dt;


        %update previous hyperbolic terms
        Hu_prev = Hu;
        Hv_prev = Hv;
        HY_prev = HY;


    % 3. apply boundary conditions to boundaries and ghost cells using tn+1
        u = bc_u(u, t);
        v = bc_v(v, t);
        Y = bc_Y3(Y, t);


    % 4. correct outlet velocities to ensure volume conservation
        [u, v] = correctOutlet(u,v);


    % 5. calculate right hand side of Poisson equation
        f= (1/dt)*calcDivV(u, v);


    % 6. solve Poisson equation for Lagrange multiplier using Neumann boundary conditions
        phi = bcGS(phi); %note we are using the same phi as in the last step... should reduce number of iters needed
        [phi, Linf, iter] = myPoisson(phi, f, h, nIterMax, epsilon);


    % 7.project velocities using Lagrange multiplier
        [u, v]= projectV(u, v, phi, dt);


    % 8. apply boundary conditions to ghost cell velocities only using tn+1
        u = bcGhost_u(u, t);
        v = bcGhost_v(v, t);


    %9. repeat till time step reached

        %increment number of time steps taken
        numTimeSteps = numTimeSteps+1;

        %calculate k and s
        k_values(numTimeSteps) = calck(u, v);
        S_values(numTimeSteps) = calcS3(Y);

        %store time
        Time_values(numTimeSteps) = t;

        %check if k, s and times are going to be too short --> if so increase
        %preallocated memory size
         if( numTimeSteps+(BlockSize/10) > currentSize )  % less than 10%*BLOCK_SIZE free slots
            currentSize = currentSize + BlockSize;       % add new BLOCK_SIZE slots
            k_values(numTimeSteps+1:currentSize) = 0;
            S_values(numTimeSteps+1:currentSize) = 0;
            Time_values(numTimeSteps+1:currentSize) = 0;
            t
         end
    end


    outputFlag = 0;
```

```
    if(k~=length(times_of_interest))
        %plot u
        examFig1 = figure(1);
        subplot(3,3,ifig);
        pcolor(xf, yc, u')
        shading interp;
        ylim([0,2]);
        xlim([0,3]);
        clim([-2.5, 2.5]);
        set(gca,'fontsize',16);
        xlabel('x'); ylabel('y');
        title(strcat('u at time =', num2str(times_of_interest(k))));
        colormap jet;
        colorbar;

         %plot v
        examFig2 = figure(2);
        subplot(3,3,ifig);
        pcolor(xc, yf, v')
        shading interp;
        ylim([0,2]);
        xlim([0,3]);
        clim([-2.5, 2.5]);
        set(gca,'fontsize',16);
        xlabel('x'); ylabel('y');
        title(strcat('v at time =', num2str(times_of_interest(k))));
        colormap jet;
        colorbar;

         %plot Y
        examFig3 = figure(3);
        subplot(3,3,ifig);
        pcolor(xc3, yc3, Y')
        shading interp;
        ylim([0,2]);
        xlim([0,3]);
        clim([0 1]);
        set(gca,'fontsize',16);
        xlabel('x'); ylabel('y');
        title(strcat('Y at time =', num2str(times_of_interest(k))));
        colormap hot;
        colorbar;


        %increment counter
        ifig = ifig+1;
    end

end


%plot k and S
examFig4 = figure(4);
plot(Time_values(1:numTimeSteps), k_values(1:numTimeSteps));
set(gca,'fontsize',16);
xlabel('time'); ylabel('k(t)');
```

```
title(' k(t) vs t');


examFig5 = figure(5);
plot(Time_values(1:numTimeSteps), S_values(1:numTimeSteps));
set(gca,'fontsize',16);
xlabel('time'); ylabel('S');
title('S vs t');
ylim([0, .25]);




K_bar = 0.1 * myTrapezoidal(Time_values(1:numTimeSteps), k_values(1:numTimeSteps))
R_bar = 0.1 * myTrapezoidal(Time_values(1:numTimeSteps), S_values(1:numTimeSteps))
```

GCI Calculator

```
format long


r = 2;
% f1 =0.373372713685821
% f2 =0.373375677478581
% f3 = 0.373411752534933
FSEC = 1.25;
f3= 9.86798754568237
f2= 9.91136520322797
f1= 9.93797586818314




p = log(abs(f3-f2)/abs(f2-f1))/log(r)
f_h0 = f1+((f1-f2)/(r^p -1))
GCI_12 = FSEC *(abs((f1-f2)/f1)/(r^p -1))
GCI_23 = FSEC *(abs((f2-f3)/f2)/(r^p -1))


AsymptoticConvergence = (GCI_12/GCI_23) * r^p


% %Final answer is fh=0 +/- GCI12 in percent
```

MAE 561-- Final Project --Movie

```
clear
clear all


clf reset


global h CFL Re Sc xf xc yf yc yc3 xc3 Lx Ly t;


%Declare some constants
M=192;
N=128;
CFL = 0.9;
Re = 500;
```

```
Sc = 20;
Lx=3;
Ly=2;


times_of_interest = .1:.1:10;


%preallocate space for movie
loops = length(times_of_interest);
YMov(loops) = struct('cdata',[],'colormap',[]);
uMov(loops) = struct('cdata',[],'colormap',[]);
vMov(loops) = struct('cdata',[],'colormap',[]);
quivMov(loops) = struct('cdata',[],'colormap',[]);


x_range = [0,Lx];
y_range = [0,Ly];


%h is equal in both x and y
h = (x_range(2)-x_range(1))/(M);


%poissons solver controls:
nIterMax = 100;
epsilon = (h)/100; %Q is first order in time, so epsilon needs only be less than h


%lets define xf, yf, yc, xc, sc3 and yc3;
xf = linspace(x_range(1),x_range(2),M+1)';
yf = linspace(y_range(1),y_range(2),N+1)';
xc = linspace(x_range(1)-h/2,x_range(1)+(x_range(2)-x_range(1))+h/2,M+2)';
yc = linspace(y_range(1)-h/2,y_range(1)+(y_range(2)-y_range(1))+h/2,N+2)';
yc3 = linspace(y_range(1)-5*h/2,y_range(1)+(y_range(2)-y_range(1))+5*h/2,N+6)';
xc3 = linspace(x_range(1)-5*h/2,x_range(1)+(x_range(2)-x_range(1))+5*h/2,M+6)';


%lets define some variables we will be storing
%we do not know the length of the vectore before hand. Thus we will
%preallocate memory in blocks and add blocks as needed
BlockSize = 1000;% size increments added to vector length
currentSize = BlockSize; %current length of vectors
k_values = zeros(BlockSize,1);
S_values = zeros(BlockSize,1);
Time_values = zeros(BlockSize,1);


%declare t
t = 0;


%applying initial conditons at t=0--------------------------
% 1. declare u, v and Y  --> set IC to 0
u = zeros(M+1, N+2);
v = zeros(M+2, N+1);
Y = zeros(M+6, N+6);
```

```
% 2. apply BCs to u, v and Y at T=0
u = bc_u(u, t);
v = bc_v(v, t);
Y = bc_Y3(Y, t);


% 3. correct outlet velocities to ensure volume conservation
[u, v] = correctOutlet(u,v);


% 4. calculate right hand side of Poisson equation using Δt = 1
dt = 1;
f = (1/dt)*calcDivV(u, v);


