MAE 561 Final Report

Austin Goodrich

April 27, 2018

Contents

List of Tables

List of Figures

1 Problem Introduction

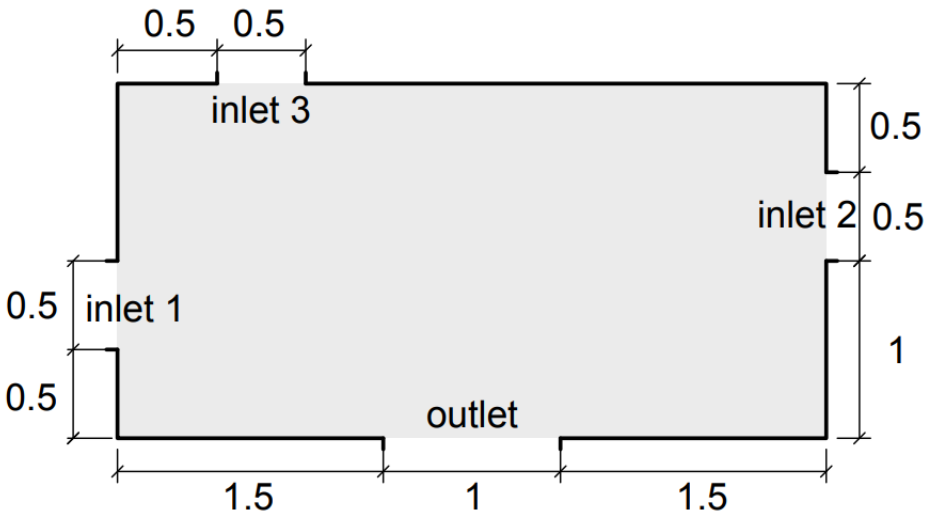

Figure 1: Mixing Chamber Geometry

Problem 1: The flow in a two-dimensional mixing chamber as depicted in Fig. 1 is modeled by the non-dimensional, incompressible, constant density, isothermal Navier-Stokes equations, the non-dimensional convection/diffusion equation for the mass fraction of a reactant Y, and the non-dimensional continuity equation. The fluid in the chamber with Reynolds number $Re = 50$ is initially at rest, and no reactant is present. The reactant has a Schmidt number of $Sc = 1$. The following equations describe the flow in the chamber.

$$\begin{aligned}
\frac{\partial u}{\partial x} + \frac{\partial v}{\partial y} &= 0 \\
\frac{\partial u}{\partial t} + u\frac{\partial u}{\partial x} + v\frac{\partial u}{\partial y} &= -\frac{\partial p}{\partial x} + \frac{1}{Re}\left(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2}\right) \\
\frac{\partial v}{\partial t} + u\frac{\partial v}{\partial x} + v\frac{\partial v}{\partial y} &= -\frac{\partial p}{\partial y} + \frac{1}{Re}\left(\frac{\partial^2 v}{\partial x^2} + \frac{\partial^2 v}{\partial y^2}\right) \\
\frac{\partial Y}{\partial t} + u\frac{\partial Y}{\partial x} + v\frac{\partial Y}{\partial y} &= \frac{1}{ReSc}\left(\frac{\partial^2 Y}{\partial x^2} + \frac{\partial^2 Y}{\partial y^2}\right)
\end{aligned} \tag{1}$$

where u is the non-dimensional velocity in the $x-$direction, v is the non-dimensional velocity in the $y-$direction, Y is the mass fraction of a chemical species. At the initial time, inlets 1, 2, and 3 are opened resulting in fully developed parabolic channel velocity profiles with constant Y and average velocities of

$$\begin{aligned}
I1: &\quad u_{avg,1} = 1, \quad v_{avg,1} = 0, \quad Y_{in,1} = 1 \\
I2: &\quad u_{avg,2} = -1, \quad v_{avg,2} = 0, \quad Y_{in,2} = 1 \\
I3: &\quad u_{avg,3} = 0, \quad v_{avg,3} = -1, \quad Y_{in,3} = 0
\end{aligned}$$

The outlet is to be modeled by

$$O: \qquad u_{out} = 0, \quad \frac{\partial v}{\partial y} = 0, \quad \frac{\partial Y}{\partial y} = 0 \tag{2}$$

All mixing chamber walls are no-slip and have zero transport of the chemical species, i.e.,

$$W: \qquad u_w = 0, \quad v_w = 0 \tag{3}$$

The performance of the mixing chamber chall be measured by the quantity $R(t)$ defined as

$$R(t) = \frac{1}{HL} \int_0^H \int_0^L Y(x,y,t)(1 - Y(x,y,t))dxdy \tag{4}$$

where H and L are the non-dimensional height and length of the mixing chamber.

2 Problem 1

The equations will be solved numerically using the Crank Nicolson Alternating Direction Implicit Fractional Step method to calculate the diffusive terms and Adams-Bashforth to calculate the hyperbolic terms of the momentum equations. Crank Nicolson ADI has an order of convergence of $O(\Delta t^2)$ in time and $O(\Delta x^2)$ in space and the Vcycle multigrid method used to solve the Poisson system is second order in space ($O(\Delta x^2)$) and it uses a Gauss-Seidel method which is also second order in space ($O(\Delta x^2)$). Adams-Bashforth has an order of convergence of $O(\Delta t^2)$ in time and $O(\Delta x^2)$ in space. The mass-fraction equations utilize WENO5-TVD-RK-3 and Crank Nicolson to numerically solve the equations. WENO5 is $O(\Delta x^5)$ in space and TVD-RK-3 is $O(\Delta t^3)$ in time. The overall order of convergence for the momentum and mass-fraction equations are given in table 1.

Table 1: Overall Order of Convergence for all equations

Equation	Spatial Order of Convergence	Temporal Order of Convergence
Momentum	$O(\Delta x^2)$	$O(\Delta t^2)$
Mass-Fraction	$O(\Delta x^2)$	$O(\Delta t^2)$

The maximum time step will be calculated with the following constraint.

$$\frac{\max_{i,j}(|a^n_{i,j}|)\Delta t}{\Delta x} + \frac{\max_{i,j}(|b^n_{i,j}|)\Delta t}{\Delta y} \leq 1 \tag{5}$$

First, the PDE's must be put in the following form.

$$\frac{\partial u}{\partial t} + a(u,v,Y,x,y,t)\frac{\partial u}{\partial x} + b(u,v,Y,x,y,t)\frac{\partial u}{\partial y} = 0 \tag{6}$$

Starting with the $u-$equation and ignoring the diffusive terms for now, the following equation is obtained.

$$\frac{\partial u}{\partial t} + u\frac{\partial u}{\partial x} + v\frac{\partial u}{\partial y} = 0 \tag{7}$$

Using $a = u$ and $b = v$, and applying it to equation 5, the following equation is obtained.

$$\frac{\max_{i+1/2,j}(|u^n_{i+1/2,j}|)\Delta t}{\Delta x} + \frac{\max_{i,j+1/2}(|v^n_{i,j+1/2}|)\Delta t}{\Delta y} \leq 1 \tag{8}$$

Resulting in the following expression to calculate the maximum time step for u which will be labeled Δt_u.

$$\Delta t_u \quad = \quad \frac{\Delta x \Delta y}{\Delta y \max_{i+1/2,j}(|u^n_{i+1/2,j}|) + \Delta x \max_{i,j+1/2}(|v^n_{i,j+1/2}|} \tag{9}$$

The same exercise is repeated for the $v-$equation.

$$\frac{\partial v}{\partial t} + u\frac{\partial v}{\partial x} + v\frac{\partial u}{\partial x} = 0 \tag{10}$$

Again using eq. 5, the following expression is used to calculate the maximum time step for v which will be labeled Δt_v.

$$\Delta t_v \quad = \quad \frac{\Delta x \Delta y}{\Delta y \max_{i+1/2,j}(|u^n_{i+1/2,j}|) + \Delta x \max_{i,j+1/2}(|v^n_{i,j+1/2}|} \tag{11}$$

The $Y-$equation is already in the proper form, so eq. 5 can be directly applied to the PDE to obtain the expression for Δt_Y, the maximum time step for Y.

$$\Delta t_Y \quad = \quad \frac{\Delta x \Delta y}{\Delta y \max_{i+1/2,j}(|u^n_{i+1/2,j}|) + \Delta x \max_{i,j+1/2}(|v^n_{i,j+1/2}|} \tag{12}$$

Finally, to keep all variables solving at the same time step, the true maximum time step will be the maximum of the three time steps as shown below.

$$\Delta t \quad = \quad \min(\Delta t_u, \Delta t_v, \Delta t_Y) \tag{13}$$

A CFL of 0.8 will be applied to the maximum time step in eq. 13 to obtain the time step used in the simulation as shown below.

$$\Delta t \quad = \quad CFL \cdot \min(\Delta t_u, \Delta t_v, \Delta t_Y) \tag{14}$$

The parameters listed in table 2 will be used to calculate the solutions in Section 3.