% 5. solve Poisson equation for Lagrange multiplier using Neumann boundary conditions
phi = ones(M+2, N+2); %inialized phi = 1
phi = bcGS(phi);
[phi, Linf, iter] = myPoisson(phi, f, h, nIterMax, epsilon);


% 6. project velocities using Lagrange multiplier using Δt = 1
[u, v]= projectV(u, v, phi, dt);


% 7.apply boundary conditions to ghost cell velocities only using initial time
u = bcGhost_u(u, t);
v = bcGhost_v(v, t);


%Adams-bashforth requires hyperbolic terms from previous step -->
%initialize these previous terms for n-1 using values at n
dt  = calcDtBurgers561(t, times_of_interest(2), u, v);
HY_prev = hyperbolic_Y_WENO_2D(Y, u, v, u, v, dt);
[Hu_prev,Hv_prev] = hyperbolic_uv_2D(u,v);


% 8. Start time loop------------------------------------


numTimeSteps =1; %number of time steps counter
outputFlag = 0;


%calculate initial k and S
%calculate k and s
k_values(1) = calck(u, v);
S_values(1) = calcS3(Y);
Time_values(1) = t;


for k = 1:length(times_of_interest)

    while(~outputFlag)
    %1. find dt using Adam bashforth limitation
    [dt, outputFlag]  = calcDtBurgers561(t, times_of_interest(k), u, v);

    %2. use CN to solve Stokes Momentum eq w/o pressure
        % we will first use adam-bashforth to calculate the hyperbolic
        % terms for u and v, add those to the souce terms and then use CN
        % to find u and v. Then we will repeat this process, but for the Y
        % terms


        %record previous u and v terms
        u_prev = u;
        v_prev = v;


        %Find q at t_n
        [Qu, Qv, QY_n] = calcSourceIBFinal(u,v,Y,t,dt);


        %Hyperbolic terms will be added to source terms to "freeze the hyperbolic term in time during ADI"
        [Hu,Hv] = hyperbolic_uv_2D(u,v);
        Qu = Qu + (1.5)*Hu - (0.5)*Hu_prev;
        Qv = Qv + (1.5)*Hv - (0.5)*Hv_prev;

        %perform step one of ADI CN
        u = parabolic_CN1_2D_u(u, Qu, dt);
        v = parabolic_CN1_2D_v(v, Qv, dt);


        %Find q at t_n+1/2
        [Qu, Qv, QY_n12] = calcSourceIBFinal(u,v,Y,t+dt/2,dt);
        Qu = Qu + (1.5)*Hu - (0.5)*Hu_prev;
        Qv = Qv + (1.5)*Hv - (0.5)*Hv_prev;


        %perform step 2 of ADI CN
        u = parabolic_CN2_2D_u(u, Qu, dt);
        v = parabolic_CN2_2D_v(v, Qv, dt);

        %%calculate Y parabolic terms
        HY = hyperbolic_Y_WENO_2D(Y, u_prev, v_prev, u, v, dt);
        QY = QY_n + HY;

        Y = parabolic_CN1_2D_Y3(Y, QY, dt);

        QY = QY_n12 + HY;


        Y = parabolic_CN2_2D_Y3(Y, QY, dt);

        %increment t by half time step
        t = t+dt;


        %update previous hyperbolic terms
        Hu_prev = Hu;
        Hv_prev = Hv;
        HY_prev = HY;


      % 3. apply boundary conditions to boundaries and ghost cells using tn+1
        u = bc_u(u, t);
        v = bc_v(v, t);
        Y = bc_Y3(Y, t);
```

```
% 4. correct outlet velocities to ensure volume conservation
    [u, v] = correctOutlet(u,v);


% 5. calculate right hand side of Poisson equation
    f= (1/dt)*calcDivV(u, v);


% 6. solve Poisson equation for Lagrange multiplier using Neumann boundary conditions
    phi = bcGS(phi); %note we are using the same phi as in the last step... should reduce number of iters needed
    [phi, Linf, iter] = myPoisson(phi, f, h, nIterMax, epsilon);


% 7.project velocities using Lagrange multiplier
    [u, v]= projectV(u, v, phi, dt);


% 8. apply boundary conditions to ghost cell velocities only using tn+1
    u = bcGhost_u(u, t);
    v = bcGhost_v(v, t);


%9. repeat till time step reached

    %increment number of time steps taken
    numTimeSteps = numTimeSteps+1;

    %calculate k and s
    k_values(numTimeSteps) = calck(u, v);
    S_values(numTimeSteps) = calcS3(Y);

    %store time
    Time_values(numTimeSteps) = t;

    %check if k, s and times are going to be too short --> if so increase
    %preallocated memory size
     if( numTimeSteps+(BlockSize/10) > currentSize )  % less than 10%*BLOCK_SIZE free slots
        currentSize = currentSize + BlockSize;       % add new BLOCK_SIZE slots
        k_values(numTimeSteps+1:currentSize) = 0;
        S_values(numTimeSteps+1:currentSize) = 0;
        Time_values(numTimeSteps+1:currentSize) = 0;
     end
 end

 outputFlag = 0;





    %plot u
    examFig1 = figure(1);
    ax = gca;
    ax.NextPlot = 'replaceChildren';
    pcolor(xf, yc, u')
    shading interp;
    ylim([0,2]);
    xlim([0,3]);
```

```
clim([-2.5, 2.5]);
set(gca,'fontsize',16);
xlabel('x'); ylabel('y');
title(strcat('u at time =', num2str(times_of_interest(k))));
colormap jet;
colorbar;
drawnow
uMov(k) = getframe(examFig1);


 %plot v
examFig2 = figure(2);
ax = gca;
ax.NextPlot = 'replaceChildren';
pcolor(xc, yf, v')
shading interp;
ylim([0,2]);
xlim([0,3]);
clim([-2.5, 2.5]);
set(gca,'fontsize',16);
xlabel('x'); ylabel('y');
title(strcat('v at time =', num2str(times_of_interest(k))));
colormap jet;
colorbar;
drawnow
vMov(k) = getframe(examFig2);

%plot Y
examFig3 = figure(3);
ax = gca;
ax.NextPlot = 'replaceChildren';
pcolor(xc3, yc3, Y')
shading interp;
ylim([0,2]);
xlim([0,3]);
clim([0 1]);
set(gca,'fontsize',16);
xlabel('x'); ylabel('y');
title(strcat('Y at time =', num2str(times_of_interest(k))));
colormap hot;
colorbar;
drawnow
YMov(k) = getframe(examFig3);


examFig4 = figure(4);
ax = gca;
ax.NextPlot = 'replaceChildren';
pcolor(xc3, yc3, Y')
shading interp;
clim([0 1]);
colormap hot;
colorbar;
hold on
quiv = quiver(xf,yf, u(1:M+1,1:N+1)',v(1:M+1,1:N+1)',2);
% [X_mesh, Y_mesh] = meshgrid(xf,yf);
% verts = stream2(X_mesh,Y_mesh,u(1:M+1,1:N+1)',v(1:M+1,1:N+1)',3*ones([length(yf), 1]), yf);
% streamline(verts)
ylim([0,2]);
xlim([0,3]);
```

```
        set(gca,'fontsize',16);
        xlabel('x'); ylabel('y');
        title(strcat('Quivers at time =', num2str(times_of_interest(k))));
        hold off
        drawnow
        quivMov(k) = getframe(examFig4);


end


fig1 = figure
movie(fig1, uMov);


fig2 = figure
movie(fig2, vMov);


fig3 = figure
movie(fig3, YMov);


fig4 = figure
movie(fig4, quivMov);







% K = 0.1 * myTrapezoidal(Time_values(1:numTimeSteps), k_values(1:numTimeSteps))
% R = 0.1 * myTrapezoidal(Time_values(1:numTimeSteps), S_values(1:numTimeSteps))


function I = myTrapezoidal(x, y)
    I = 0;
    for i = 2:length(x)
        %perform composite trapazoidal integration
        I = I + ( 0.5 * (x(i) - x(i-1)) * (y(i-1)+y(i)) );
    end


end
function [Hu,Hv] = hyperbolic_uv_2D(u,v)
    global h
    %Solves for convective terms in the 2d- viscous burgers' eq on a
    %staggered mesh (see mod 7 for details) when moved to the RHS of the equation

    %get M and N from v
    M = size(v,1)-2; % in x dir
    N = size(v,2)-1;  %y dir

    %lets declare Hu and Hv arrays for staggered mesh
    Hu = zeros(M+1, N+2);
    Hv = zeros(M+2, N+1);

    %calculate Hu on interior points only
    for j = 2:N+1
```

```
        for i = 2:M

            duu_dx =-( (0.5*(u(i+1,j)+u(i,j)))^2 - (0.5*(u(i,j)+u(i-1,j)))^2 ) / h;
            duv_dy =-( (0.5*(u(i,j)+u(i,j+1)))*(0.5*(v(i,j)+v(i+1,j))) - (0.5*(u(i,j-1)+u(i,j)))*(0.5*(v(i,j-1)+v(i+1,j-1))) ) / h;

            Hu(i,j) = duu_dx + duv_dy;
        end
    end

    %calculate Hv on interior points only
    for j = 2:N
        for i = 2:M+1

            dvv_dy =-((0.5*(v(i,j) + v(i,j+1)))^2 - (0.5*(v(i,j)+v(i,j-1)))^2 ) / h;
            duv_dx =-( (0.5*(v(i,j)+v(i+1,j)))*(0.5*(u(i,j)+u(i,j+1))) - (0.5*(v(i,j)+v(i-1,j)))*(0.5*(u(i-1,j)+u(i-1,j+1))) ) / h;

            Hv(i,j) = dvv_dy + duv_dx;
        end
    end

end

function HY = hyperbolic_Y_WENO_2D(Y, u, v, u1, v1, dt)
    global t

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

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

    %Generate a0, a1, a2, b0, b1 and b2 at cell centers

    a0(4:M+3,4:N+3) = (u(1:M, 2:N+1) + u(2:M+1,2:N+1))/2;
    a1(4:M+3,4:N+3) = (u1(1:M, 2:N+1) + u1(2:M+1,2:N+1))/2;
    b0(4:M+3,4:N+3) = (v(2:M+1, 1:N) + v(2:M+1, 2:N+1))/2;
    b1(4:M+3,4:N+3)= (v1(2:M+1, 1:N) + v1(2:M+1, 2:N+1))/2;

    a2 =(a0+a1) / 2;
    b2 =(b0+b1) / 2;

    %assuming BC have previously been applied to Y
    %step 1
    dphi_n_dx = calc_adYdx_WENO_2D(Y, a0);
    dphi_n_dy = calc_bdYdy_WENO_2D(Y, b0);

    Y1 = Y - dt * (dphi_n_dx + dphi_n_dy);
    Y1 = bc_Y3(Y1, t+dt);

    %step 2
    dphi_1_dx = calc_adYdx_WENO_2D(Y1, a1);
    dphi_1_dy = calc_bdYdy_WENO_2D(Y1, b1);

    Y2 = Y1 +(3/4)*dt*(dphi_n_dx + dphi_n_dy) - (1/4)*dt*(dphi_1_dx + dphi_1_dy);
    Y2 = bc_Y3(Y2, t+(1/2)*dt);

    %step 3
    dphi_2_dx=calc_adYdx_WENO_2D(Y2, a2);
    dphi_2_dy=calc_bdYdy_WENO_2D(Y2, b2);
    Ys = Y2+(1/12)*dt*(dphi_n_dx + dphi_n_dy) + (1/12)*dt*(dphi_1_dx + dphi_1_dy) - (2/3)*dt*(dphi_2_dx + dphi_2_dy);
    Ys = bc_Y3(Ys, t+dt);

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

End

function [u, v] = correctOutlet(u,v)
    global h yc ucorr;

    %from problem definition --> outlet on bottom face for 1<x<2.5
    OutletYLimits = [.25, .75];

    %get M and N from v
    M = size(v,1)-2; % in x dir
    N = size(v,2)-1;  %y dir

    % calculate q_dot*
    q_dot_star = h*(sum(u(1,2:N+1)) - sum(u(M+1,2:N+1)) + sum(v(2:M+1,1)) - sum(v(2:M+1,N+1)));

    %calculate u_corr
    ucorr = q_dot_star/ (OutletYLimits(2) - OutletYLimits(1));

    %correction will be applied to u at the outlet bc
    for j = 2:N+1
        if(yc(j)>OutletYLimits(1) && yc(j)<OutletYLimits(2))
            u(1,j) = u(1,j)-ucorr;
        end
    end

end

function k = calck(u, v)
    global h;

     %get size of Mesh
    M = size(u,1)-1;
    N = size(u,2)-2;

    %perform 2d composite midpoint integration
    k = 0;
    for j = 2:N+1
        for i = 1:M
            %calculate kx
            k = k + ((u(i+1,j) + u(i, j) )/2)^2;
        end
    end

    for j = 1:N
        for i = 2:M+1
            %calculate kx
            k = k + ((v(i,j+1) + v(i, j) )/2)^2;
        end
    end


    k = 0.5*k*h^2;
end
```

```
function [S] = calcS3(Y)
   global Lx Ly h;


       %get size of Mesh
   M = size(Y,1)-6;
   N = size(Y,2)-6;

   S = 0;

   for j = 4:N+3
      for i = 4:M+3
         %calculate S
         S = S + Y(i,j)*(1-Y(i,j));
      end
   end


   S = S*(h^2) / (Lx*Ly);
end
function [dt, outputFlag] =  calcDtBurgers561(t,  outputTime, u, v)
   global h CFL

      %initialize output flag as 0
      outputFlag = 0;

      %calculate max stable dt step for u and v
      dt_max_u =h/(2*max(max(abs(u))) + max(max(abs(v))));
      dt_max_v =h/(max(max(abs(u))) + 2*max(max(abs(v))));

      dt_max = min(dt_max_u, dt_max_v);

      %apply security factor
      dt = dt_max*CFL;

      %check to see if we overshot our output time

      if (t < outputTime) && (t + dt >= outputTime)
         outputFlag = 1;
         dt = outputTime-t;
      end

end

function divV = calcDivV(u, v)
   global h;

    %get size of u
   M = size(u,1)-1; %node based in x dir
   N = size(u,2)-2; %cell centered in y dir

   %initialized divV withj zeros for all cells (including ghost cells)
   divV = zeros(M+2, N+2);

   for j = 2:N+1
      for i = 2:M+1
         %calculate 2nd order finite diff for divV
         divV(i,j) = ((u(i,j) - u(i-1,j) ) / (h)) +((v(i,j) - v(i,j-1) ) / (h)) ;

      end
   end
```

```
end
function [a, b, c, d] = bcCN2_v(a, b, c, d, t)

   global xc;