Table 2: Parameters Used in Solution Calculation

Parameter	Value
M	256
N	128
Δx	0.015625
Δy	0.015625
CFL	0.8

3 Results: Problem 1

3.1 Contour plots for u

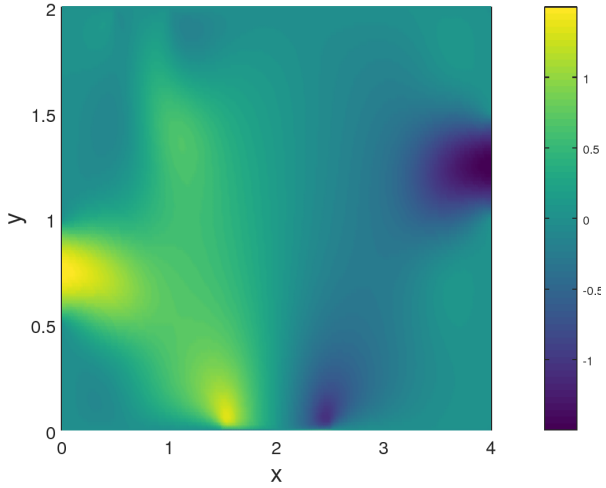

Figure 2: Solution for u at $t = 1.0$

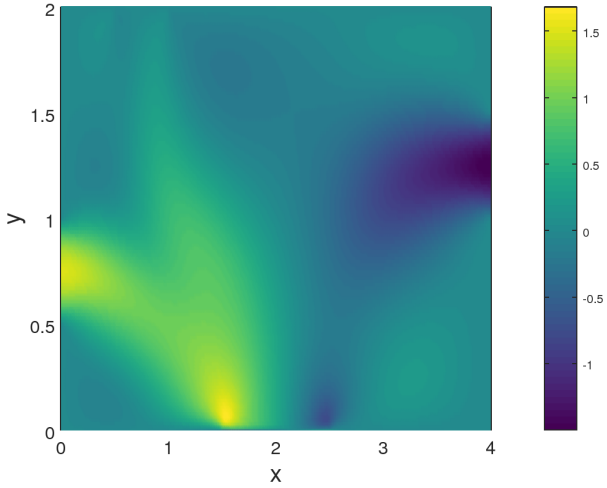

Figure 3: Solution for u at $t = 3.0$

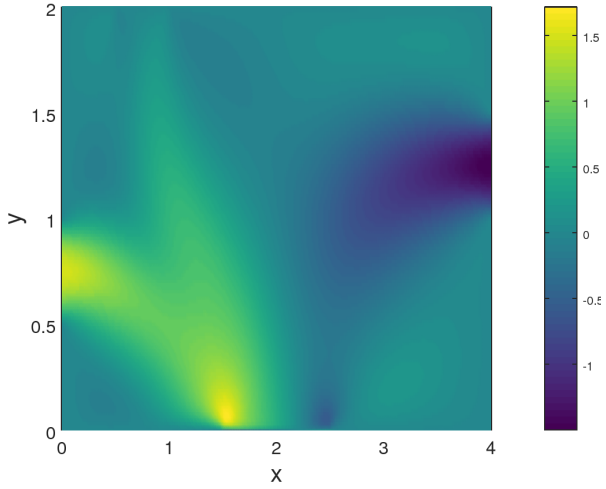

Figure 4: Solution for u at $t = 5.0$

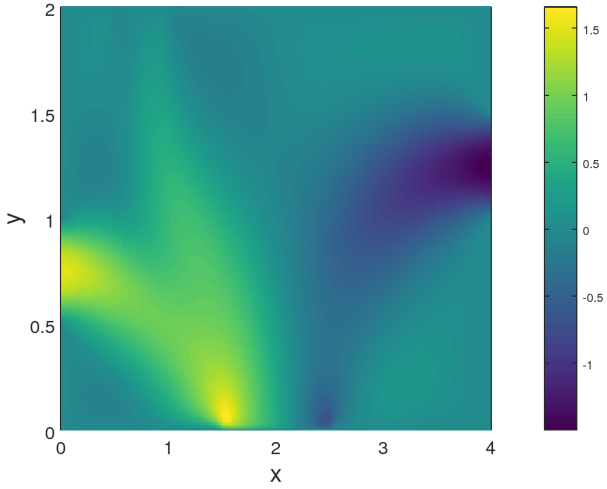

Figure 5: Solution for u at $t = 10.0$

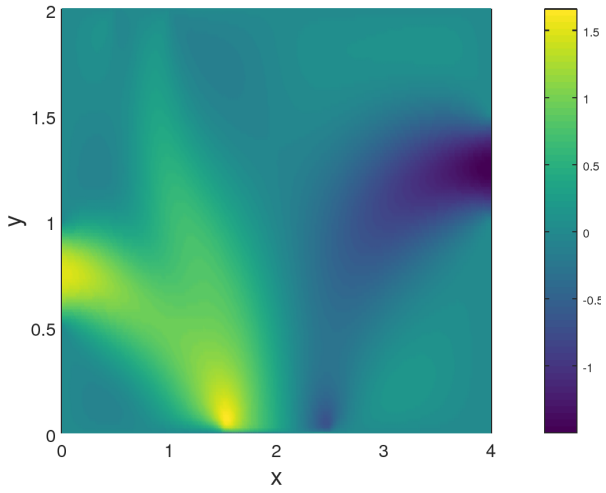

Figure 6: Solution for u at $t = 15.0$

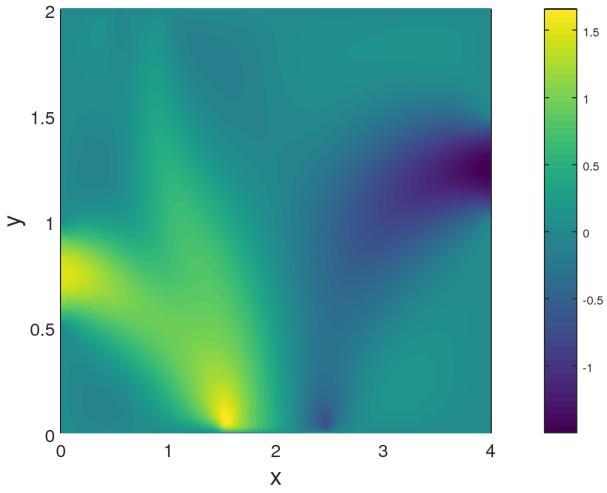

Figure 7: Solution for u at $t = 20.0$

3.2 Contour plots for v

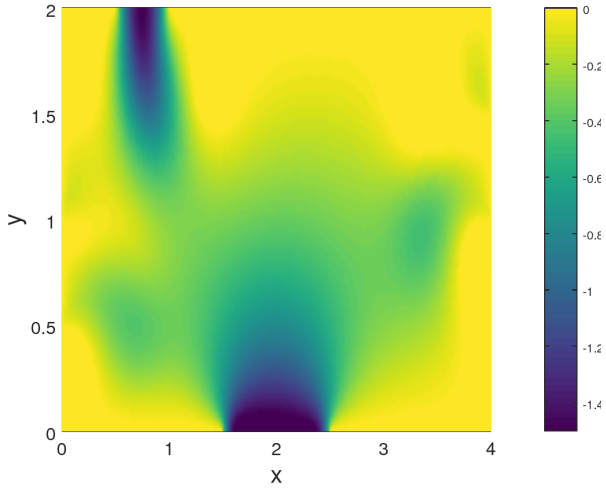

Figure 8: Solution for v at $t = 1.0$

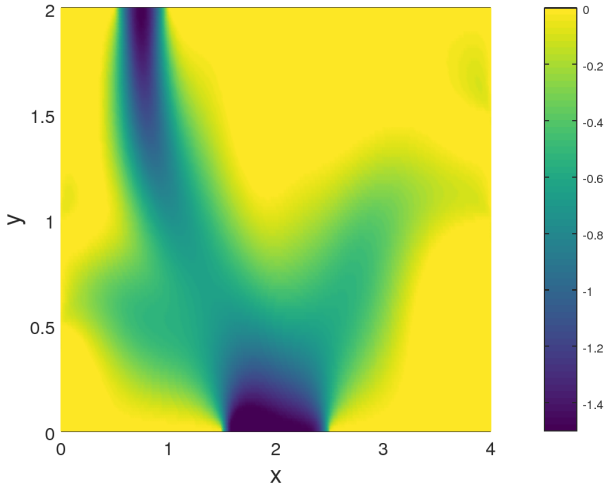

Figure 9: Solution for v at $t = 3.0$

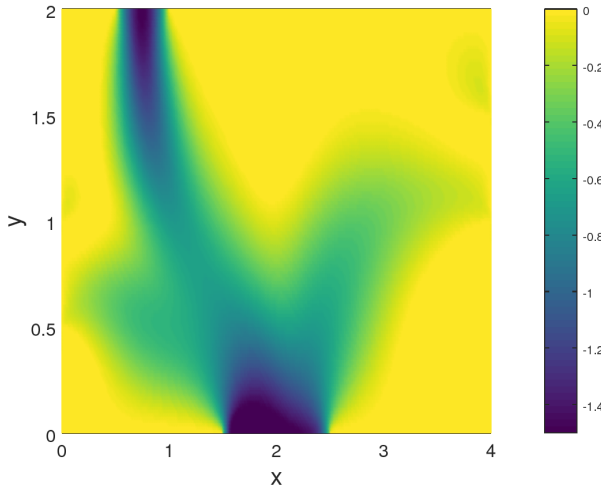

Figure 10: Solution for v at $t = 5.0$

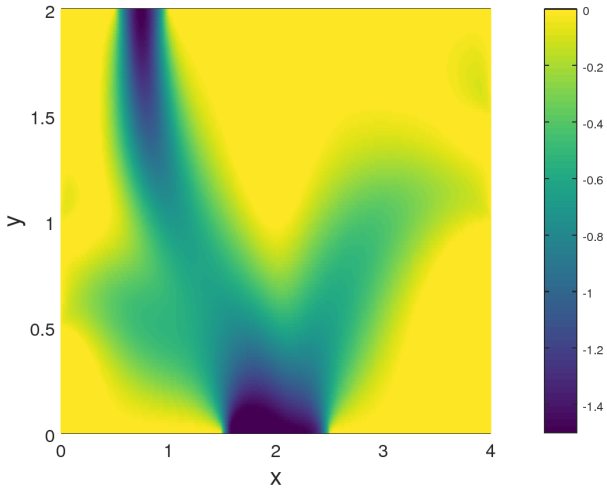

Figure 11: Solution for v at $t = 10.0$

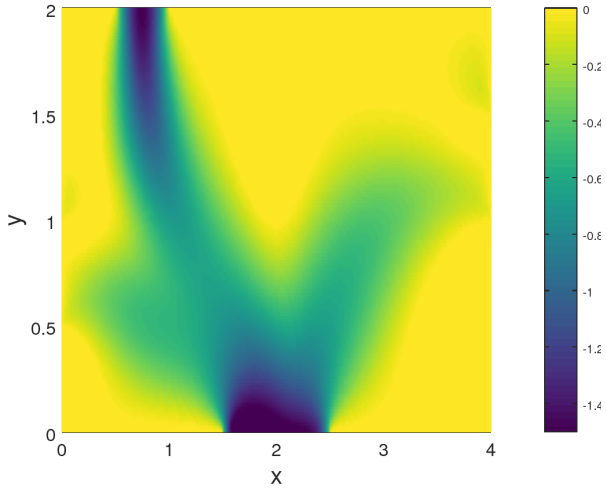

Figure 12: Solution for v at $t = 15.0$

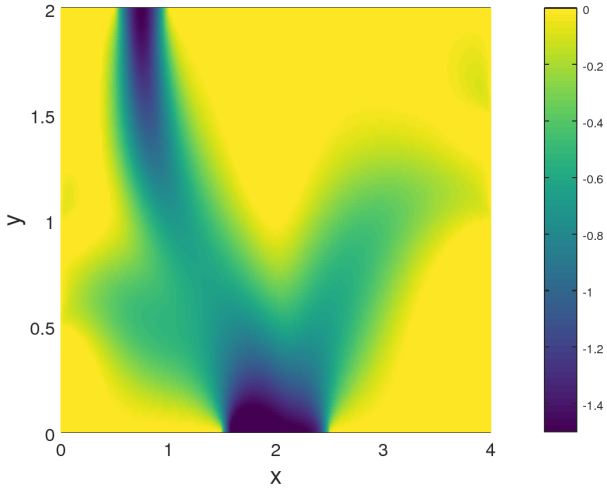

Figure 13: Solution for v at $t = 20.0$

3.3 Contour plots for Y

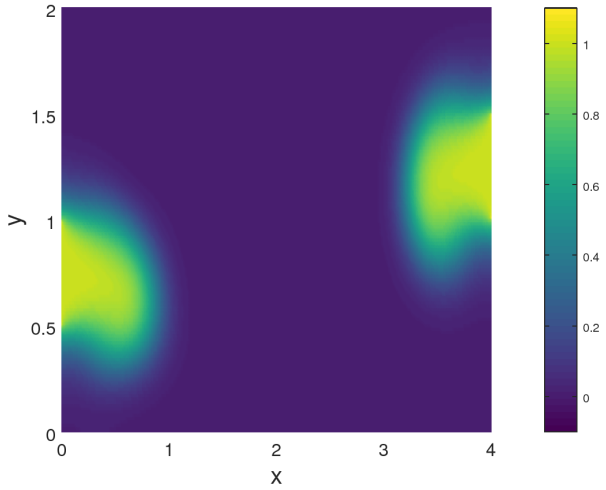

Figure 14: Solution for Y at $t = 1.0$

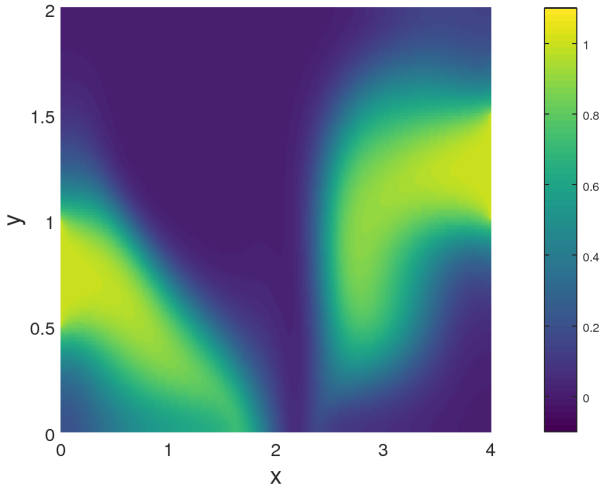

Figure 15: Solution for Y at $t = 3.0$

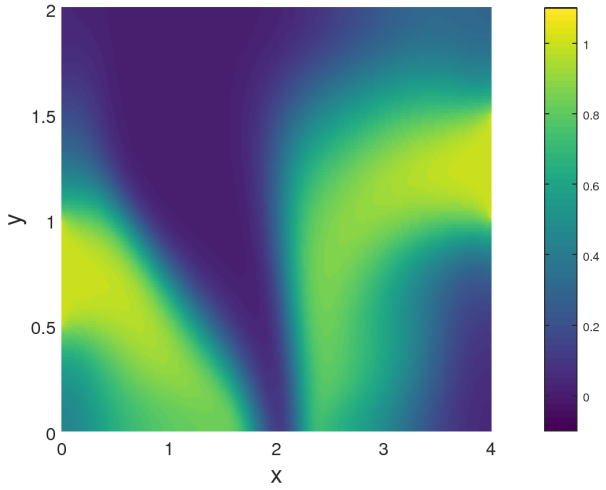

Figure 16: Solution for Y at $t = 5.0$

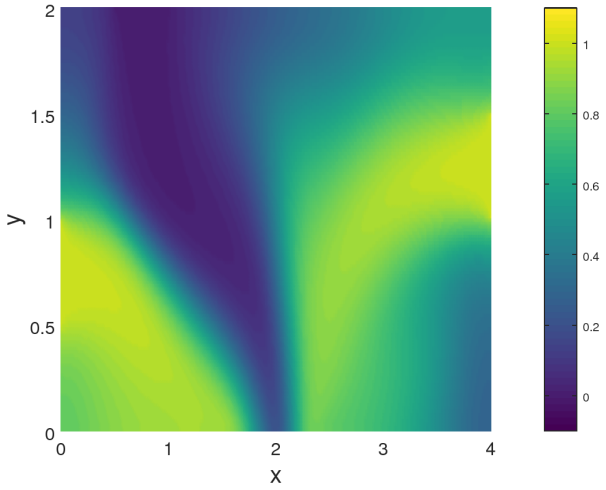

Figure 17: Solution for Y at $t = 10.0$

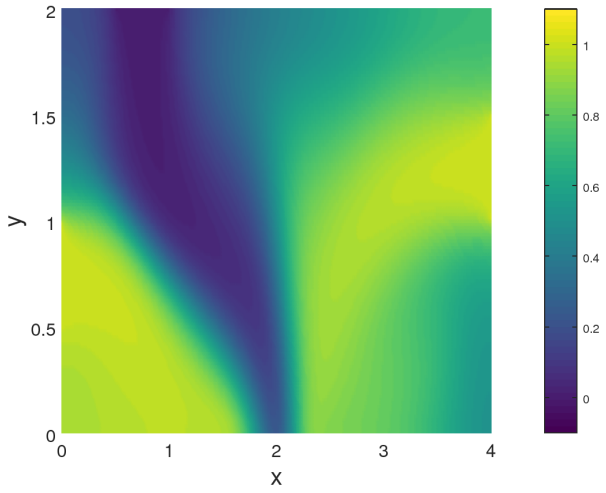

Figure 18: Solution for Y at $t = 15.0$

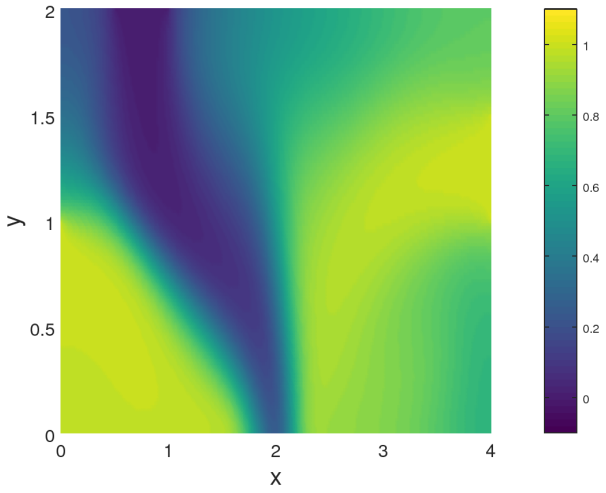

Figure 19: Solution for Y at $t = 20.0$

3.4 Contour plots for $Y(1 - Y)$

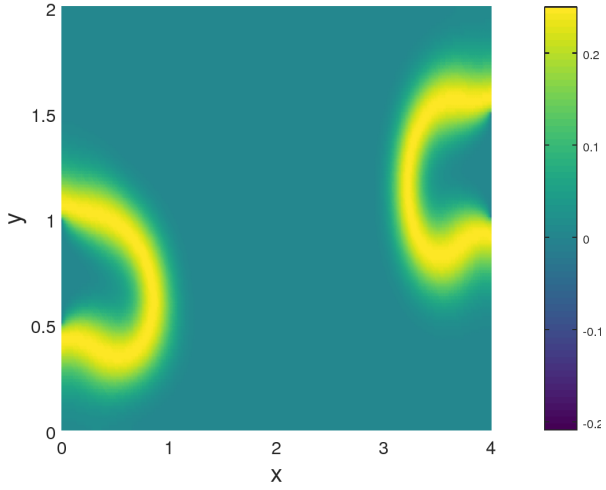

Figure 20: Solution for $Y(1 - Y)$ at $t = 1.0$

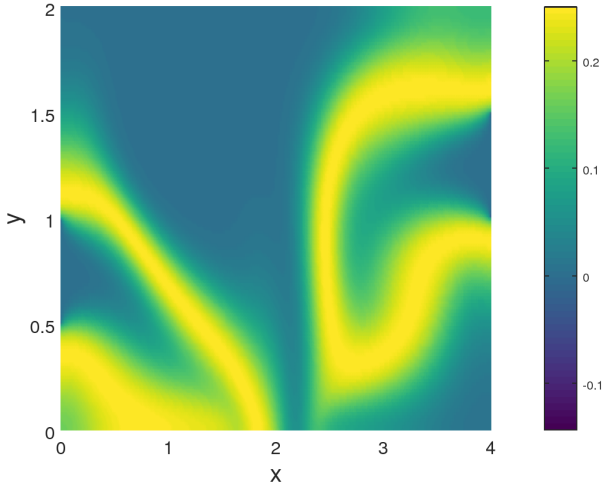

Figure 21: Solution for $Y(1 - Y)$ at $t = 3.0$

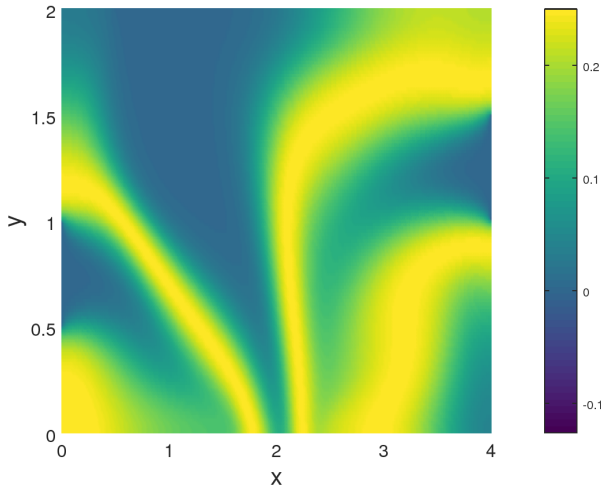

Figure 22: Solution for $Y(1-Y)$ at $t = 5.0$

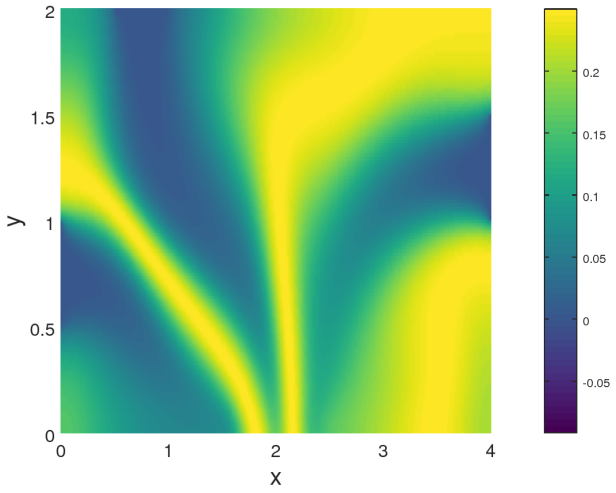

Figure 23: Solution for $Y(1-Y)$ at $t = 10.0$

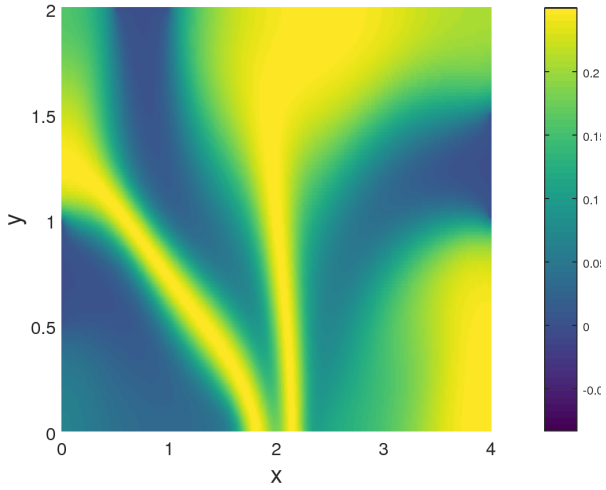

Figure 24: Solution for $Y(1-Y)$ at $t = 15.0$

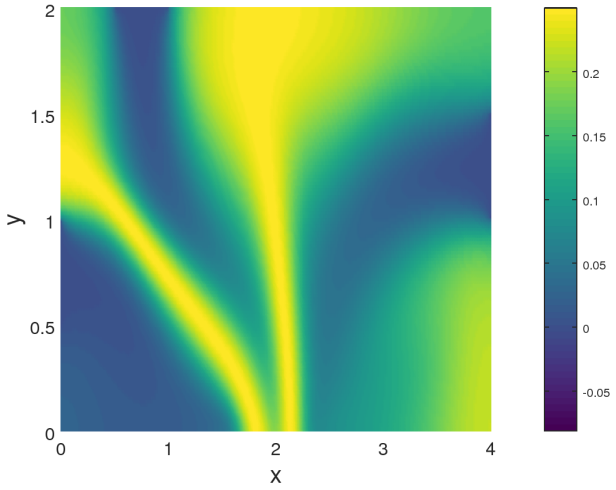

Figure 25: Solution for $Y(1-Y)$ at $t = 20.0$

3.5 Time evolution of $R(t)$ and $S(t)$

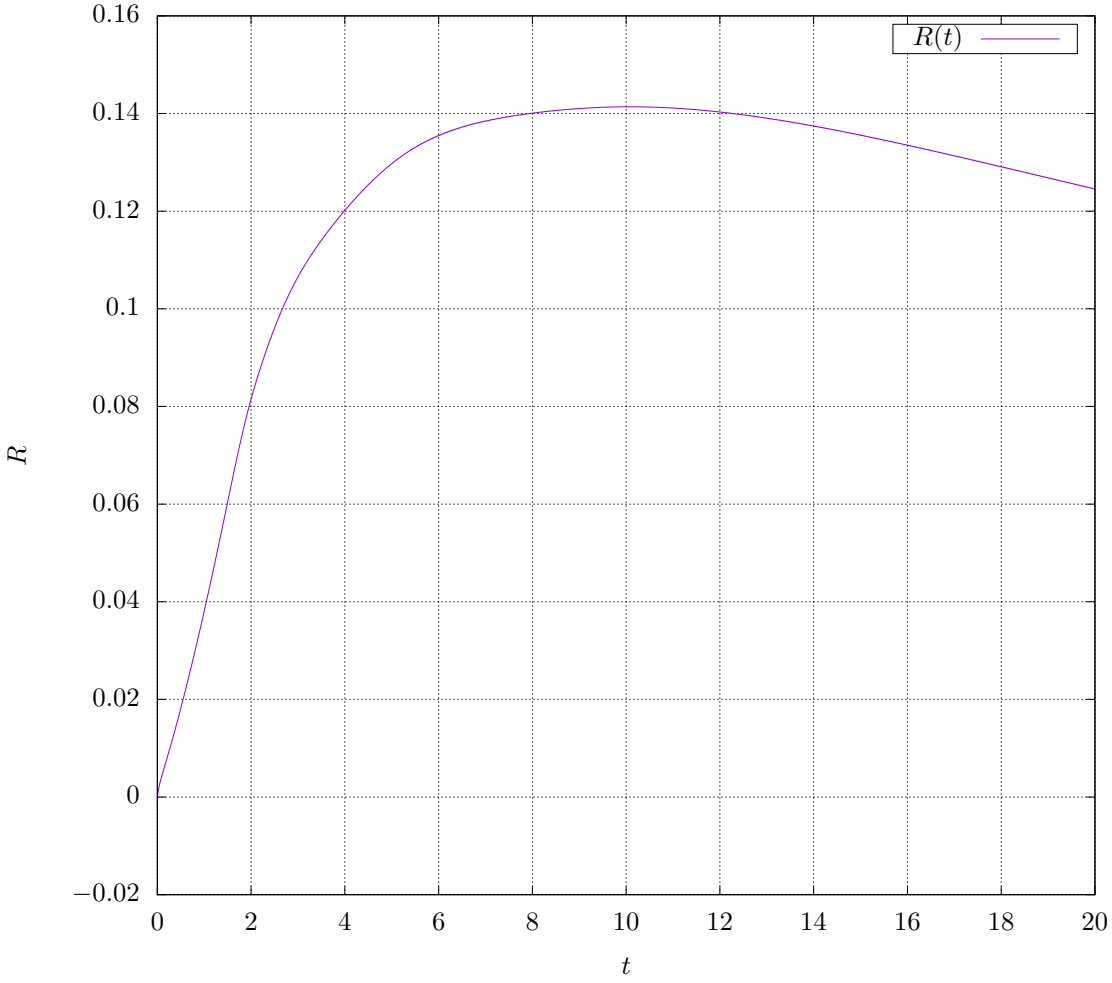

Figure 26: $R(t)$

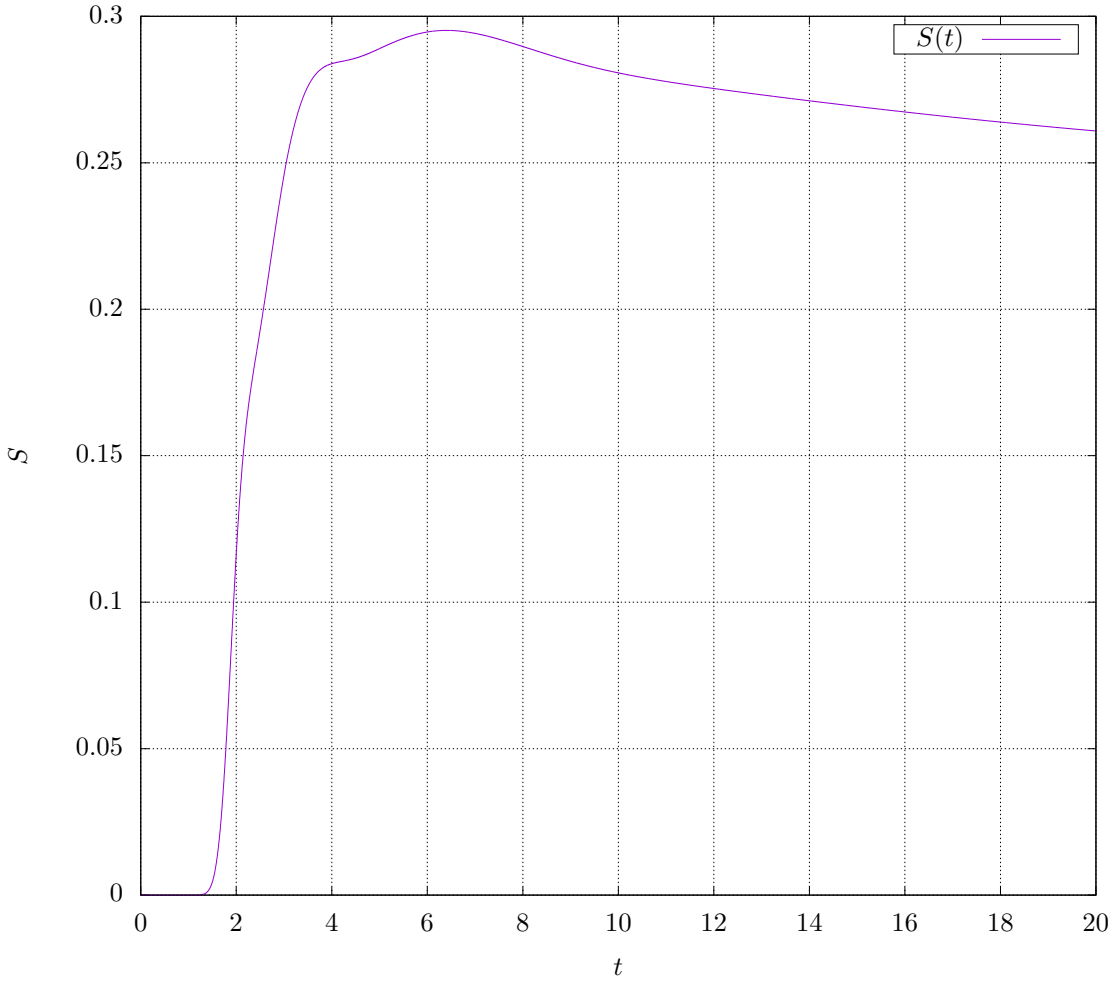

Figure 27: $S(t)$

3.6 Solution verifcation for T

Determine the quantity T to within an acceptable error range.

Table 3: GCI Analysis

Grid	Cells: M	Cells: N	T
1	512	256	0.269233
2	256	128	0.269711
3	128	64	0.271708

Calculate the observed order of convergence for T.

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

$$p = 2.061$$

Determine estimate for exact solution using Richardson extrapolation.

$$f_{h=0} \approx f_1 + \frac{f_1 - f_2}{r^p - 1}$$

$$T_{h=0} \approx 0.269082$$

Calculate GCI's using $F_{sec} = 1.25$.

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

$$GCI_{12} = 0.07\%$$

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

$$GCI_{23} = 0.3\%$$

Confirm asymptotic range of convergence is close to 1.

$$\frac{GCI_{12}}{GCI_{23}} r^p \approx 1$$

$$\text{range}_T = 1.002$$

The asymptotic range of convergence is within an acceptable range near 1 for T. The final solution for T is

$$T = 0.269082 \pm 0.07\%.$$

The final solutions for T is within 0.07% of the true solution which is an acceptable range for the purpose of this project.

4 Problem 2:

The mixing chamber will now be optimized to attain a value of $T \geq 0.35$ so as to produce more evenly mixed reactant at the outlet. The problem will be solved with the same numerical methods as discussed in the prior section, but the location and average velocity of the inlets will be adjusted to produce a more evenly mixed solution. The problem will be optimized under the following constraints:

1. The chamber size is fixed to 4×2
2. The chamber is filled with fluid at rest and $Y = 0$
3. Only one outlet of length $L_{out} = 1$ can be placed on any side of the chamber.
4. No more than 6 fixed inlets can be placed in arbitrary positions, with a total combined length not exceeding 1.5.
5. An inlet can only issue either $Y = 1$ or $Y = 0$.

6. All inlets have normal direction parabolic velocity profiles with a maximum average velocity of 2 and zero tangential velocity.

7. Any inlet may be turned completely off.

8. At any moment in time, the volumetric flow rate $\int -\vec{u} \cdot \vec{n} dx$ of $Y = 1$ through all inlets cannot exceed 1.

9. At any moment in time, the volumetric flow rate $\int -\vec{u} \cdot \vec{n} dx$ of $Y = 0$ through all inlets cannot exceed 1.

10. An inlet cannot change velocity and/or Y more often than once every one time unit.

4.1 Iteration 1

For the first iteration of the design, 3 inlets were added to the top wall of the chamber, 1 on the left and 1 on the right, all with inlet velocities of 1 and inlet lengths of 0.3. The design of the chamber is shown in Fig. 28

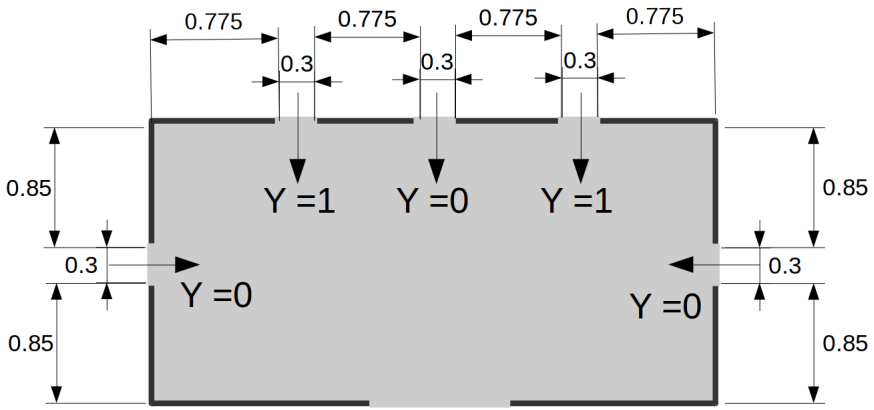

Figure 28: First Iteration of the Optimized Chamber

The $v-$velocity and $Y(1-Y)$ contour plots are shown in Figs. 29 & 30.

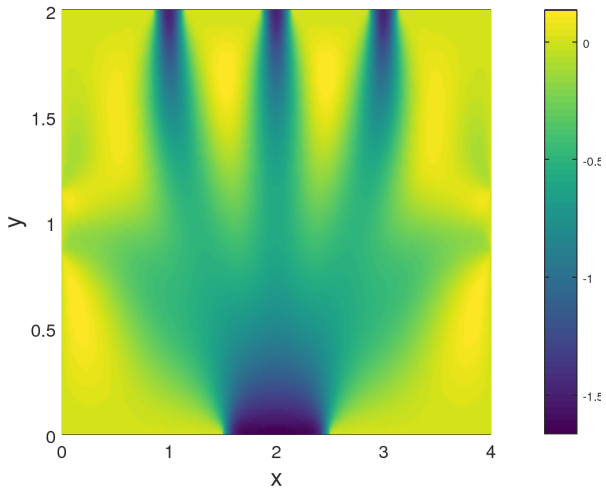

Figure 29: $v-$velocity at $t = 20$, first iteration.

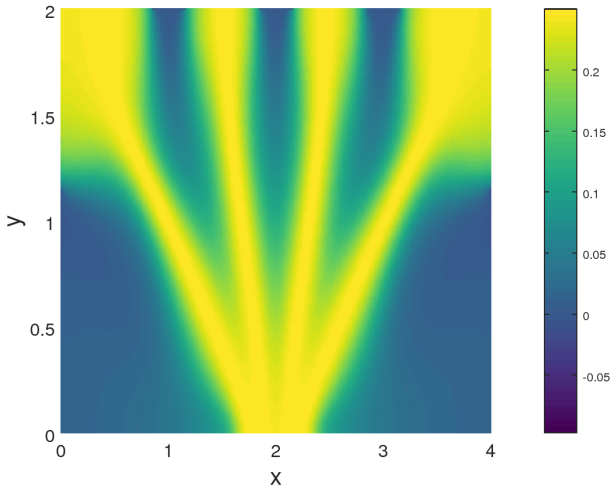

Figure 30: $Y(1-Y)$ at $t = 20$, first iteration.

Unfortunately this design did not meet the requirement of $T \geq 0.35$, so the design was altered to increase the mixing capabilities. The flaw in this design is that there are 3 inlets with $Y = 0$ fluid entering and only 2 inlets of $Y = 1$ entering as shown in Fig. 31.

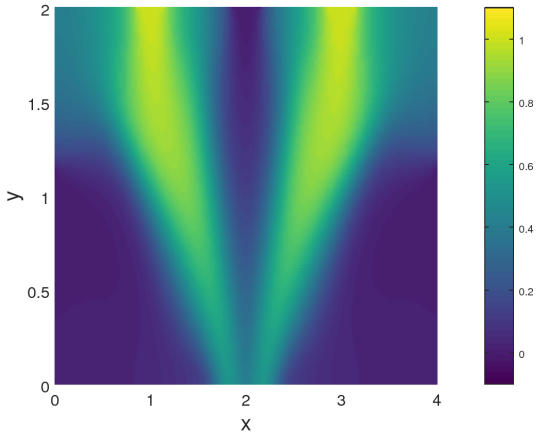

Figure 31: Y at $t = 20$, first iteration.

This causes there to be more fluid at $Y = 0$ going in the chamber at any given time, so the top center inlet was removed, the remaining inlets on the top were moved outwards toward the right and left walls and the inlets on the left and right walls were moved up towards the top wall. See Fig. 32 for an illustration of the chamber.

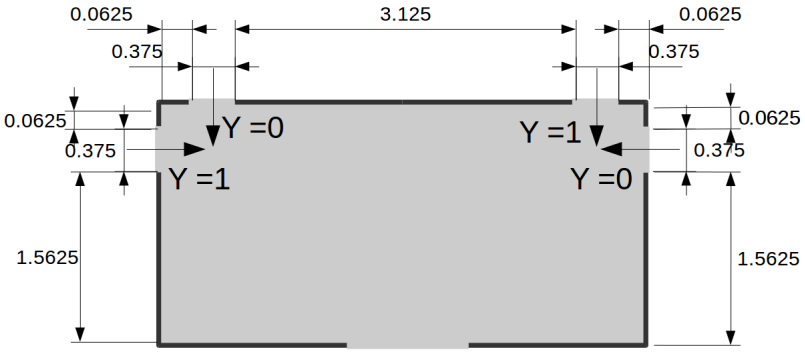

Figure 32: Final Design of Optimized Chamber

This design puts the inlets of the fluid with $Y = 0$ and $Y = 1$ right next to each other and facing perpendicular to each other. The idea behind this design is to mix the fluids right as they enter the chamber so that they have time to mix well before they leave the chamber. The parameters to solve this problem are shown in table 4.

Table 4: Parameters Used in Solution Calculation

Parameter	Value
M	256
N	128
Δx	0.015625
Δy	0.015625
CFL	0.8

The average velocity at the inlets is kept constant at 1.3, so there should be no problem violating constraints 8 & 9, but to validate that this constraint is met, the average velocities at each inlet are plotted as functions of time in Figs. 33-36. These figures show that the velocity and concentration at each inlet is kept constant throughout the duration of the simulation.

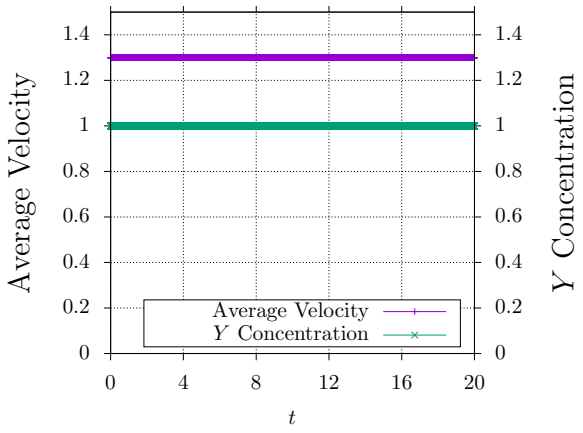

Figure 33: Inlet 1

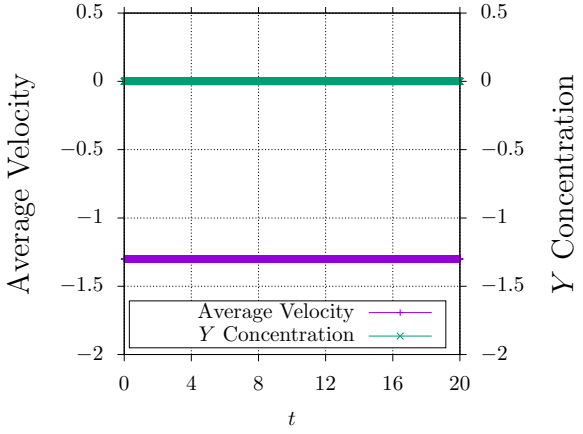

Figure 34: Inlet 2

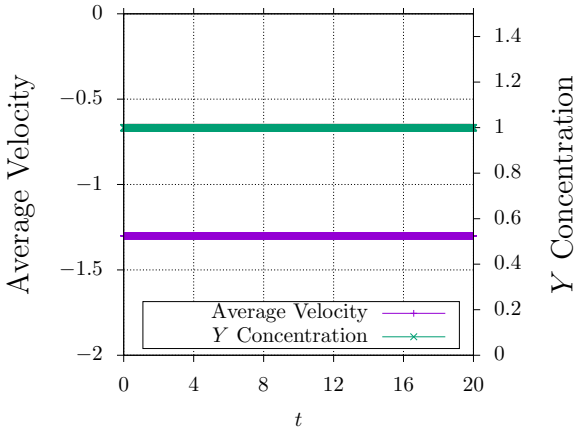

Figure 35: Inlet 3

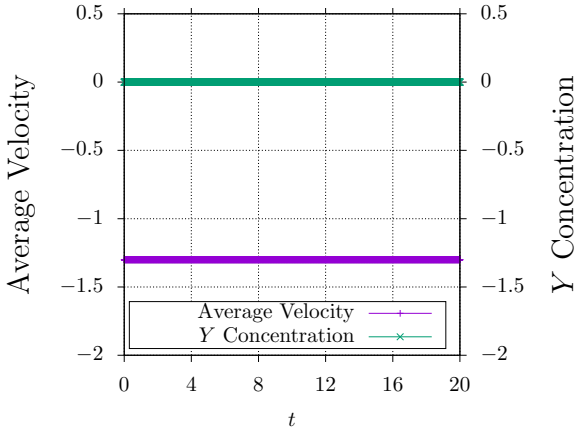

Figure 36: Inlet 4

To show that the volumetric flow rate coming in the chamber does not exceed 1, Fig. 37 shows the volumetric flow rate into the chamber across the whole time domain $0 \leq t \leq 20$.

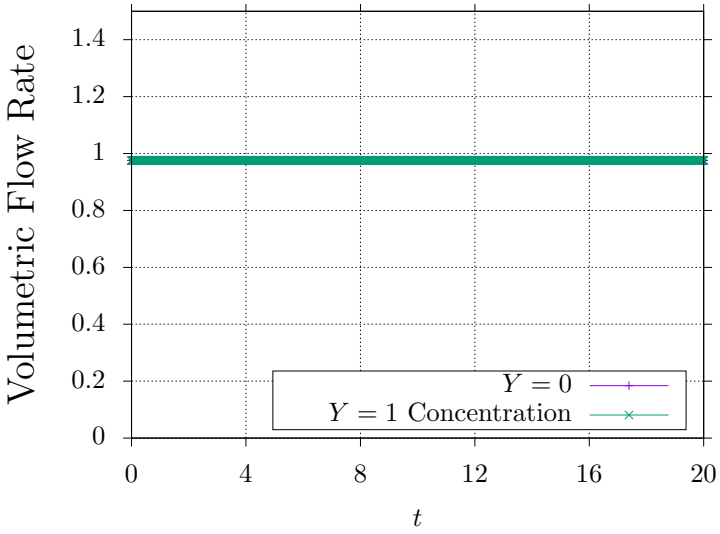

Figure 37: Volume flow into chamber

Note that the volumetric flow rate of the inlets with $Y = 1$ is equal to the volumetric flow rate of the inlets with $Y = 0$, so the lines in Fig. 37 are directly on top of each other. This is intentional in the design because the goal is to mix the chemcial species as evenly as possible.