   %get size of v
   M = size(a,1)-2; %x dir
   N = size(a,2)-1; %y dir

   %for each col
   for i = 2:M+1

      %apply bottom BC
      if((xc(i) >= 1.5) && (xc(i) <= 2) ) %inlet condition
         z2 = 2*(xc(i)-1.5);
         g2 = sqrt(3)*6*z2*(1-z2);

         d(i,2) = d(i,2) - g2* a(i,2);
         a(i,2) = 0;

         %b and c remains unchanged
      else %wall condition
         a(i,2) = 0;
         %b, c and d remain unchanged
      end


      %apply top BC
      if((xc (i) >= 2) && (xc(i) <= 2.5) ) %inlet condition
         z3 = 2*(xc(i)-2);
         g3 = -9*z3*(1-z3);

         d(i,N) = d(i,N) - g3*c(i,N);
         c(i,N) = 0;
         %a and b remains unchanged

      else %wall condition
         c(i,N) = 0;
         %b, a and d remain unchanged
      end
   end
end
function v = bc_v(v, t)
   global xc yf;
   % xc: x-cordinates of cell centers incl. ghost cells
   % yf: y-cordinates of cell faces

   %get size of v
   M = size(v,1)-2; % in x dir
   N = size(v,2)-1;  %y dir


   for j = 1:N+1

      %left BC
      if ( (yf(j) >= 0.25) && (yf(j) <= 0.75) ) %outlet condition
         v(1,j) = v(2,j);
      else                          %wall condition
         v(1,j) = -v(2,j);
      end
```

```
        %Right Bc
        v(M+2,j) = -v(M+1,j);
    end

    for i = 1:M+2

        %Bottom BC
       if ( (xc(i) >= 1.5) && (xc(i) <= 2) ) %inlet condition
           z2 = 2*(xc(i)-1.5);
           g2 = sqrt(3)*6*z2*(1-z2);

           v(i,1) = g2;
        else                            %wall condition
           v(i,1) = 0;
       end


       %Top BC
       if ( (xc(i) >= 2) && (xc(i) <= 2.5) ) %inlet condition
           z3 = 2*(xc(i)-2);
           g3 = -9*z3*(1-z3);
           v(i,N+1) = g3;
       else                             %wall condition
           v(i,N+1) = 0;
       end

    end

end
function Y = parabolic_CN2_2D_Y3(Y, QY, dt)
   global Re Sc t h;

   %get size of y
    M = size(Y,1)-6;
    N = size(Y,2)-6;

    %declare a, b, c and d
    a = ones(M+6, N+6);
    b = ones(M+6, N+6);
    c = ones(M+6, N+6);
    d = zeros(M+6, N+6);

    %calculate d1/ d2 (equidistant mesh in both x and y, so both are equal
    %--> calling it dh arbitartily
    dh = dt/(2*Re*Sc*h^2);

    %setting a, b and c
    a= -dh*a;
    b = (1+2*dh)*b;
    c = -dh*c;

    %applying BCs to v at t_n+1/2
    Y = bc_Y3(Y, t+dt/2);

    %calculate d
    for j = 1:N+6
        for i = 2:M+5
           d(i,j) = dh*Y(i+1,j)+(1-2*dh)*Y(i,j)+dh*Y(i-1,j) + QY(i,j)*(dt/2);
        end
    end
    %apply implicit BCs for t_n+dt
    [a, b, c, d] = bcCN2_Y3(a, b, c, d, t+(dt));

    %loop over horizontal slices solving the tridiagonal system
    for i = 2:M+5
        Y(i,2:N+5) = mySolveTriDiag(a(i,2:N+5), b(i,2:N+5), c(i,2:N+5), d(i,2:N+5));
    end

    %applying BCs to v at t_n+dt
    Y = bc_Y3(Y, t+(dt));
end
function v = parabolic_CN2_2D_v(v, Qv, dt)
    global Re t h;

    %get size of v
    M = size(v,1)-2; % x dir
    N = size(v,2)-1; % y dir

    %declare a, b, c and d
    a = ones(M+2, N+1);
    b = ones(M+2, N+1);
    c = ones(M+2, N+1);
    d = zeros(M+2, N+1);

    %calculate d1/d2 (equidistant mesh in both x and y, so both are equal
    %--> calling it dh arbitartily
    dh = dt/(2*Re*h^2);

    %setting a, b and c
    a= -dh*a;
    b = (1+2*dh)*b;
    c = -dh*c;

    %applying BCs to v at t_n (t_n should be t+dt)
    v = bc_v(v, t+dt/2);

    %calculate d
    for j = 1:N+1
        for i = 2:M+1
            d(i,j) = dh*v(i+1,j)+(1-2*dh)*v(i,j)+dh*v(i-1,j) + Qv(i,j)*(dt/2);
        end
    end

    %apply implicit BCs for t_n+1/2
    [a, b, c, d] = bcCN2_v(a, b, c, d, t+dt);

    %loop over horizontal slices solving the tridiagonal system
    for i = 2:M+1
        v(i,2:N) = mySolveTriDiag(a(i,2:N), b(i,2:N), c(i,2:N), d(i,2:N));
    end

    %applying BCs to v at t_n+1/2
    v = bc_v(v, t+(dt));
end
function u = parabolic_CN2_2D_u(u, Qu, dt)
    global Re t h;

    %get size of u
    M = size(u,1)-1; %node based in x dir
    N = size(u,2)-2; %cell centered in y dir
    %declare a, b, c and d
    a = ones(M+1, N+2);
    b = ones(M+1, N+2);
    c = ones(M+1, N+2);
    d = zeros(M+1, N+2);

    %calculate d1/ d2 (equidistant mesh in both x and y, so both are equal
    %--> calling it dh arbitartily
    dh = dt/(2*Re*h^2);

    %setting a, b and c
    a= -dh*a;
    b = (1+2*dh)*b;
    c = -dh*c;

    %applying BCs to u at t_n+1/2
    u = bc_u(u, t+dt/2);

    %calculate d
    for j = 1:N+2
        for i = 2:M
            d(i,j) = dh*u(i+1,j)+(1-2*dh)*u(i,j)+dh*u(i-1,j) + Qu(i,j)*(dt/2);
        end
    end

    %apply implicit BCs for t_n+dt
    [a, b, c, d] = bcCN2_u(a, b, c, d, t+(dt));

    %loop over horizontal slices solving the tridiagonal system
    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

    %applying BCs to u at t_n+dt
    u = bc_u(u, t+(dt));
end
function Y = parabolic_CN1_2D_Y3(Y, QY, dt)
    global Re Sc t h;

   %get size of y
    M = size(Y,1)-6;
    N = size(Y,2)-6;


    %declare a, b, c and d
    a = ones(M+6, N+6);
    b = ones(M+6, N+6);
    c = ones(M+6, N+6);
    d = zeros(M+6, N+6);

    %calculate d1/ d2 (equidistant mesh in both x and y, so both are equal
    %--> calling it dh arbitartily
    dh = dt/(2*Re*Sc*h^2);

    %setting a, b and c
    a= -dh*a;
    b = (1+2*dh)*b;
    c = -dh*c;

    %applying BCs to v at t_n
    Y = bc_Y3(Y, t);
    %calculate d
    for j = 2:N+5
        for i = 1:M+6
            d(i,j) = dh*Y(i,j+1)+(1-2*dh)*Y(i,j)+dh*Y(i,j-1) + QY(i,j)*(dt/2);
        end
    end