4.2 Contour plots for u

The solutions of u, v, Y and $Y(1-Y)$ at $t = 10, 12, 14, 16, 18, 20$ are shown in Figs. 38-61.

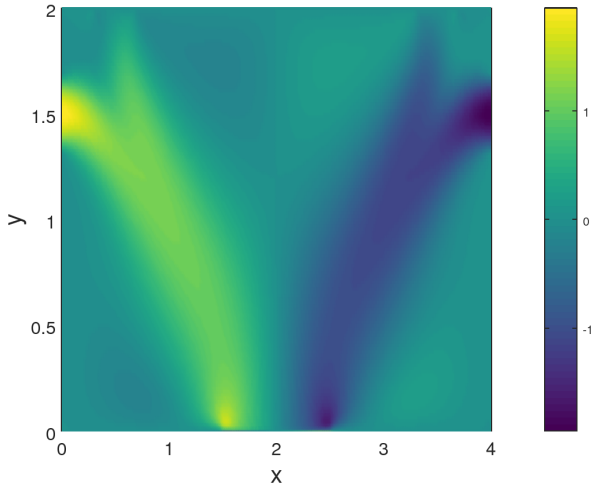

Figure 38: Solution for u at $t = 10.0$

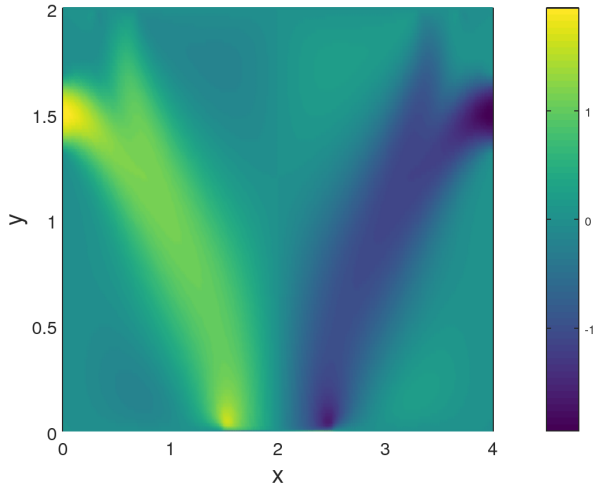

Figure 39: Solution for u at $t = 1.0$

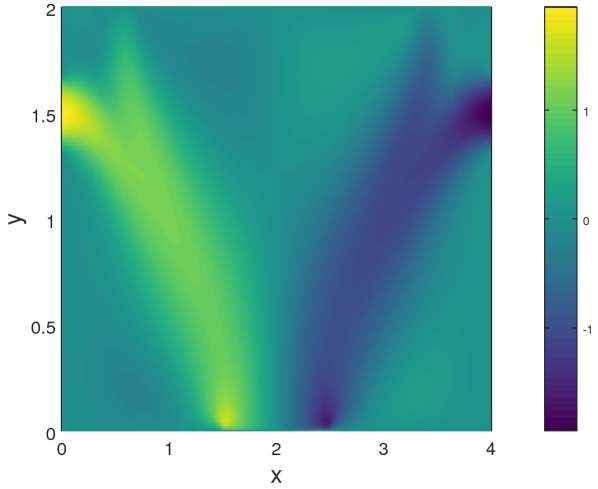

Figure 40: Solution for u at $t = 14.0$

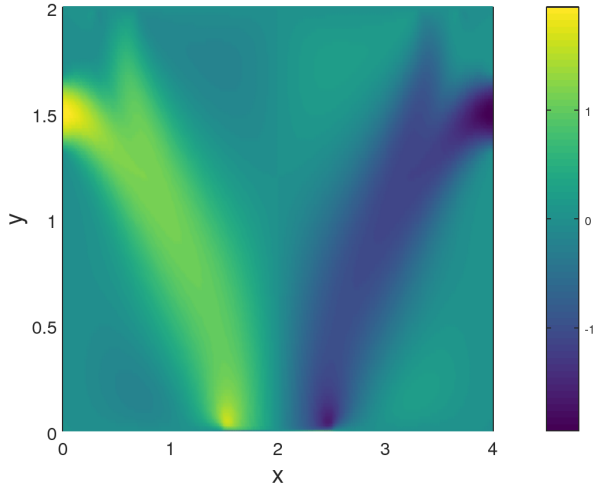

Figure 41: Solution for u at $t = 16.0$

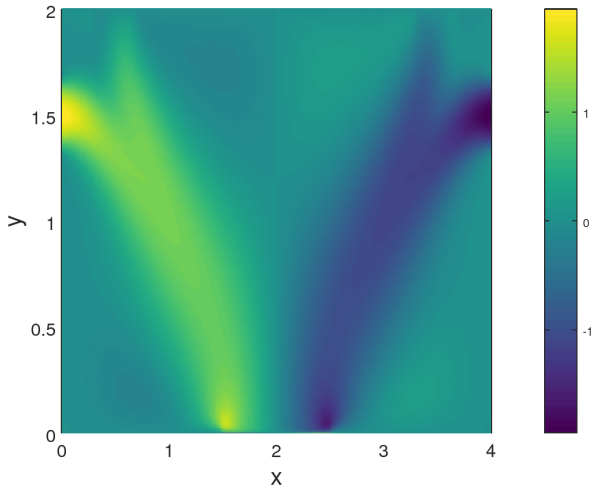

Figure 42: Solution for u at $t = 18.0$

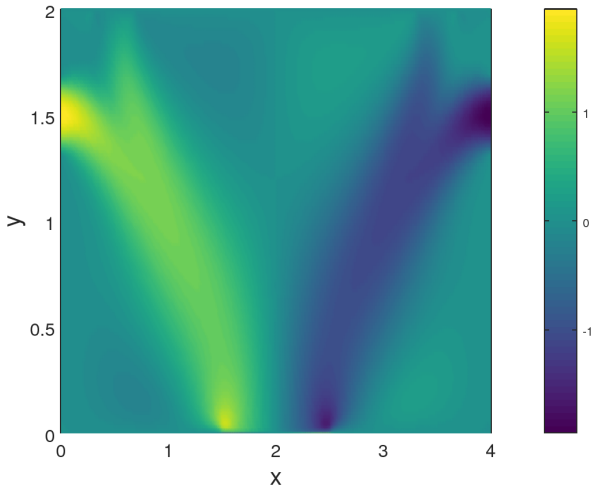

Figure 43: Solution for u at $t = 20.0$

4.3 Contour plots for v

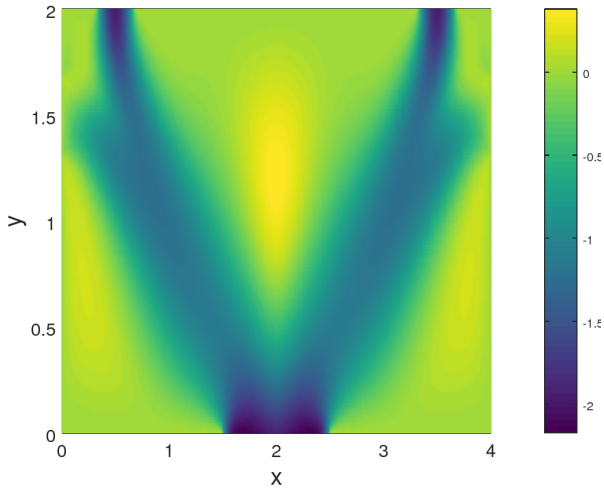

Figure 44: Solution for v at $t = 10.0$

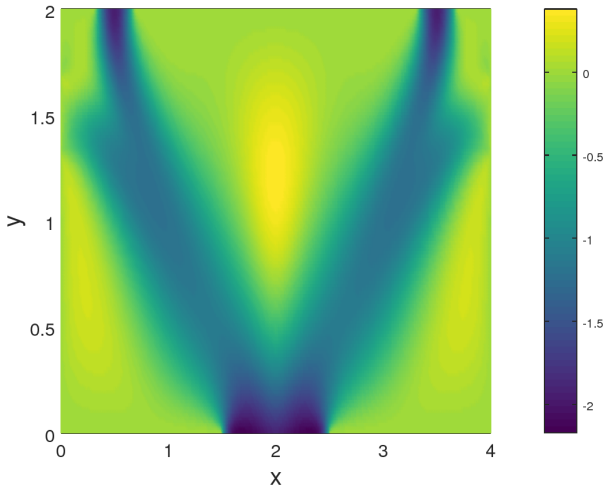

Figure 45: Solution for v at $t = 12.0$

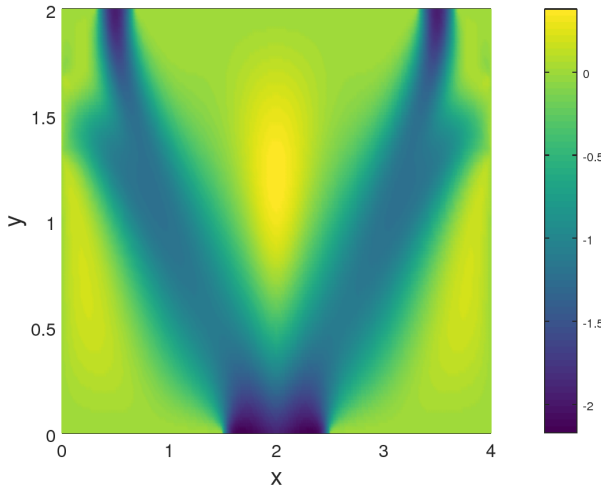

Figure 46: Solution for v at $t = 14.0$

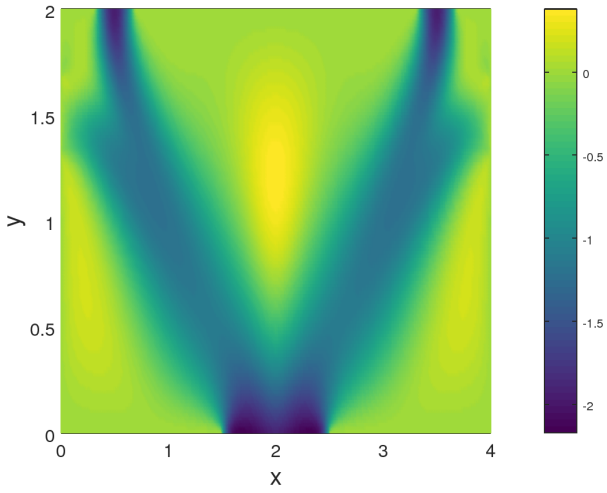

Figure 47: Solution for v at $t = 16.0$

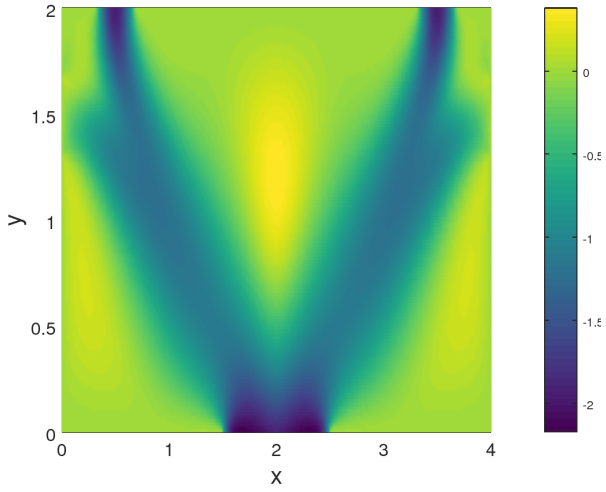

Figure 48: Solution for v at $t = 18.0$

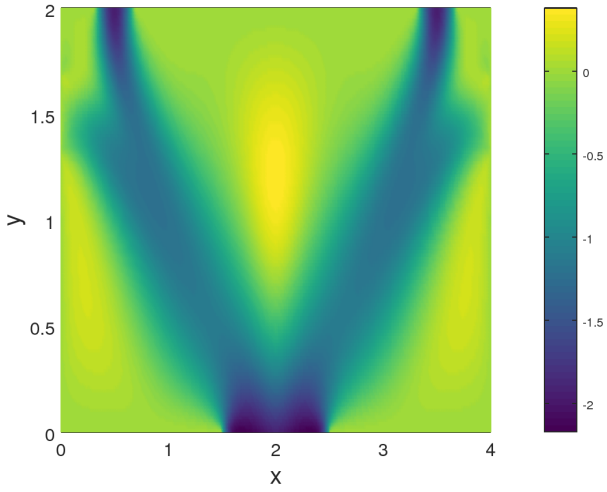

Figure 49: Solution for v at $t = 20.0$

4.4 Contour plots for Y

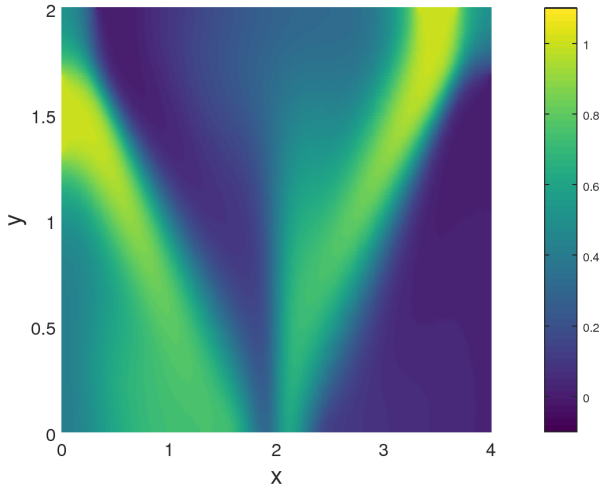

Figure 50: Solution for Y at $t = 10.0$

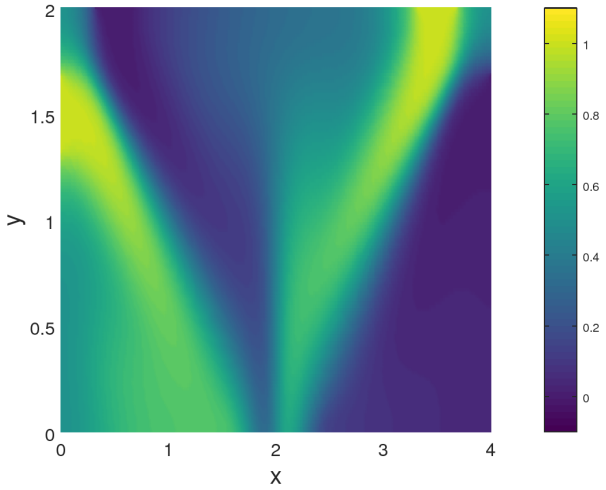

Figure 51: Solution for Y at $t = 12.0$

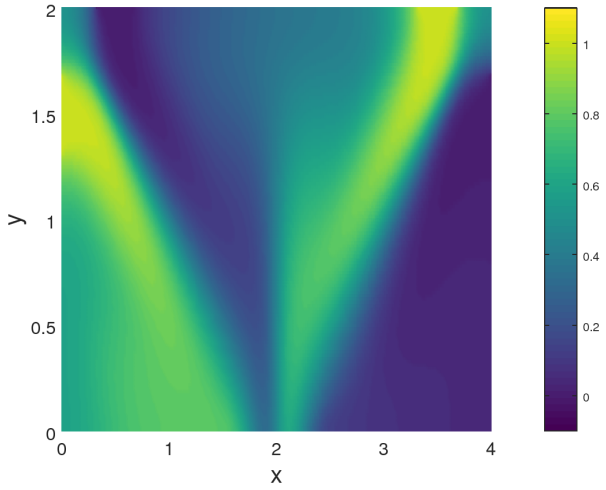

Figure 52: Solution for Y at $t = 14.0$

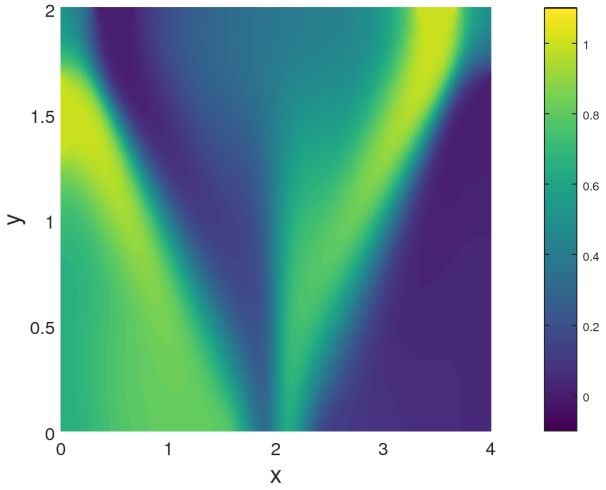

Figure 53: Solution for Y at $t = 16.0$

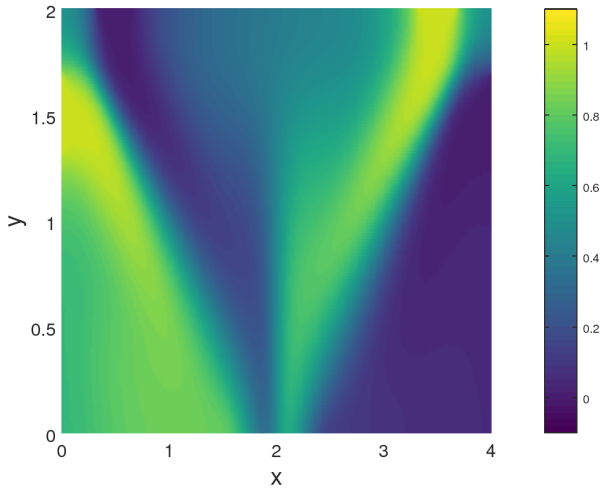

Figure 54: Solution for Y at $t = 18.0$

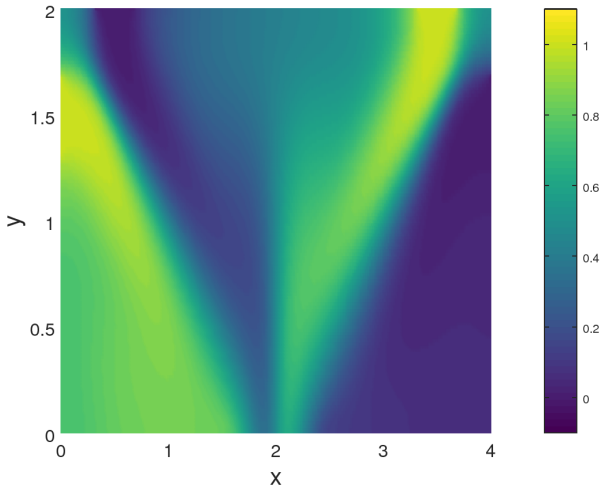

Figure 55: Solution for Y at $t = 20.0$

4.5 Contour plots for $Y(1 - Y)$

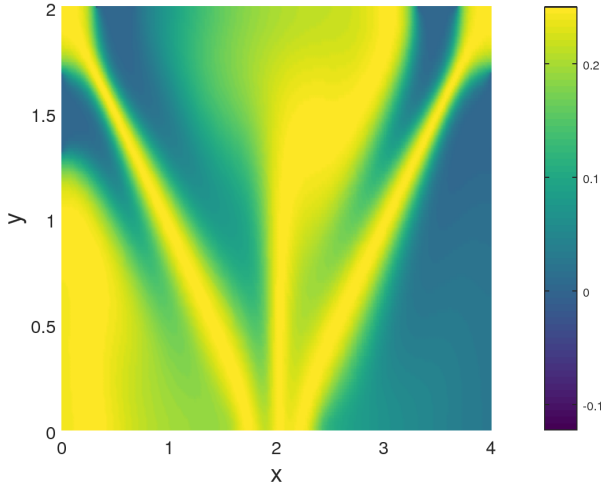

Figure 56: Solution for $Y(1 - Y)$ at $t = 10.0$

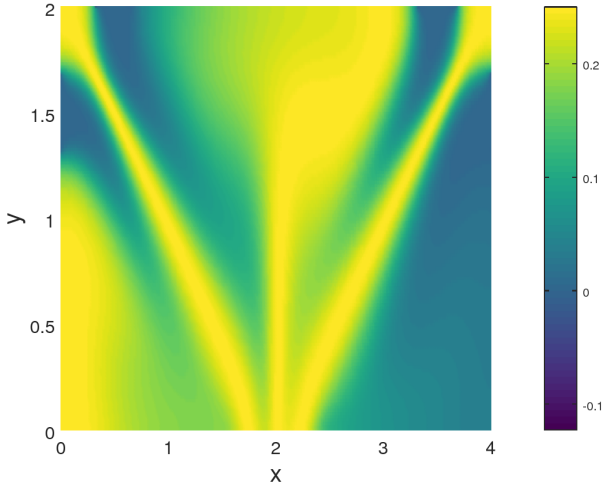

Figure 57: Solution for $Y(1 - Y)$ at $t = 12.0$

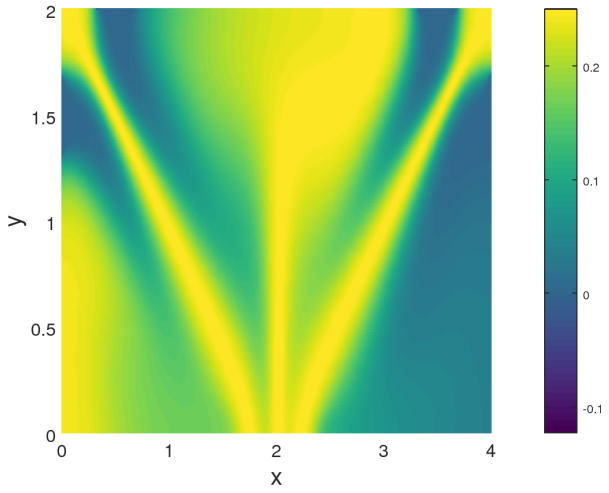

Figure 58: Solution for $Y(1-Y)$ at $t = 14.0$

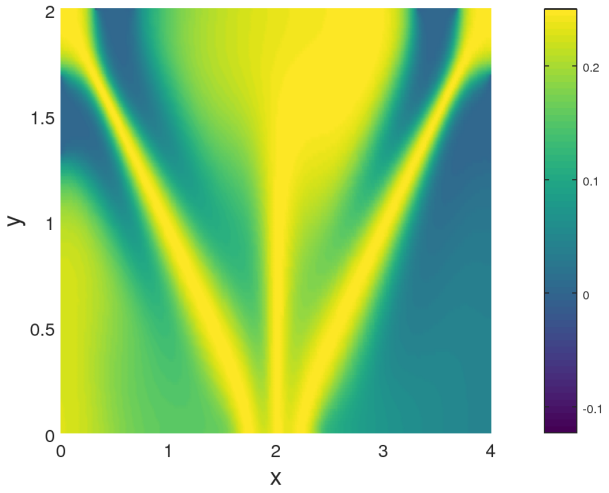

Figure 59: Solution for $Y(1-Y)$ at $t = 16.0$

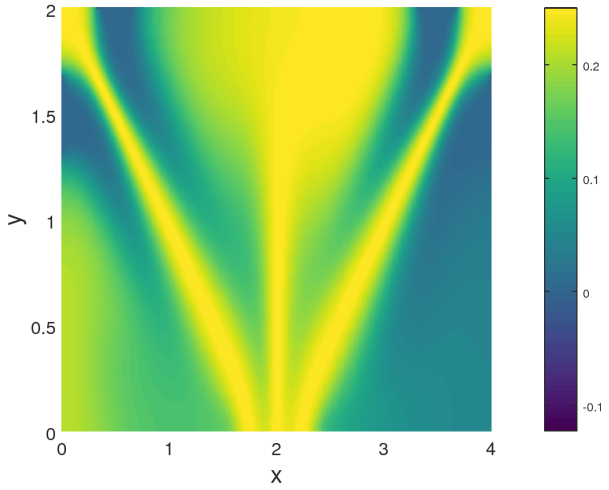

Figure 60: Solution for $Y(1-Y)$ at $t = 18.0$

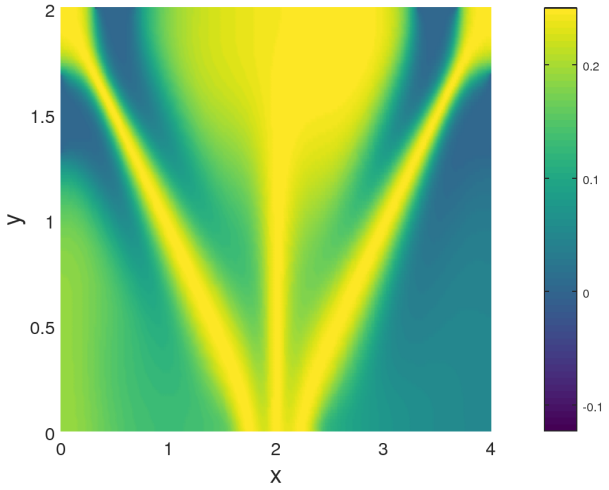

Figure 61: Solution for $Y(1-Y)$ at $t = 20.0$

4.6 Time Evolution of $R(t)$ and $S(t)$

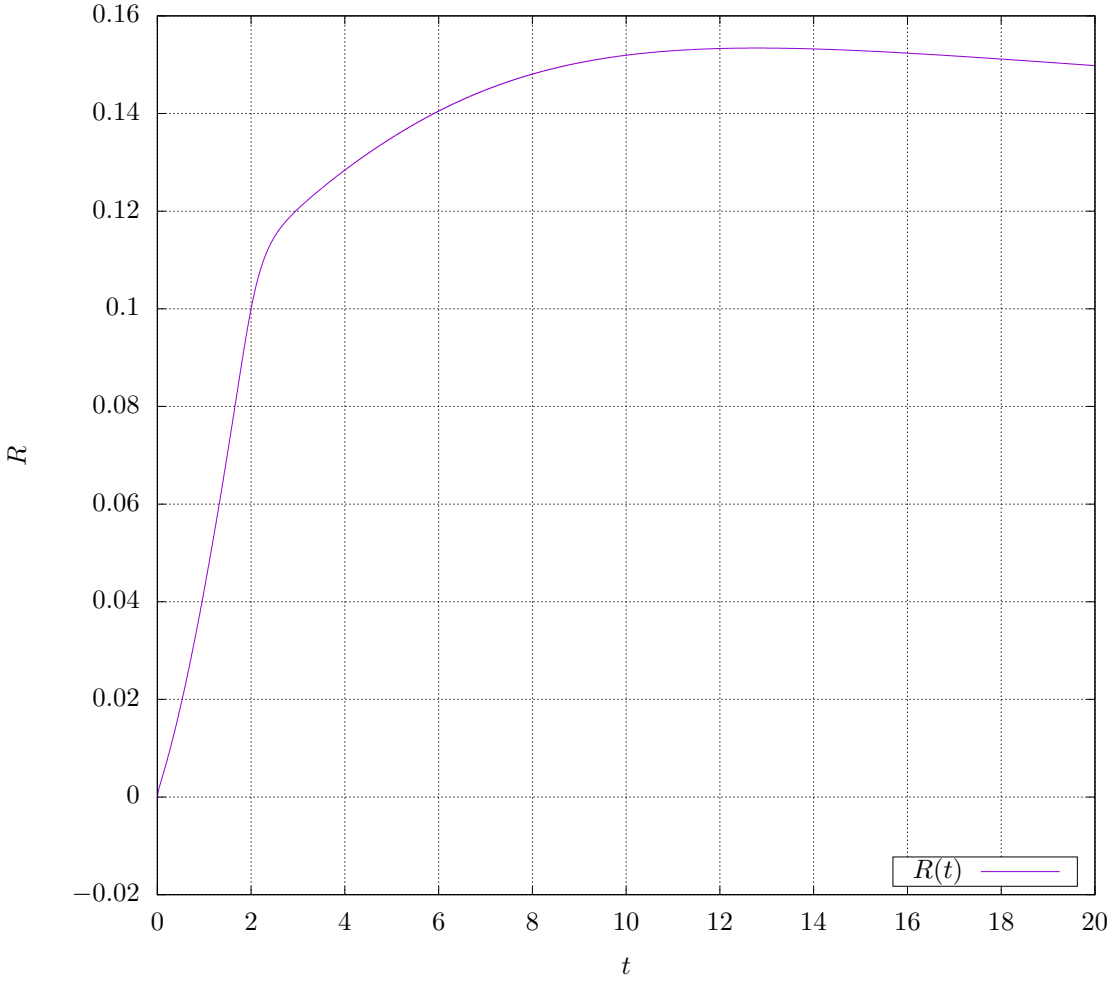

Figure 62: $R(t)$

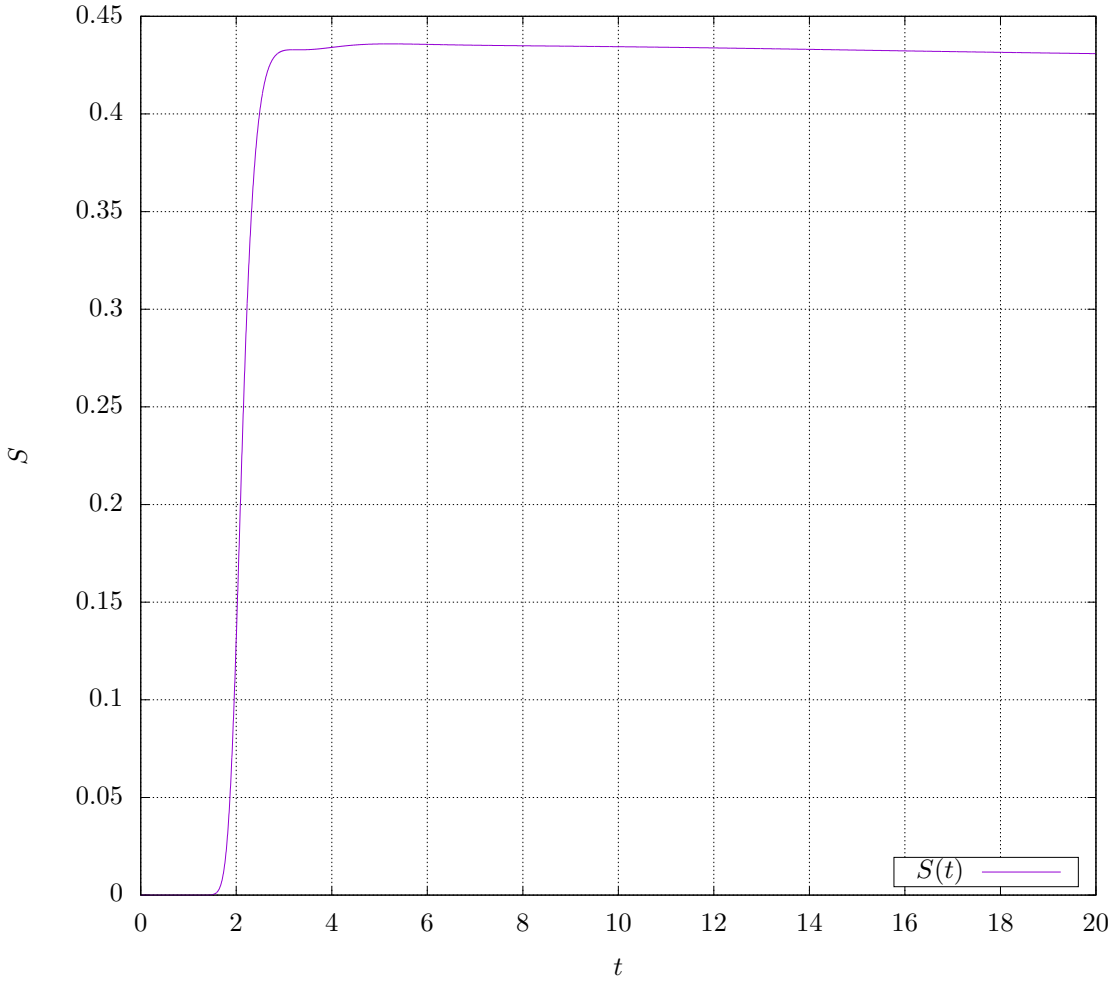

Figure 63: $S(t)$

4.7 Solution verifcation for T

Using the same analysis as shown for problem 1, the quantity T will be determined to within an acceptable error range.

Table 5: GCI Analysis

Grid	Cells: M	Cells: N	T	p	range
1	1024	512	0.434063	0.917	0.9989
2	512	256	0.433588	-4.03	0.998
3	256	128	0.432690	5.25	1.0001
4	128	64	0.432745	4.139	1.005
5	64	32	0.434848	0.545617	0.914
6	32	16	0.397786		
7	16	8	0.451884		

The overall order of convergence of the numerical methods used in this simulation is second order in space and second order in time, so the expected order of convergence for the GCI analysis should be around 2. However, that is not what is observed as shown in table 5. The observed order of convergence actually jumps

around from about 4 and 5 to -4 and then finally to 0.917. There are a number of reasons that could cause the observed order of convergence to not match the formal order of convergence of the methods used. It is possible that the CFL of 0.8 is too high and that time discretization errors are dominating over the spacial errors. To remedy this issue, the study could be repeated with a CFL of 0.4 or smaller to see if the observed order of convergence approaches 2. It is also possible that the grid is not fine enough and even finer meshes are required to reach the formal order of convergence. Both of these options, however, would require more time to complete and due to the time constraints of the final project, it would not be feasible to complete the simulations in time. It is also possible that the outlet conditions are causing an error because of the high velocity gradients they are causing. The current outlet conditions strictly apply a $v-$velocity outflow with no tangential velocity ($u-$velocity), and it is discontinuous at the wall-outlet interface. With the viscous effects activated in the Navier-Stokes equations, this will cause high velocity gradients near the outlet and could lead to a lower observed order of convergence. With this in mind, if we take the value of T calculated at the finest mesh, the estimate for exact solution using Richardson extrapolation can be calculated with the following.

$$f_{h=0} \approx f_1 + \frac{f_1 - f_2}{r^p - 1}$$

$$T_{h=0} \approx 0.434598$$

Calculate GCI's using $F_{sec} = 1.25$.

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

$$GCI_{12} = 0.15\%$$

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

$$GCI_{23} = 0.29\%$$

The final solution for T is

$$T = 0.434598 \pm 0.15\%.$$

The final solutions for T is within 0.15% of the true solution which is an acceptable range for the purpose of this project. Again, since the observed order of convergence did not match exactly with the formal order of convergence, this value should be taken with some skepticism. The GCI analysis does show however, that the asymptotic range of convergence is showing the correct trend with the values approaching 1.

5 Bonus Problem:

Problem 2 will be repeated, but with the Reynolds number changed to $Re = 200$ and the Schmidt number changed to $Sc = 5$. The same design as shown in Fig. 32 is used for an the design of the chamber. Table 6 shows the parameters used in this simulation.

Table 6: Parameters Used in Solution Calculation

Parameter	Value
M	256
N	128
Δx	0.015625
Δy	0.015625
CFL	0.3

The average velocity at the inlets is kept constant at 1.3, so there should be no problem violating constraints 8 & 9, but to validate that this constraint is met, the average velocities at each inlet are again plotted as functions of time in Figs. 64-67. These figures show that the velocity and concentration at each inlet is kept constant throughout the duration of the simulation.