    %apply implicit BCs for t_n+1/2
    [a, b, c, d] = bcCN1_Y3(a, b, c, d, t+(dt/2));

    %loop over horizontal slices solving the tridiagonal system
    for j = 1:N+6
        Y(2:M+5,j) = mySolveTriDiag(a(2:M+5,j), b(2:M+5,j), c(2:M+5,j), d(2:M+5,j));
    end

    %applying BCs to v at t_n+1/2
     Y = bc_Y3(Y, t+(dt/2));
end
function u = parabolic_CN1_2D_u(u, Qu, dt)
    global Re t h;

    %get size of u
    M = size(u,1)-1; %node based in x dir
    N = size(u,2)-2; %cell centered in y dir

    %declare a, b, c and d
    a = ones(M+1, N+2);
    b = ones(M+1, N+2);
    c = ones(M+1, N+2);
    d = zeros(M+1, N+2);

    %calculate d1/ d2 (equidistant mesh in both x and y, so both are equal
    %--> calling it dh arbitartily
    dh = dt/(2*Re*h^2);

    %setting a, b and c
    a= -dh*a;
    b = (1+2*dh)*b;
    c = -dh*c;

    %applying BCs to u at t_n
    u = bc_u(u, t);

    %calculate d
    for j = 2:N+1
        for i = 1:M+1
            d(i,j) = dh*u(i,j+1)+(1-2*dh)*u(i,j)+dh*u(i,j-1) + Qu(i,j)*(dt/2);
        end
    end

    %apply implicit BCs for t_n+1/2
    [a, b, c, d] = bcCN1_u(a, b, c, d, t+(dt/2));

    %loop over horizontal slices solving the tridiagonal system
    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

    %applying BCs to u at t_n+1/2
    u = bc_u(u, t+(dt/2));
```

```
end
function v = parabolic_CN1_2D_v(v, Qv, dt)
   global Re t h;

   %get size of u
   M = size(v,1)-2; %node based in x dir
   N = size(v,2)-1; %cell centered in y dir

   %declare a, b, c and d
   a = ones(M+2, N+1);
   b = ones(M+2, N+1);
   c = ones(M+2, N+1);
   d = zeros(M+2, N+1);

   %calculate d1/ d2 (equidistant mesh in both x and y, so both are equal
   %--> calling it dh arbitartily
   dh = dt/(2*Re*h^2);

   %setting a, b and c
   a= -dh*a;
   b = (1+2*dh)*b;
   c = -dh*c;

   %applying BCs to v at t_n
   v = bc_v(v, t);

   %calculate d
   for j = 2:N
      for i = 1:M+2
         d(i,j) = dh*v(i,j+1)+(1-2*dh)*v(i,j)+dh*v(i,j-1) + Qv(i,j)*(dt/2);
      end
   end

   %apply implicit BCs for t_n+1/2
   [a, b, c, d] = bcCN1_v(a, b, c, d, t+(dt/2));

   %loop over horizontal slices solving the tridiagonal system
   for j = 2:N
      v(2:M+1,j) = mySolveTriDiag(a(2:M+1,j), b(2:M+1,j), c(2:M+1,j), d(2:M+1,j));
   end

   %applying BCs to v at t_n+1/2
   v = bc_v(v, t+(dt/2));
end
function u = projectu(u, phi, dt)
   global t h;

    %get size of u
   M = size(u,1)-1; %node based in x dir
   N = size(u,2)-2; %cell centered in y dir

   for j = 2:N+1
      for i = 2:M
         %calculate u at n+1
         u(i,j) = u(i,j)-dt*(phi(i+1,j) - phi(i,j))/h;

      end
   end

   %apply BCs
   u = bcGhost_u(u, t);
```

```
end
function [u, v]= projectV(u, v, phi, dt)
   global t h;

    %get size of M and N
   M = size(u,1)-1;
   N = size(u,2)-2;

   for j = 2:N+1
      for i = 2:M
         %calculate u at n+1
         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
         %calculate v at n+1
         v(i,j) = v(i,j)-dt*(phi(i,j+1) - phi(i,j))/h;
      end
   end

   %apply BCs
   u = bcGhost_u(u, t);
   v = bcGhost_v(v, t);
end
function [psi] = psiWENO(a,b,c,d)
   epsilon = 10^-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-.5).*(b - 2.*c+d);
end
function Sol = mySolveTriDiag(a, b, c, d)
   % This functions uses gaussian elimination to solve tri-diagonal matices
   % See Slides 42 to 60 of module 2 to see detailed derivation of method


   % Check that all vectors are same length by attempting to concatonate all
   % vectors --> this will throw an error if vectors are not equal lengths
   try
      [a b c d];
   catch
      error("Vectors must all be same Length")
   end

   %find length of p
   p = length(a);

   %perform first elimination step
   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

    %perform back substitution step
    d(p) = d(p)/b(p);
    for i = (p-1):-1:1
        d(i) = (d(i) - c(i)*d(i+1))/b(i);
    end

    Sol = d;
end
function r2h = myRestrict(rh)
%Restrict function for cell centered mesh

    %get fine mesh dimensions
    M = size(rh,1)-2;
    N = size(rh,2)-2;

    %determine courser mesh size
    M_2h = M/2;
    N_2h = N/2;

    %delcare r2h with ghost cells
    r2h = zeros(M_2h+2, N_2h+2);

    for j = 2:N_2h+1
        for i = 2:M_2h+1

            % average fine mesh points to the coarse mesh
            r2h(i,j) = (1/4) * (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
function r = myResidual(phi, f, h)

    %get mesh size from input var (-2 to remove ghosts cells)
    [M,N] = size(phi);
    N = N - 2;
    M = M - 2;

    % declare / initialized r array
    r = zeros(size(phi));

    for j = 2:N+1 %loop over interior j cells
        for i = 2:M+1 %loop over interior i cells
            %use a second order central difference method to calculate residuals
            %r_ij = f_ij - (phi"_x_ij + phi"_y_ij)

            d2phidx2_ij= (phi(i+1,j) - 2* phi(i,j) +phi(i-1,j)) / (h^2) ;

            d2phidy2_ij= (phi(i,j+1) - 2* phi(i,j) +phi(i,j-1)) / (h^2) ;

            r(i,j) = f(i,j) - (d2phidx2_ij + d2phidy2_ij);
        end
    end

end
function rr = myRelResNorm(phi, f, h)
%calculates infinity norm of relative residual relative to f_inf. This is
%for a cell centered mesh
```

```
    %get
    [M, N] = size(phi);
    N = N - 2;
    M = M - 2;

    %calculate the resisduals using fucntion coded in problem 2
    r = myResidual(phi, f, h);

    %calulate infintiy norm of r and f
    r_inf_norm = max(max(abs(r(2:M+1 , 2:N+1))));

    f_inf_norm= max(max(abs(f(2:M+1 , 2:N+1))));

    %calculate relative residual norm
    rr = r_inf_norm/f_inf_norm;

end
function eh = myProlong(e2h)
%Prolongation function for cell centered mesh

    %get fine mesh dimensions
    M_2h = size(e2h,1)-2;
    N_2h = size(e2h,2)-2;

    %determine courser mesh size
    M_h = M_2h*2;
    N_h = N_2h*2;