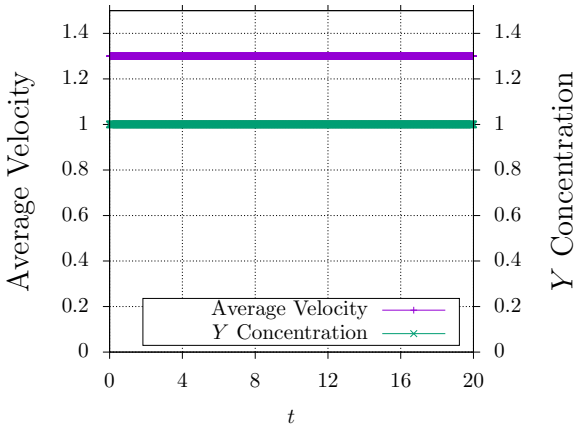

Figure 64: Inlet 1

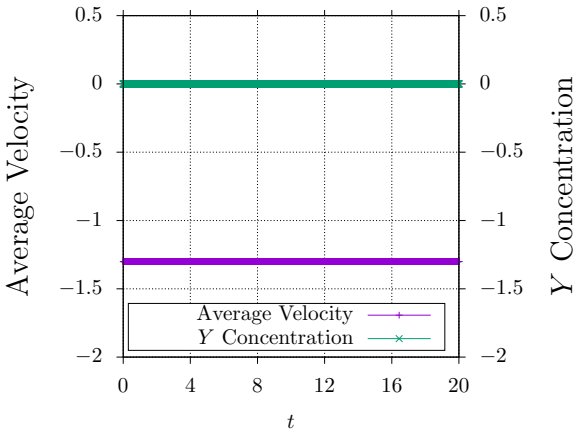

Figure 65: Inlet 2

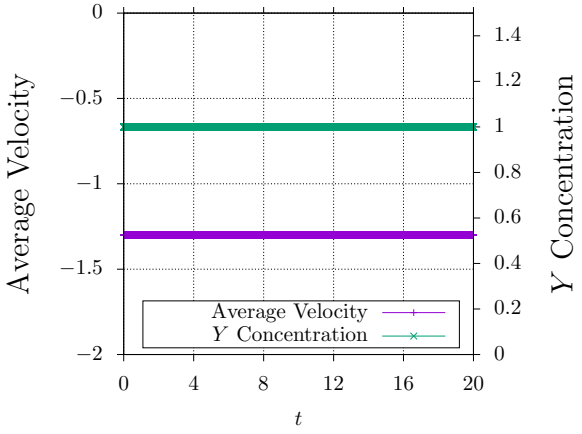

Figure 66: Inlet 3

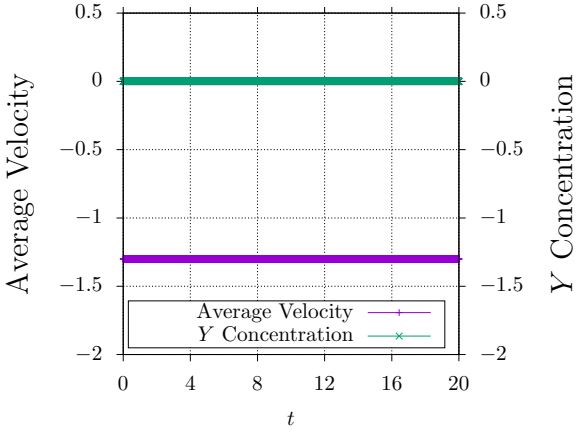

Figure 67: Inlet 4

To show that the volumetric flow rate coming in the chamber does not exceed 1, Fig. 37 shows the volumetric flow rate into the chamber across the whole time domain $0 \leq t \leq 20$.

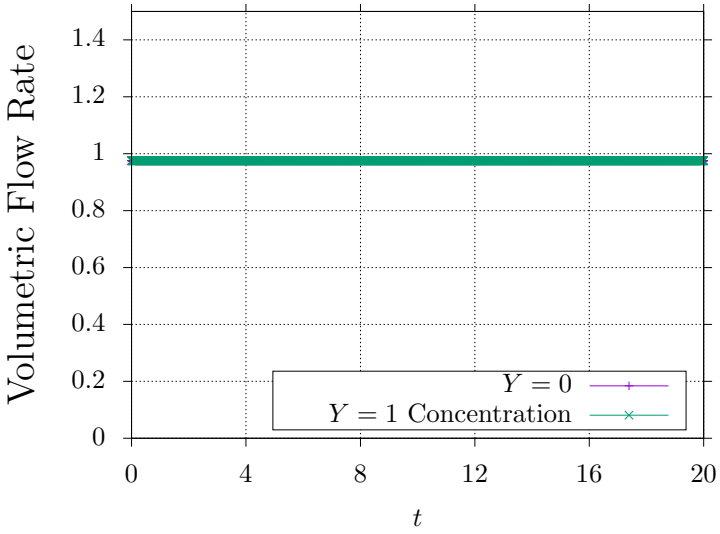

Figure 68: Volume flow into chamber

Note that the volumetric flow rate of the inlets with $Y = 1$ is equal to the volumetric flow rate of the inlets with $Y = 0$, so the lines in Fig. 68 are directly on top of each other. This is intentional in the design because the goal is to mix the chemcial species as evenly as possible. This setup is the same setup as used in the design for problem 2.

5.1 Contour plots for u

The solutions of u, v, Y and $Y(1-Y)$ at $t = 10, 12, 14, 16, 18, 20$ are shown in Figs. 69-92.

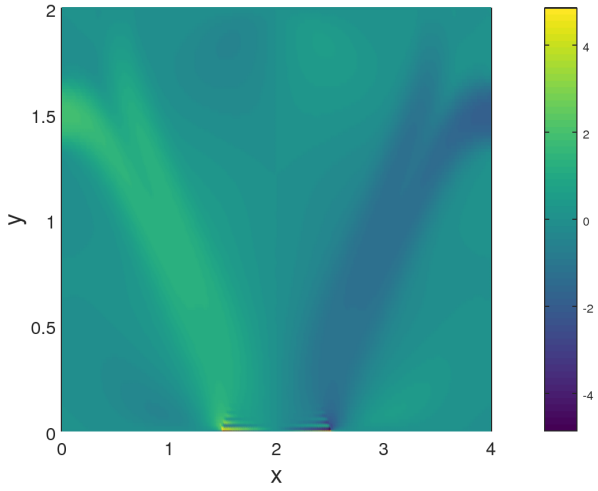

Figure 69: Solution for u at $t = 10.0$

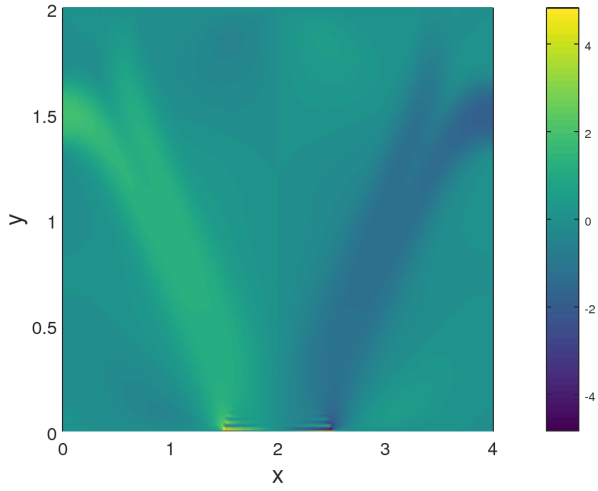

Figure 70: Solution for u at $t = 1.0$

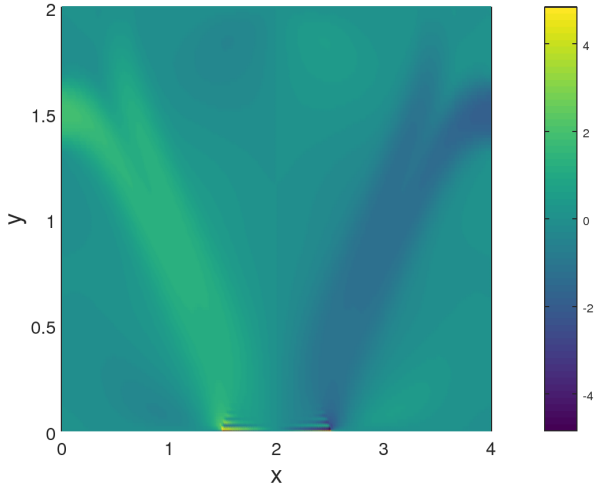

Figure 71: Solution for u at $t = 14.0$

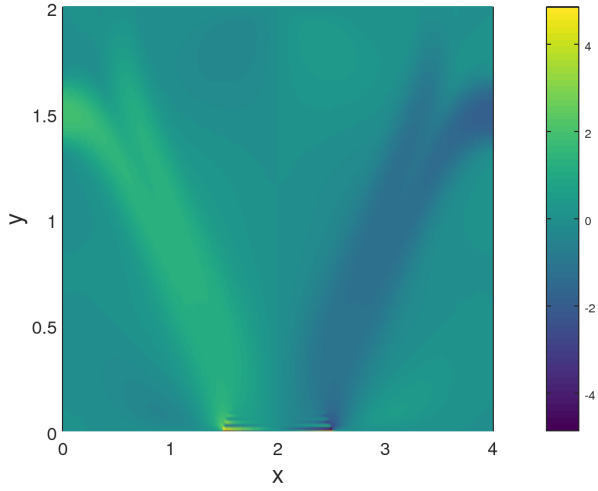

Figure 72: Solution for u at $t = 16.0$

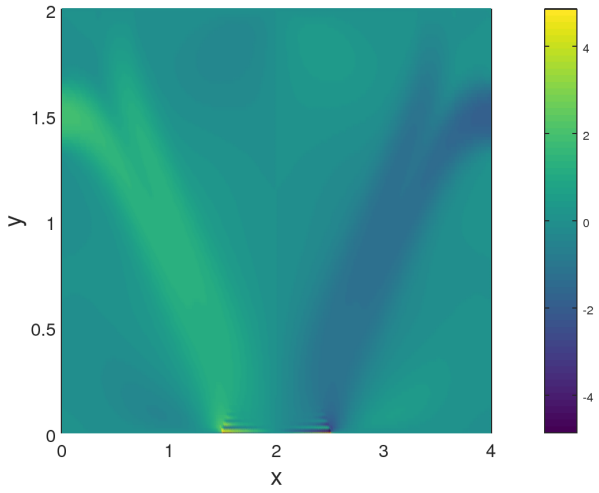

Figure 73: Solution for u at $t = 18.0$

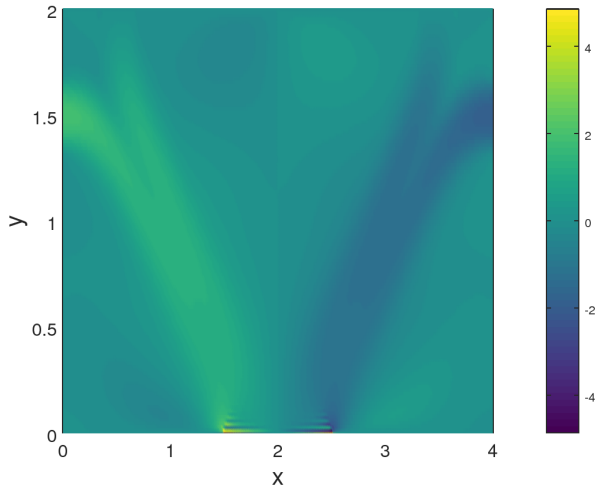

Figure 74: Solution for u at $t = 20.0$

5.2 Contour plots for v

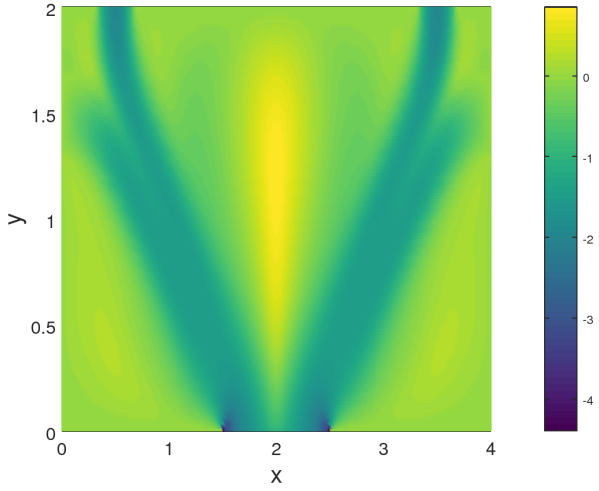

Figure 75: Solution for v at $t = 10.0$

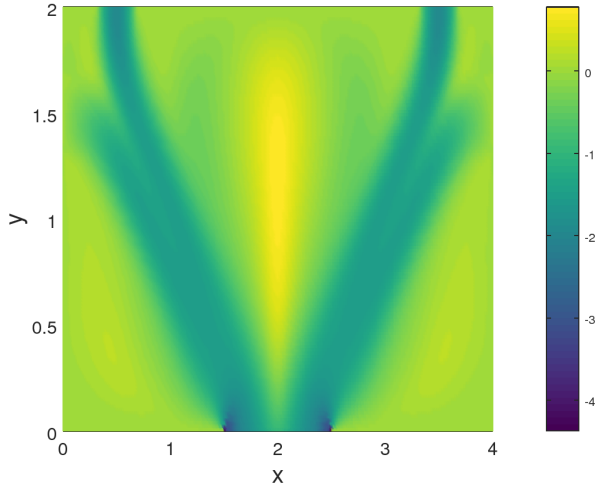

Figure 76: Solution for v at $t = 12.0$

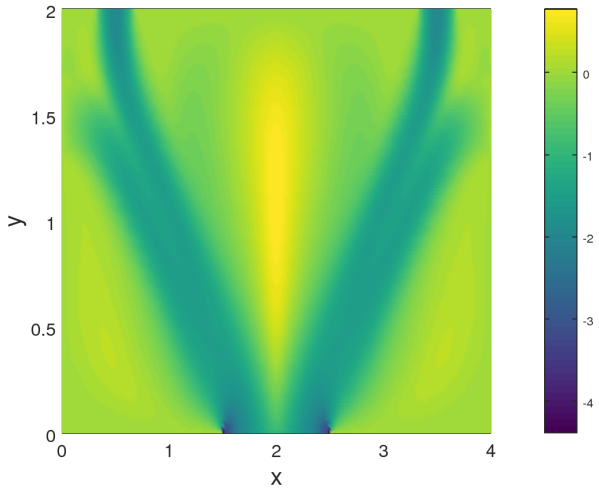

Figure 77: Solution for v at $t = 14.0$

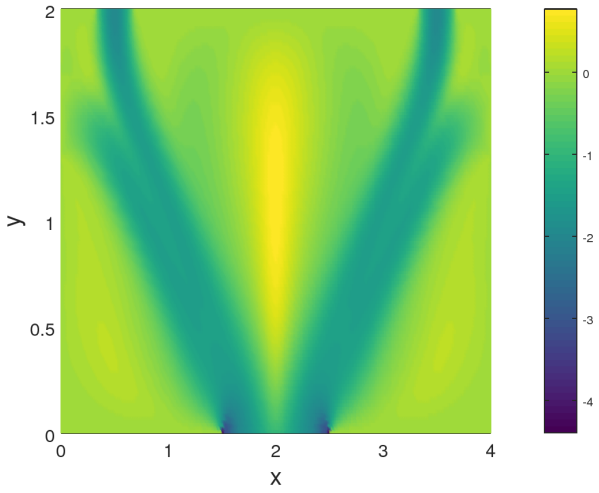

Figure 78: Solution for v at $t = 16.0$

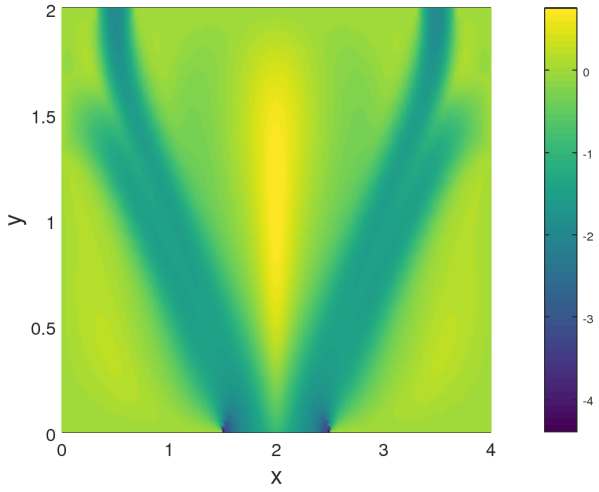

Figure 79: Solution for v at $t = 18.0$

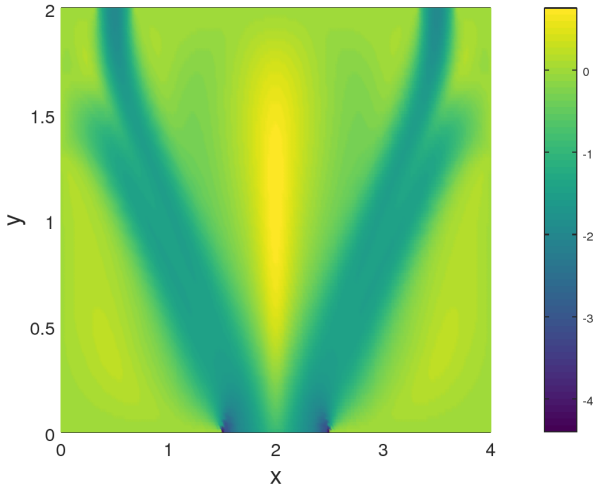

Figure 80: Solution for v at $t = 20.0$

5.3 Contour plots for Y

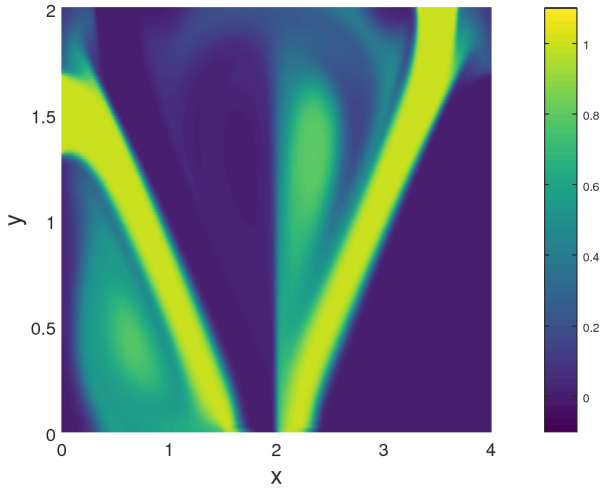

Figure 81: Solution for Y at $t = 10.0$

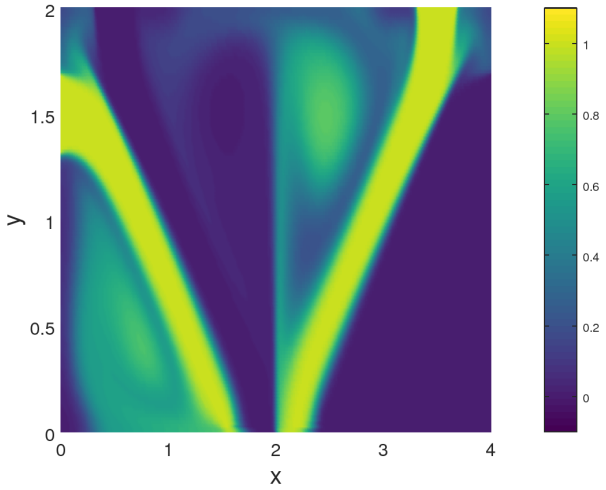

Figure 82: Solution for Y at $t = 12.0$

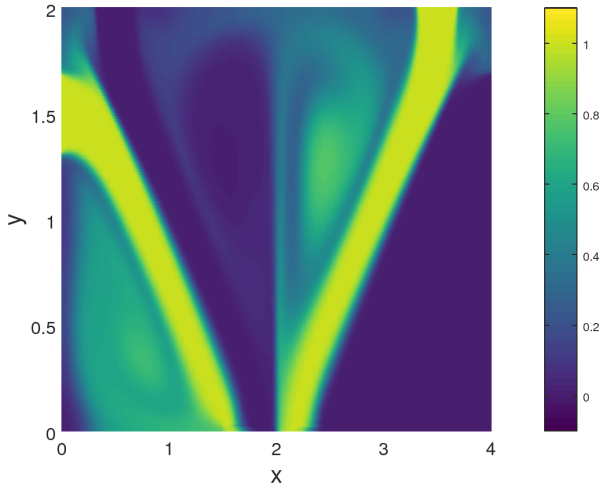

Figure 83: Solution for Y at $t = 14.0$

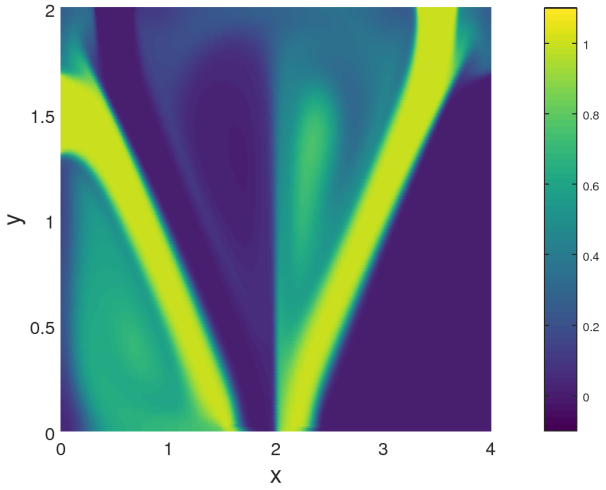

Figure 84: Solution for Y at $t = 16.0$

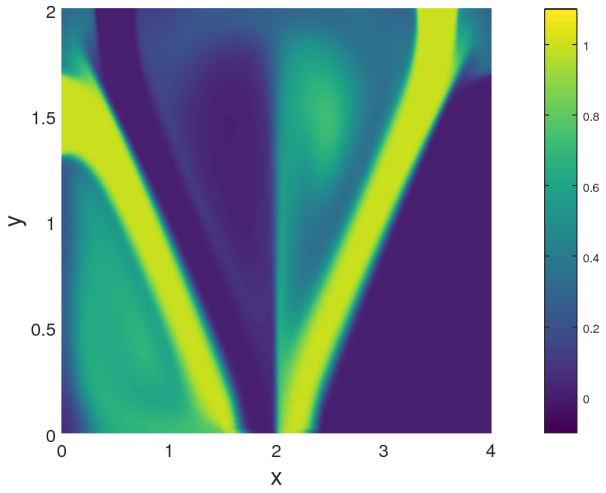

Figure 85: Solution for Y at $t = 18.0$

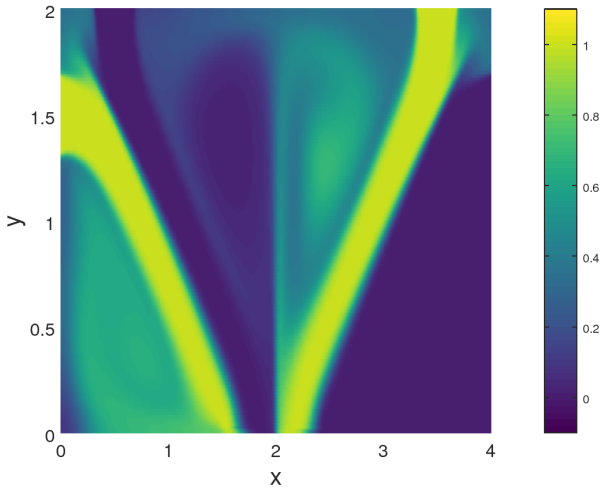

Figure 86: Solution for Y at $t = 20.0$

5.4 Contour plots for $Y(1 - Y)$

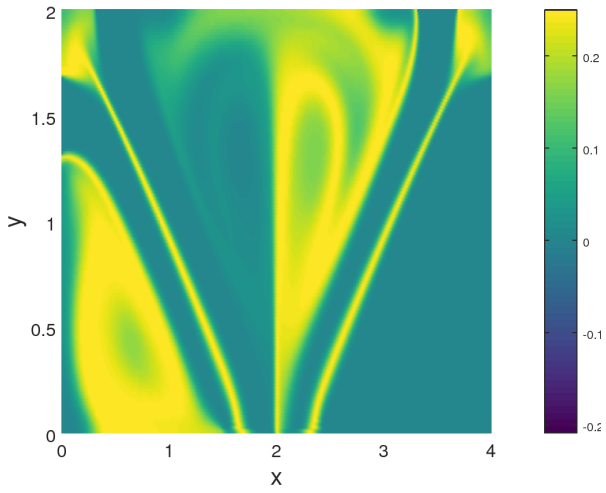

Figure 87: Solution for $Y(1 - Y)$ at $t = 10.0$

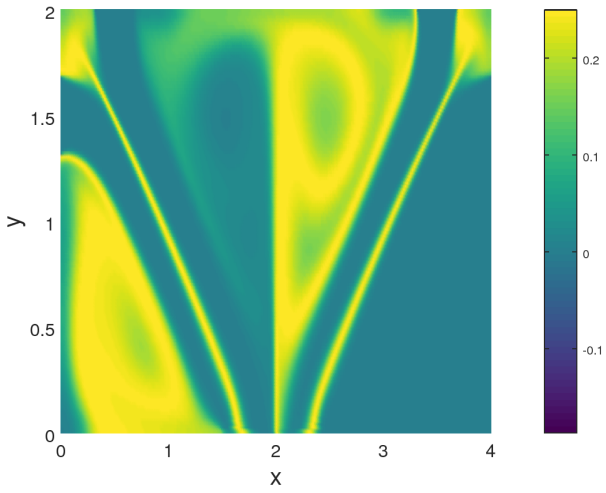

Figure 88: Solution for $Y(1 - Y)$ at $t = 12.0$

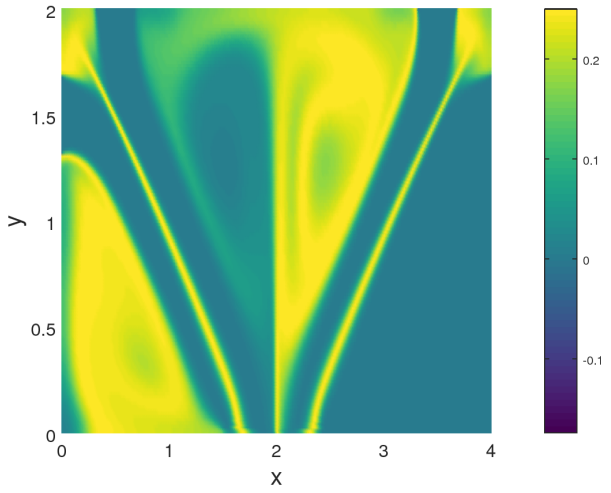

Figure 89: Solution for $Y(1-Y)$ at $t = 14.0$

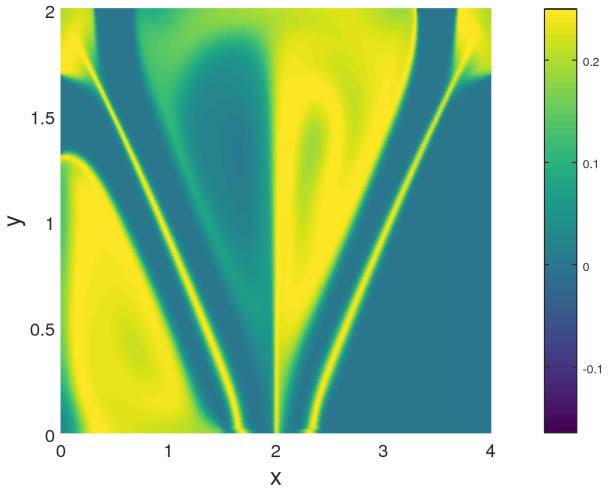

Figure 90: Solution for $Y(1-Y)$ at $t = 16.0$

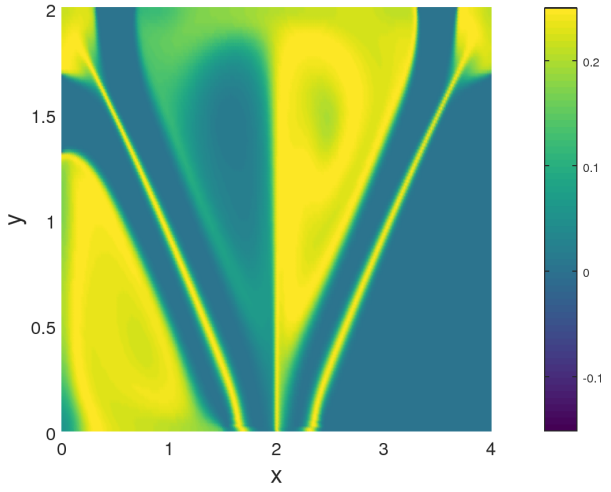

Figure 91: Solution for $Y(1-Y)$ at $t = 18.0$

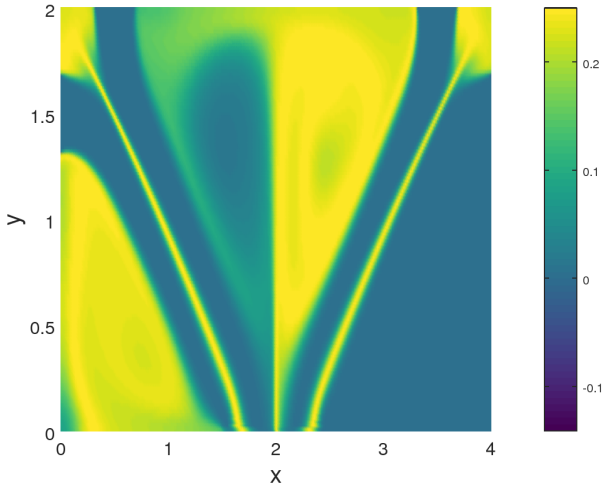

Figure 92: Solution for $Y(1-Y)$ at $t = 20.0$

5.5 Time Evolution of $R(t)$ and $S(t)$

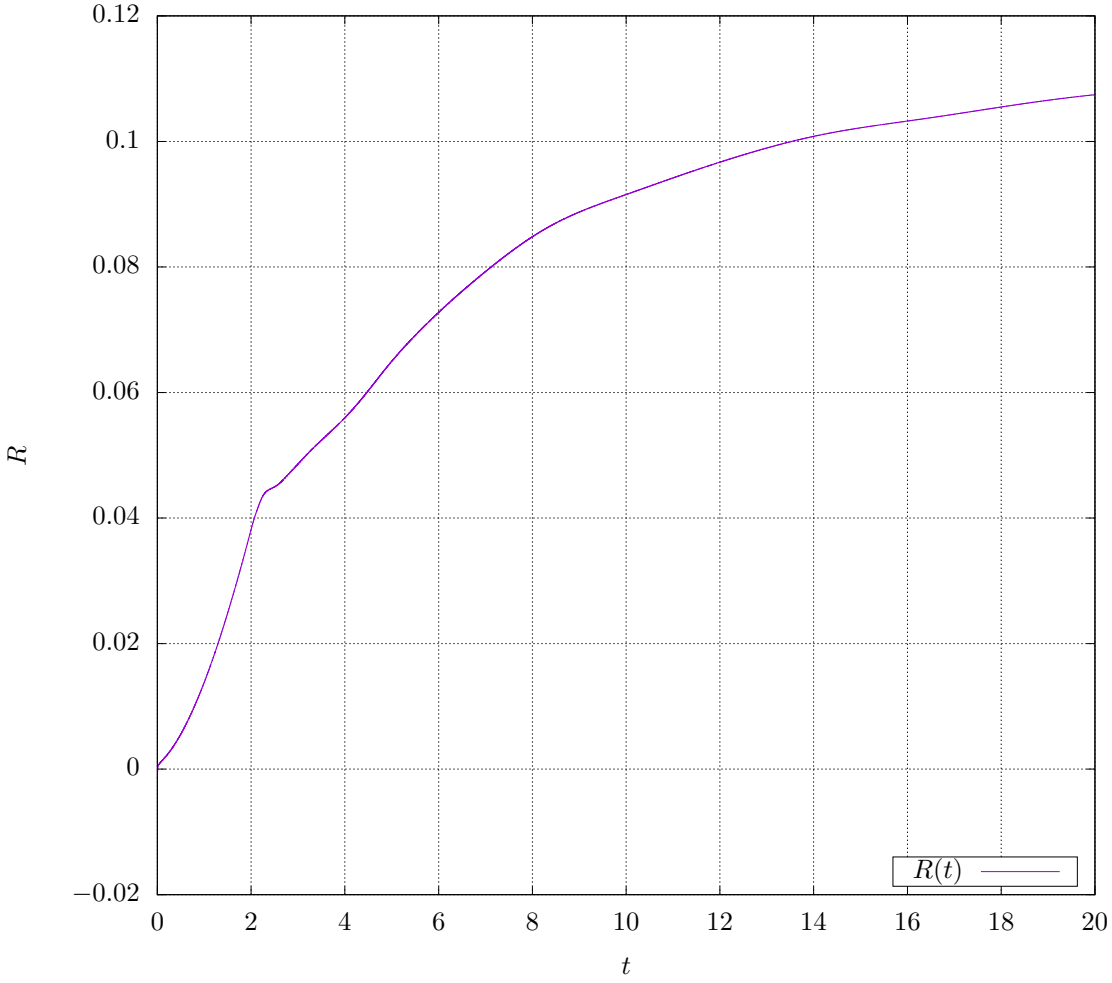

Figure 93: $R(t)$

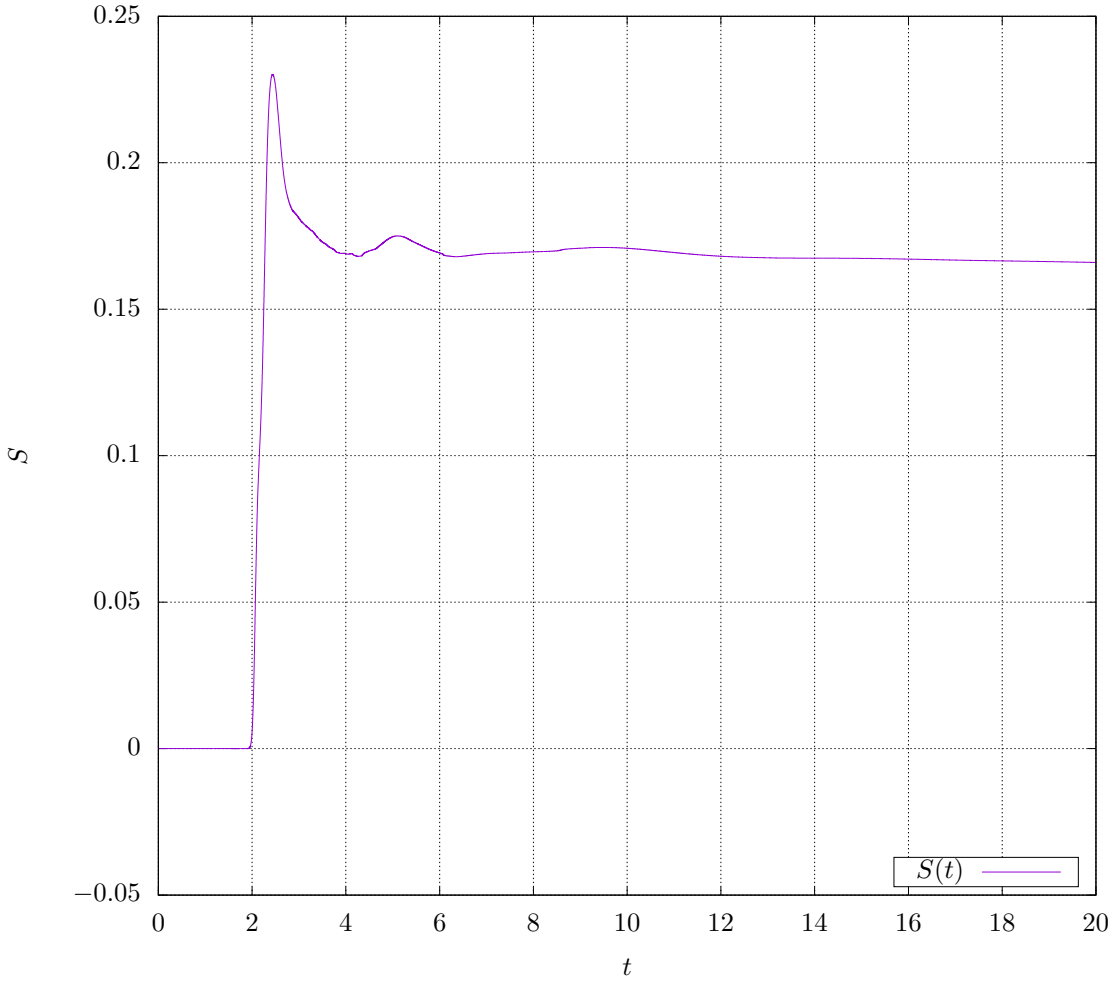

Figure 94: $S(t)$

5.6 Solution verifcation for T

Using the same analysis as shown for problem 1, the quantity T will be determined to within an acceptable error range.