    %delcare eh with ghost cells
    eh = zeros(M_h+2, N_h+2);

    for j = 2:N_2h+1
        for i = 2:M_2h+1

            %apply prolongation scheme for cell centered mesh
            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
function [phi, Linf, iter] = myPoisson(phi, f, h, nIterMax, epsilon)
%function to solve elliptical Poisson Equation using GaussSiedel Multigrid
%method on cell centered mesh


    phi = myMultigrid(phi, f, h); %perform at least 1 multigrid iteration
    Linf = myRelResNorm(phi, f, h); %calculate relative residual norm

    iter = 1; %we've performed 1 iteration

    while(iter < nIterMax &&  Linf>epsilon)
        phi = myMultigrid(phi, f, h); %perform multigrid iteration
        Linf = myRelResNorm(phi, f, h); %calculate relative residual norm
        iter = iter+1; %increment iteration counter
    end
```

```
end
function phi = myMultigrid(phi, f, h)
%multigrid recursive function for cell centered mesh
%essentially the same code from lecture notes

   %set some constants:
   n = 1;
   nc = 3;

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

   %perform n iters of GS algorithm
   phi = myGaussSeidel(phi,f,h,n);

   %if we are not on the coarsest possible mesh:
   if mod(M,2)==0 && mod(N,2)==0

      rh = myResidual(phi,f,h); %get phi residuals
      r2h = myRestrict(rh);   %restrict fine mesh
      e2h = zeros(M/2+2,N/2+2); %declare e2h cell centered mesh
      e2h = myMultigrid(e2h,r2h,2*h); %recursively run myMultigrid
      eh = myProlong(e2h); %prolong coarse mesh
      phi = phi + eh; % add error to phi to improve phi
      phi = bcGS(phi); %reapply BCs
      phi = myGaussSeidel(phi,f,h,n); %run GS again for n iters

   else
      %perform nc iters of GS algorithm at coarsest mesh
      phi = myGaussSeidel(phi,f,h,nc);

   end

end
function phi = myGaussSeidel(phi, f, h, ninter)

   %get mesh size from input var (-2 to remove ghosts cells)
   [M,N] = size(phi);
   N = N-2;
   M=M-2;

   %step 1 --> define solution domain
   %not including ghost cells:
   % x = linspace(ys-h/2, (ys-h/2)+(N+2)*h, N+2)
   % x = linspace(xs-h/2, (xs-h/2)+(M+2)*h, M+2)

   %step 2--> Define mesh --> cell centered
   %grid is M+2 x N+2 with corresponding i = 1...M+2, j = 1...N+2

   %step 3,4 --> skipping for now

   %step 5 --> apply BCs
   phi = bcGS(phi);

   %precalculate optimal omega
   rho_pj = 0.5*(cos(pi/M)+cos(pi/N));

   %Using same code as SOR algorithm, but with omega = 1
   omega =1;
```

```
    %Apply SOR algorithm to iteratively solve equation A*phi = b where A
    %and b are given from finite difference method

    for k = 1:ninter
        for j = 2:N+1 %loop over interior j indices
            for i = 2:M+1 %loop over interior i indices
                %update Phi with SOR scheme
                phi(i,j)=phi(i,j)+omega*((phi(i,j-1)+phi(i-1,j)+phi(i+1,j)+phi(i,j+1))/4-h^2*f(i,j)/4-phi(i,j));
            end
        end
         %make sure that BC are still maintained
         phi = bcGS(phi);
    end

end
function HY = hyperbolic_Y_WENO_2D(Y, u, v, u1, v1, dt)
    global t

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

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

    %Generate a0, a1, a2, b0, b1 and b2 at cell centers

    a0(4:M+3,4:N+3) = (u(1:M, 2:N+1) + u(2:M+1,2:N+1))/2;
    a1(4:M+3,4:N+3) = (u1(1:M, 2:N+1) + u1(2:M+1,2:N+1))/2;
    b0(4:M+3,4:N+3) = (v(2:M+1, 1:N) + v(2:M+1, 2:N+1))/2;
    b1(4:M+3,4:N+3)= (v1(2:M+1, 1:N) + v1(2:M+1, 2:N+1))/2;

    a2 =(a0+a1) / 2;
    b2 =(b0+b1) / 2;

    %assuming BC have previously been applied to Y
    %step 1
    dphi_n_dx = calc_adYdx_WENO_2D(Y, a0);
    dphi_n_dy = calc_bdYdy_WENO_2D(Y, b0);

    Y1 = Y - dt * (dphi_n_dx + dphi_n_dy);
    Y1 = bc_Y3(Y1, t+dt);

    %step 2
    dphi_1_dx = calc_adYdx_WENO_2D(Y1, a1);
    dphi_1_dy = calc_bdYdy_WENO_2D(Y1, b1);

    Y2 = Y1 +(3/4)*dt*(dphi_n_dx + dphi_n_dy) - (1/4)*dt*(dphi_1_dx + dphi_1_dy);
    Y2 = bc_Y3(Y2, t+(1/2)*dt);

    %step 3
    dphi_2_dx=calc_adYdx_WENO_2D(Y2, a2);
    dphi_2_dy=calc_bdYdy_WENO_2D(Y2, b2);

    Ys = Y2+(1/12)*dt*(dphi_n_dx + dphi_n_dy) + (1/12)*dt*(dphi_1_dx + dphi_1_dy) - (2/3)*dt*(dphi_2_dx + dphi_2_dy);
    Ys = bc_Y3(Ys, t+dt);

    %calculate HY
    HY = (Ys - Y)/dt;
```

```
end
function f = calc_bdYdy_WENO_2D(Y,b)
   global h;

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

   %declare f --  Set Ghost cells to zero (this will not be changed)
   f = zeros(M+6,N+6);
   dp = zeros(M+6,N+6);
   dmdp = zeros(M+6,N+6);

   % get dp and dmdp
   dp(4:M+3,1:N+5) = Y(4:M+3,2:N+6) - Y(4:M+3, 1:N+5);
   dmdp(4:M+3,2:N+5) = Y(4:M+3,3:N+6) - 2*Y(4:M+3, 2:N+5) + Y(4:M+3, 1:N+4);

   %calculate f
   for i = 4:M+3
      for j = 4:N+3

         %get WENO value --> will be added to dphidx
         if(b(i,j)>=0)
            WENO_val = psiWENO(dmdp(i, j-2)/h, dmdp(i, j-1)/h,dmdp(i, j)/h,dmdp(i, j+1)/h);
         else
            WENO_val = -psiWENO(dmdp(i, j+2)/h, dmdp(i, j+1)/h,dmdp(i, j)/h,dmdp(i, j-1)/h);
         end

         dphidx = (1/(12*h)) *(-dp(i, j-2)+ 7*dp(i, j-1) +7*dp(i, j) - dp(i, j+1)) - WENO_val;

         f(i,j) = b(i,j)*dphidx;

      end
   end

end
function f = calc_adYdx_WENO_2D(Y,a)
   global h;

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

   %declare f --  Set Ghost cells to zero (this will not be changed)
   f = zeros(M+6,N+6);
   dp = zeros(M+6,N+6);
   dmdp = zeros(M+6,N+6);

   % get dp and dmdp
   dp(1:M+5,4:N+3) = Y(2:M+6,4:N+3) - Y(1:M+5,4:N+3);
   dmdp(2:M+5, 4:N+3) = Y(3:M+6,4:N+3) - 2*Y(2:M+5,4:N+3) + Y(1:M+4,4:N+3);

   %calculate f
   for j = 4:N+3
      for i = 4:M+3

         %get WENO value --> will be added to dphidx
         if(a(i,j)>=0)
            WENO_val = psiWENO(dmdp(i-2,j)/h, dmdp(i-1,j)/h,dmdp(i,j)/h,dmdp(i+1,j)/h);
         else
            WENO_val = -psiWENO(dmdp(i+2,j)/h, dmdp(i+1,j)/h,dmdp(i,j)/h,dmdp(i-1,j)/h);
            end

            dphidx = (1/(12*h)) *(-dp(i-2,j)+ 7*dp(i-1,j) +7*dp(i,j) - dp(i+1,j)) - WENO_val;

            f(i,j) = a(i,j)*dphidx;

        end
    end

end
function f =  calc_adYdx_WENO(Y, a)

    global h;

    %get size of Y
    M = size(Y,1)-6;

    %declare f
    f = zeros( M+6,1);
    dp = zeros( M+6,1);
    dmdp = zeros( M+6,1);

    % get dp and dmdp
    dp(1:M+5) = Y(2:M+6) - Y(1:M+5);
    dmdp(2:M+5) = Y(3:M+6) - 2*Y(2:M+5) + Y(1:M+4);


    %calculate f
    for i = 4:M+3

        %get WENO value --> will be added to dphidx
        if(a(i)>=0)
            WENO_val = psiWENO(dmdp(i-2)/h, dmdp(i-1)/h,dmdp(i)/h,dmdp(i+1)/h);
        else
            WENO_val = -psiWENO(dmdp(i+2)/h, dmdp(i+1)/h,dmdp(i)/h,dmdp(i-1)/h);
        end

        dphidx = (1/(12*h)) *(-dp(i-2)+ 7*dp(i-1) +7*dp(i) - dp(i+1)) - WENO_val;

        f(i) = a(i)*dphidx;

    end


end
function phi = bcGS(phi)

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


    %apply zero neumann condition to left bc
    phi(:,1) = phi(:,2);

    %apply zero neumann condition to right bc
    phi(:,N+2) = phi(:, N+1);

    %apply zero neumann condition top bc
    phi(M+2,:) = phi(M+1,:);
    %apply zero neumann condition to bottom bc
    phi(1,:) = phi(2,:);

end
function v = bcGhost_v(v, t)
    global yf;
    % xf: x-cordinates of cell faces

     %get size of v
    M = size(v,1)-2;
    N = size(v,2)-1;

    for j = 1:N+1

        %left BC
        if ( (yf(j) >= 0.25) && (yf(j) <= 0.75) ) %outlet condition
            v(1,j) = v(2,j);
        else                                %wall condition
            v(1,j) = -v(2,j);
        end

        %Right Bc
        v(M+2,j) = -v(M+1,j);
    end

end
function u = bcGhost_u(u, t)
    global xf;
    % xf: x-cordinates of cell faces

     %get size of u
    M = size(u,1)-1; %node based in x dir
    N = size(u,2)-2; %cell centered in y dir

 %Bottom BC -- y= 0, x=(0,3)--> defining u(i, 1) as we are cell centered
    for i = 1:M+1
        z2 = 2*(xf(i)-1.5);
        g2 = 6*z2*(1-z2);

        if ( (xf(i) > 1.5) && (xf(i) < 2) )
            u(i,1) = 2*g2-u(i,2);
        else % if x >= 2
            u(i,1) = -u(i,2);
        end
    end

    %Top BC -- y= 2, x=(0,3)--> defining u(i, N+2) as we are cell centered
    for i = 1:M+1
        z3 = 2*(xf(i)-2);
        g3 = sqrt(3)*(3/2)*6*z3*(1-z3);

        if ( (xf(i) > 2) && (xf(i) < 2.5) )
            u(i,N+2) = 2*g3-u(i,N+1);
        else % if x >= 2.5
            u(i,N+2) = -u(i,N+1);
        end
    end

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

```
    global xc3;

    %get size of y
    M = size(a,1)-6;
    N = size(a,2)-6;

    %for each col
    for i = 1:M+6
       %apply bottom BC
       if((xc3(i) >= 1.5) && (xc3(i) <= 2) ) %inlet condition
          b(i,2:4) = b(i,2:4)-a(i,2:4);
          a(i,2:4) = 0;
       else %wall condition
          b(i,2:4) = b(i,2:4)+a(i,2:4);
          a(i,2:4) = 0;
       end

       %apply top BC
       if((xc3 (i) >= 2) && (xc3(i) <= 2.5) ) %inlet condition
          b(i,N+3:N+5) = b(i,N+3:N+5)-c(i,N+3:N+5);
          c(i,N+3:N+5) = 0;
       else %wall condition
          b(i,N+3:N+5) = b(i,N+3:N+5)+c(i,N+3:N+5);
          c(i,N+3:N+5) = 0;
       end
    end
end
function [a, b, c, d] = bcCN2_v(a, b, c, d, t)

    global xc;

    %get size of v
    M = size(a,1)-2; %x dir
    N = size(a,2)-1; %y dir

    %for each col
    for i = 2:M+1

       %apply bottom BC
       if((xc(i) >= 1.5) && (xc(i) <= 2) ) %inlet condition
          z2 = 2*(xc(i)-1.5);
          g2 = sqrt(3)*6*z2*(1-z2);

          d(i,2) = d(i,2) - g2* a(i,2);
          a(i,2) = 0;

          %b and c remains unchanged
       else %wall condition
          a(i,2) = 0;
          %b, c and d remain unchanged
       end


       %apply top BC
       if((xc (i) >= 2) && (xc(i) <= 2.5) ) %inlet condition
          z3 = 2*(xc(i)-2);
          g3 = -9*z3*(1-z3);

          d(i,N) = d(i,N) - g3*c(i,N);
          c(i,N) = 0;
          %a and b remains unchanged
```

```
        else %wall condition
            c(i,N) = 0;
            %b, a and d remain unchanged
        end
    end
end
function [a, b, c, d] = bcCN2_u(a, b, c, d, t)

    global xf;

    %get size of u
    M = size(a,1)-1; %node based in x dir
    N = size(a,2)-2; %cell centered in y dir

    %for each col
    for i = 2:M

        %apply top BC
        if((xf(i) > 1.5) && (xf(i) < 2) ) %inlet condition
            z2 = 2*(xf(i)-1.5);
            g2 = 6*z2*(1-z2);

            d(i,2) = d(i,2) - 2*g2*a(i,2);
            b(i,2) = b(i,2) - a(i,2);
            a(i,2) = 0;
            %c remains unchanged
        else %wall condition
            b(i,2) = b(i,2) - a(i,2);
            a(i,2) = 0;
            %c and d remain unchanged
        end


        %apply bottom BC
        if((xf(i) > 2) && (xf(i) < 2.5) ) %inlet condition
            z3 = 2*(xf(i)-2);
            g3 = sqrt(3)*(3/2)*6*z3*(1-z3);

            d(i,N+1) = d(i,N+1) - 2*g3*c(i,N+1);
            b(i,N+1) = b(i,N+1) - c(i,N+1);
            c(i,N+1) = 0;
            %a remains unchanged

        else %wall condition
            b(i,N+1) = b(i,N+1) - c(i,N+1);
            c(i,N+1) = 0;
            %a and d remain unchanged
        end
    end
end
function [a, b, c, d] = bcCN1_Y3(a, b, c, d, t)

    global yc3;

    %get size of y
    M = size(a,1)-6;
    N = size(a,2)-6;

    %for each row
    for j = 1:N+6
```

```
        %apply left BC
        b(2:4,j) = b(2:4,j)+a(2:4,j);
        a(2:4,j) = 0;
        %c and d unchanged

        %apply right BC
        if ( (yc3(j) >= .25) && (yc3(j) <= 1.25) ) %inlet condition
            b(M+3:M+5, j) = b(M+3:M+5, j) -  c(M+3:M+5, j);
            d(M+3:M+5, j) = d(M+3:M+5, j) -  2* c(M+3:M+5, j);
            c(M+3:M+5, j) = 0;
            %a unchanged

        else %wall condition
            b(M+3:M+5, j) = b(M+3:M+5, j) +  c(M+3:M+5, j);
            c(M+3:M+5, j) = 0;
            %a and d unchanged
        end

    end
end
function [a, b, c, d] = bcCN1_v(a, b, c, d, t)

    global yf;

    %get size of v
    M = size(a,1)-2; %x dir
    N = size(a,2)-1; %y dir

    %for each row
    for j = 2:N
        %apply left BC
        if((yf(j) >= 0.25) && (yf(j) <= 0.75) ) %outlet condition
            b(2, j) = b(2, j) + a(2, j);
            a(2, j) = 0;
            %d and c unchanged
        else %wall condition
            b(2, j) = b(2, j) - a(2, j);
            a(2, j) = 0;
            %d and c unchanged
        end

        %apply right BC
        b(M+1, j) = b(M+1, j) - c(M+1, j);
        c(M+1, j) = 0;
        % d and a unchanged

    end
end
function [a, b, c, d] = bcCN1_u(a, b, c, d, t)

    global yc;

    %get size of u
    M = size(a,1)-1; %node based in x dir
    N = size(a,2)-2; %cell centered in y dir

    %for each row
    for j = 2:N+1
        %apply left BC
        if((yc(j) > 0.25) && (yc(j) < 0.75) ) %outlet condition
            b(2, j) = b(2, j) + (4/3)*a(2, j);
            c(2, j) = c(2, j) - (1/3)*a(2, j);
            a(2, j) = 0;
            %d2 unchanged
        else %wall condition
            a(2, j) = 0;
            %d2, b2, c2 unchanged
        end

        %apply right BC
        if((yc(j) > 0.25) && (yc(j) < 1.25) ) %inlet condition
            z1 = yc(j)-.25;
            g1 = -(3/4)*6*z1*(1-z1);

            d(M, j) = d(M, j) - c(M, j)*g1;
            c(M, j) = 0;

            % b_M, a_M unchanged
        else %wall condition
            c(M, j) = 0;
            %d_M, b_M, a_M unchanged
        end
    end
end
function Y = bc_Y3(Y,t)
    global xc3 yc3;

    %get size of y
    M = size(Y,1)-6;
    N = size(Y,2)-6;

    for j = 1:N+6 %run through all columns

        %left BC
        Y(3,j) = Y(4,j);
        Y(2,j) = Y(5,j);
        Y(1,j) = Y(6,j);

        %Right Bc
        if ( (yc3(j) >= .25) && (yc3(j) <= 1.25) ) %inlet condition
            Y(M+4,j) = 2-Y(M+3,j);
            Y(M+5,j) = 2-Y(M+2,j);
            Y(M+6,j) = 2-Y(M+1,j);
        else                                    %wall condition
            Y(M+4,j) = Y(M+3,j);
            Y(M+5,j) = Y(M+2,j);
            Y(M+6,j) = Y(M+1,j);
        end

    end

    for i = 1:M+6

        %Bottom BC
       if ( (xc3(i) >= 1.5) && (xc3(i) <= 2) ) %inlet condition
            Y(i,3) = -Y(i,4);
            Y(i,2) = -Y(i,5);
            Y(i,1) = -Y(i,6);
       else                                  %wall condition
            Y(i,3) = Y(i,4);
            Y(i,2) = Y(i,5);
            Y(i,1) = Y(i,6);
       end



       %Top BC
       if ( (xc3(i) >= 2) && (xc3(i) <= 2.5) ) %inlet condition
           Y(i,N+4) = -Y(i,N+3);
           Y(i,N+5) = -Y(i,N+2);
           Y(i,N+6) = -Y(i,N+1);
       else                                %wall condition
           Y(i,N+4) = Y(i,N+3);
           Y(i,N+5) = Y(i,N+2);
           Y(i,N+6) = Y(i,N+1);
       end

    end

end
function u = bc_u(u, t)
    global xf yc;
    % xf: x-cordinates of cell faces
    % yc: y-cordinates of cell centers incl. ghost cells

     %get size of u
    M = size(u,1)-1; %node based in x dir
    N = size(u,2)-2; %cell centered in y dir

    %left BC -- x= 0, y=(0,2) --> defining u(1.5, j) as we are node based
    for j = 1:N+2
        if yc(j) <= 0.25
            u(1,j) = 0;
        elseif ( (yc(j) > 0.25) && (yc(j) < 0.75) )
            u(1,j) = (-1/3)*(u(3,j) - 4*u(2,j)); %nuemann condition w/ 2nd order Forward diff
        else % if y >= .75
            u(1,j) = 0;
        end
    end

    %Right Bc -- x = 3, y=(0,2) --> defining u(M+3/2, j) as we are node based
    for j = 1:N+2
        z1 = yc(j)-.25;
        g1 = -(3/4)*6*z1*(1-z1);

        if yc(j) <= 0.25
            u(M+1,j) = 0;
        elseif ( (yc(j) > 0.25) && (yc(j) < 1.25) )
            u(M+1,j) = g1;
        else % if y >= 1.25
            u(M+1,j) = 0;
        end
    end

    %Bottom BC -- y= 0, x=(0,3)--> defining u(i, 1) as we are cell centered
    for i = 1:M+1
        z2 = 2*(xf(i)-1.5);
        g2 = 6*z2*(1-z2);

        if xf(i) <= 1.5
            u(i,1) = -u(i,2);
        elseif ( (xf(i) > 1.5) && (xf(i) < 2) )
            u(i,1) = 2*g2-u(i,2);
        else % if x >= 2
```

```
            u(i,1) = -u(i,2);
        end
    end

    %Top BC -- y= 2, x=(0,3)--> defining u(i, N+2) as we are cell centered
    for i = 1:M+1
        z3 = 2*(xf(i)-2);
        g3 = sqrt(3)*(3/2)*6*z3*(1-z3);

        if xf(i) <= 2
            u(i,N+2) = -u(i,N+1);
        elseif ( (xf(i) > 2) && (xf(i) < 2.5) )
            u(i,N+2) = 2*g3-u(i,N+1);
        else % if x >= 2.5
            u(i,N+2) = -u(i,N+1);
        end
    end

end
function [Y] = bc3(Y, t)

    %get size of Y
    M = size(Y,1)-6;

    %apply left BC


%for questions 10
% g = (21+5*sqrt(2)+20*cos(5*t*pi))/50;
    % Y(1) = 2*g - Y(6);
    % Y(2) = 2*g - Y(5);
    % Y(3) = 2*g - Y(4);
    %
    %
    % %apply right bc
    % Y(M+4) = Y(M+3);
    % Y(M+5) = Y(M+2);
    % Y(M+6) = Y(M+1);

    %for questions 11 --periodic BC

    Y(1) = Y(M+1);
    Y(2) = Y(M+2);
    Y(3) = Y(M+3);


    %apply right bc
    Y(M+4) = Y(4);
    Y(M+5) = Y(5);
    Y(M+6) = Y(6);

end
```