Table 7: GCI Analysis

Grid	Cells: M	Cells: N	T	p	range
1	512	256	0.130849	2.28	1.28
2	256	128	0.167478	-1.765	2.063
3	128	64	0.345426	-0.740	1.152
4	64	32	0.397774	2.096	0.921
5	32	16	0.366431		
6	16	8	0.500472		

The observed order of convergence for this simulation is a little all over the place with the early meshes, but at the finest mesh run it is getting closer to the expected order of convergence of 2. The asymptotic range of convergence is a major concern, since it should be very close to 1 but it is much higher than 1 even at the finest mesh. It was observed early on that this simulation was having stability issues with higher CFL's and

it's possible that the CFL of 0.3 was still to high for this simulation and that an even lower CFL is required to get an accurate result. If there was more time alloted for this section, the simulation would be repeated at a smaller CFL and with a finer mesh of $M = 1024$ cells to see if the observed order of convergence would approach 2 and if the asymptotic range of convergence would approach 1.

$$f_{h=0} \approx f_1 + \frac{f_1 - f_2}{r^p - 1}$$

$$T_{h=0} \approx 0.121355$$

Calculate GCI's using $F_{sec} = 1.25$.

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

$$GCI_{12} = 9.06\%$$

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

$$GCI_{23} = 34.4\%$$

The final solution for T is

$$T = 0.12135 \pm 9.06\%.$$

Due to the less than desirable asymptotic range of convergence and the strange behavior of the observed orders of convergence, the accuracy of the result for T is much lower than in problem 1 and 2. Again this could be improved with more time to run simulations that have finer meshes and lower CFL numbers.

Appendix

A Included Code

"Final.f95":

```
program Final
  ! use statement includes module files
  use Vcycle
  use AdamsB
  use probe
  use CN
  use swissarmy
  use WENO5
  implicit none
  integer :: i, j, M, N, iterations, nItermax, timeiterations, totaltimes,&
  n1, numleftinlets, numrightinlets, numtopinlets, moviesteps!, iu, ju, iv, jv, iY, jY
  real (kind = 8) :: dt, deltax, deltay, dx, dy, d1, d2, Re, Sc, x0, y0, xM, yN, alpha,
↪  CFL, time, &
  timefinal, start, finish, uprobe, vprobe, Yprobe, Ydx, Ydy, Yd1, Yd2, dtu, dtv, dtY, R,
↪  eps, S,&
  inlet_length, VinY1, VinY0, Veps
  real (kind = 8), dimension(:), allocatable :: au, bu, cu, du, ub, ul, ur, ut, xu, yu
  real (kind = 8), dimension(:), allocatable :: av, bv, cv, dv, vt, xv, yv, xphi, yphi,
↪  dphidn, Linf
  real (kind = 8), dimension(:), allocatable :: ay, by, cy, dyy, xy, yy, timesrequested,
↪  leftinletloc, rightinletloc, topinletloc,&
  Yinletconditions, ual, uar, vat, Yall_loc
  real (kind = 8), dimension(:,:), allocatable :: u, unext, uprev, uprev1, v, vnext,
↪  vprev, vprev1, &
  Y, Ynext, Yprev, Yprev1, Yconv, phi, gradphix, gradphiy, f, Qu, Qv, QY

  call cpu_time(start)

  ! Define some of the problem constants
  Re = 50.0d0; Sc = 1.0d0; x0 = 0.0d0; y0 = 0.0d0; xM = 4.0d0; yN = 2.0d0;
  alpha = 1.0d0/(Re*Sc); CFL = 0.8d0; M = 256; N = 128; deltax = (xM-x0)/dble(M);
  deltay = (yN-y0)/dble(N); time = 0.0d0; eps = 1.0d-10; Veps = 1.0d-10;

  ! Set the number of inlets on each wall
  numleftinlets = 1; ! Leave at 1 for Problem 2
  numtopinlets = 1; ! Change to 2 for Problem 2
  numrightinlets = 1; ! Leave at 1 for Problem 2
  inlet_length = dble(1.5/(numtopinlets+numleftinlets+numrightinlets))


↪  allocate(leftinletloc(1:numleftinlets),rightinletloc(1:numrightinlets),topinletloc(1:numtopinlets),&

↪  Yinletconditions(1:numleftinlets+numtopinlets+numrightinlets),Yall_loc(1:numleftinlets+numtopinlets+

  ! Locate inlets on the walls
  leftinletloc(1:numleftinlets) = (/ 0.75 /); ! Change to 1.5 for problem 2
  topinletloc(1:numtopinlets) = (/ 0.75 /); ! Change to 0.5, 3.5 for problem 2
  rightinletloc(1:numrightinlets) = (/ 1.25 /); ! Change to 1.5 for problem 2
```

```
42
43    allocate(ual(1:numleftinlets),uar(1:numrightinlets),vat(1:numtopinlets))
44
45    ! Set average velocities for left, right and top inlets
46    ual(1:numleftinlets) = (/ 1.0d0 /); ! Change to 1.3d0 for problem 2
47    uar(1:numrightinlets) = (/ -1.0d0 /); ! Change to -1.3d0 for problem 2
48    vat(1:numtopinlets) = (/ -1.0d0 /); ! Change to -1.3d0, -1.3d0 for problem 2
49
50    ! Set inlet concentrations for left, top and right inlets
51    Yinletconditions(1:numleftinlets+numtopinlets+numrightinlets) = (/ 1.0d0, 0.0d0, 1.0d0
      ↪  /); ! Change to 1.0d0, 0.0d0, 1.0d0, 0.0d0 for problem 2
52    Yall_loc(1:numleftinlets) = leftinletloc;
53    Yall_loc(numleftinlets+1:numtopinlets+numleftinlets) = topinletloc;
54    Yall_loc(numtopinlets+numleftinlets+1:numleftinlets+numtopinlets+numrightinlets) =
      ↪  rightinletloc;
55
56    ! Change timefinal to obtain different results
57    totaltimes = 6
58    allocate(timesrequested(1:totaltimes))
59    timesrequested(1:totaltimes) = (/ 1.0, 3.0, 5.0, 10.0, 15.0, 20.0 /)
60    moviesteps = 0;
61
62    ! Populate data arrays used for u
63    allocate(xu(0:M),yu(0:N+1))
64    call popu(xu,yu,deltax,deltay,x0,y0,N,M)
65
66    ! Apply boundary conditions
67    allocate(ub(0:M),ut(0:M),ul(0:N+1),ur(0:N+1))
68    ub(:) = 0.0d0; ut(:) = 0.0d0
69
70
      ↪  allocate(u(0:M,0:N+1),unext(0:M,0:N+1),uprev(0:M,0:N+1),uprev1(0:M,0:N+1),Qu(0:M,0:N+1));
71    call
      ↪  iboundaryu(u,ual,uar,yu,ul,ur,ub,ut,numleftinlets,numrightinlets,leftinletloc,rightinletloc,inlet_l
72    ! call ICu(u,xu,yu,N,M);
73    call boundaryu(u,ur,ul,N,M)
74    uprev(:,:) = u(:,:);
75    allocate(au(1:(M-1)*N),bu(1:(M-1)*N),cu(1:(M-1)*N),du(1:(M-1)*N))
76
77    ! Populate data arrays used for v
78    allocate(xv(0:M+1),yv(0:N))
79    call popv(xv,yv,deltax,deltay,x0,y0,N,M)
80
81    ! Apply boundary conditions
82    allocate(vt(0:M+1));
83
      ↪  allocate(v(0:M+1,0:N),vnext(0:M+1,0:N),vprev(0:M+1,0:N),vprev1(0:M+1,0:N+1),Qv(0:M+1,0:N))
84    call iboundaryv(v,vat,xv,vt,numtopinlets,topinletloc,inlet_length,N,M)
85    ! call ICv(v,xv,yv,N,M);
86    call boundaryv(v,xv,vt,N,M)
87    vprev(:,:) = v(:,:);
88    allocate(av(1:(N-1)*M),bv(1:(N-1)*M),cv(1:(N-1)*M),dv(1:(N-1)*M))
89
90    ! Populate data arrays used for Y
```

```
91    allocate(xy(-2:M+3),yy(-2:N+3))
92    call popY(xy,yy,deltax,deltay,x0,y0,N,M)
93
94    ! Apply boundary conditions
95
      ↪  allocate(Y(-2:M+3,-2:N+3),Ynext(-2:M+3,-2:N+3),Yprev(-2:M+3,-2:N+3),Yprev1(-2:M+3,-2:N+3),Qy(-2:M+3
96    call
      ↪  iboundaryY(Y,Yinletconditions,xy,yy,numleftinlets,numtopinlets,numrightinlets,Yall_loc,inlet_length
97    ! call ICY(Y,xy,yy,N,M);
98    call
      ↪  boundaryY(Y,Yinletconditions,xy,yy,numleftinlets,numtopinlets,numrightinlets,Yall_loc,inlet_length,I
99    Yprev(:,:) = Y(:,:); Yconv(:,:) = Y(:,:);
100
101   allocate(ay(1:M*N),by(1:M*N),cy(1:M*N),dyy(1:M*N))
102
103   ! Populate data arrays used for phi
104   nItermax = 1000;
105   allocate(phi(0:M+1,0:N+1),gradphix(0:M+1,0:N+1),gradphiy(0:M+1,0:N+1),f(0:M+1,0:N+1))
106   allocate(xphi(0:M+1),yphi(0:N+1),dphidn(0:3),Linf(0:nItermax)); dphidn(:) = 0.0d0;
107   call popPhi(xphi,yphi,x0,y0,deltax,deltay,N,M)
108   phi(:,:)=0.0d0;
109
110   ! Mark the indeces that match the probe location
111   ! uprobe: x = 1, y = 0.5
112   ! vprobe: x = 1, y = 1.5
113   ! Yprobe: x = 2, y = 0.5
114   ! call findindex(xu,yu,iu,ju,1.0d0,0.5d0,M,N+1)
115   ! call findindex(xv,yv,iv,jv,1.0d0,1.5d0,M+1,N)
116   ! call findindexY(xy,yy,iy,jy,2.0d0,0.5d0,M+3,N+3)
117
118
119   ! Interpolate to calculate the probe value
120   iterations = 0;
121   ! call interpolate3D(xu,yu,iu,ju,1.0d0,0.5d0,M,N+1,uprobe,u)
122   ! call interpolate3D(xv,yv,iv,jv,1.0d0,1.5d0,M+1,N,vprobe,v)
123   ! call interpolate3DY(xy,yy,iy,jy,2.0d0,0.5d0,M+3,N+3,Yprobe,Y)
124
125
126   ! open (11, file = 'dummyuprobe'); open (12, file = 'dummyvprobe'); open (13, file =
      ↪  'dummyYprobe'); open(14, file ='dummyRprobe')
127   ! open(15, file ='dummySprobe'); open(16, file ='dummyTprobe'); open(17, file
      ↪  ='dummyVinY1Probe'); open(18, file='dummyVinY0Probe')
128   ! Uncomment the next line to store data in the correct probe files
129   open (11, file = 'uprobe'); open (12, file = 'vprobe'); open(13, file='Yprobe');
      ↪  open(14, file ='Rprobe')
130   open(15, file ='Sprobe'); open(16, file='Tprobe'); open(17,file ='VinY1Probe');
      ↪  open(18, file='VinY0Probe')
131   open(19, file='inlet1'); open(20, file='inlet2'); open(21, file='inlet3'); open(22,
      ↪  file='inlet4')
132   ! write (11,*) time, uprobe; write (12,*) time, vprobe; write (13,*) time, Yprobe;
133
134   do timeiterations = 1,totaltimes
135
136     timefinal = timesrequested(timeiterations);
```

```
137
138     do while(time<timefinal)
139       ! do iterations = 1,2
140
141       call calcR(R,Y,xM,yN,deltax,deltay,N,M)
142       write(14,*) time, R
143       call calcS(S,Y,v,deltax,1.0d0,N,M)
144       write(15,*) time, S
145       if (time >= 10.0d0) then
146         write(16,*) time, S*dt
147       end if
148       write(19,*) time, ual(1), Yinletconditions(1)
149       write(20,*) time, vat(1), Yinletconditions(2)
150       write(21,*) time, vat(2), Yinletconditions(3)
151       write(22,*) time, uar(1), Yinletconditions(4)
152
153       ! Calculate volumetric flow rates into the chamber for Y = 1 and Y = 0 inlets
154       call Volflow(VinY1,VinY0,numleftinlets,numtopinlets,numrightinlets,&
155
    ↪  Yinletconditions,Yall_loc,inlet_length,N,M,deltax,deltay,u(0,:),u(M,:),v(:,N),xv,yu)
156       write(17,*) time, VinY1
157       write(18,*) time, VinY0
158
159
160       ! Store current values to prev1 variables
161       uprev1 = u; vprev1 = v; Yprev1 = Y; Yconv = Y;
162
163       dtu = (deltax*deltay)/(deltay*maxval(dabs(u))+deltax*maxval(dabs(v))+eps)
164       dtv = (deltax*deltay)/(deltay*maxval(dabs(u))+deltax*maxval(dabs(v))+eps)
165       dtY = (deltax*deltay)/(deltay*maxval(dabs(u))+deltax*maxval(dabs(v))+eps)
166
167       dt = CFL*min(dtu,dtv,dtY)
168
169       dx = dt/(Re*deltax**2); dy = dt/(Re*deltay**2); d1 = dx/2.0d0; d2 = dy/2.0d0;
170       Ydx = dt/(Re*Sc*deltax**2); Ydy = dt/(Re*Sc*deltay**2); Yd1 = Ydx/2.0d0; Yd2 =
    ↪  Ydy/2.0d0;
171
172       ! Calculate Adams-Bashforth part for u and v equations
173
174       call TVDRK3(Yconv,u,v,N,M,xy,yy,deltax,deltay,dt,Yall_loc,&
175       inlet_length,Yinletconditions,numleftinlets,numtopinlets,numrightinlets)
176       call crunchQu(u,v,uprev,vprev,Qu,deltax,deltay,N,M)
177       call crunchQv(u,v,uprev,vprev,Qv,deltax,deltay,N,M)
178       Qy = 0.0d0;
179
180       do j = 1,N
181         do i = 1,M
182           Qy(i,j) = (Yconv(i,j)-Y(i,j))/dt; !print *, i+3, j+3, Yconv(i,j), Y(i,j)
183         end do
184       end do
185
186       !~~~~~~~~~~~~~~~~~~~~ Begin Crank Nicolson ADI ~~~~~~~~~~~~~~~~~~~~!
187
188       !~~~~~~~~~~~~~~~~~~~~~~~~ Implicit x ~~~~~~~~~~~~~~~~~~~~~~~~~~~!
```

```
189
190       !~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ u ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~!
191       call setux(au,bu,cu,du,Qu,d1,d2,dt,ul,ur,u,N,M)
192       ! print *, '!~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ u Implicit x
    ↪  ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~!',
193       ! do i = 1,(M-1)*N
194       !     print *, i, du(i)
195       ! end do
196       call tdsolve(au,bu,cu,du,1,(M-1)*N)
197       call sortux(unext,ub,ut,ul,ur,du,N,M)
198       ! Store unext in u to be used for implicit y
199       u = unext;
200
201       !~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ v ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~!
202       call setvx(av,bv,cv,dv,Qv,d1,d2,dt,vt,v,N,M,xv)
203       ! print *, '!~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ v Implicit x
    ↪  ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~!',
204       ! do i = 1,(N-1)*M
205       !     print *, i, dv(i)
206       ! end do
207       call tdsolve(av,bv,cv,dv,1,(N-1)*M)
208       call sortvx(vnext,dv,xv,vt,N,M)
209       ! Store vnext in v to be used for implicit y
210       v = vnext;
211
212       !~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ Y ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~!
213       call
    ↪  setYx(ay,by,cy,dyy,Qy,Yd1,Yd2,dt,Y,N,M,yy,xy,Yinletconditions,numleftinlets,numrightinlets,numtopinl
214       Yall_loc,inlet_length)
215       ! print *, '!~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ Y Implicit x
    ↪  ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~!',
216       ! do i = 1,N*M
217       !     print *, i, dyy(i)
218       ! end do
219       call tdsolve(ay,by,cy,dyy,1,N*M)
220       call sortYx(Ynext,dyy,N,M)
221       call
    ↪  boundaryY(Ynext,Yinletconditions,xy,yy,numleftinlets,numtopinlets,numrightinlets,Yall_loc,inlet_leng
222       ! Store Ynext in Y to be used for implicit y
223       Y = Ynext;
224
225       !~~~~~~~~~~~~~~~~~~~~~~~~ Implicit y ~~~~~~~~~~~~~~~~~~~~~~~~~~~!
226
227       !~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ u ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~!
228       call setuy(au,bu,cu,du,Qu,d1,d2,dt,ul,ur,u,N,M)
229       ! print *, '!~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ u Implicit y
    ↪  ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~!',
230       ! do i = 1,(M-1)*N
231       !     print *, i, du(i)
232       ! end do
233       call tdsolve(au,bu,cu,du,1,(M-1)*N)
234       call sortuy(unext,ub,ut,ul,ur,du,N,M)
235       ! Store unext in u to be used for next time step iteration
236       u = unext;
```

```
237
238         !~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ v ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~!
239         call setvy(av,bv,cv,dv,Qv,d1,d2,dt,vt,v,N,M,xv)
240         ! print *, '!~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ v Implicit y
    ↪   ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~!'
241         ! do i = 1,(N-1)*M
242         !     print *, i, dv(i)
243         ! end do
244         call tdsolve(av,bv,cv,dv,1,(N-1)*M)
245         call sortvy(vnext,dv,xv,vt,N,M)
246         ! Store vnext in v to be used for next time step
247         v = vnext;
248
249         !~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ Y ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~!
250         call setYy(ay,by,cy,dyy,Qy,Yd1,Yd2,dt,Y,N,M,yy,xy,Yinletconditions,&
251         numleftinlets,numrightinlets,numtopinlets,Yall_loc,inlet_length)
252         ! print *, '!~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ Y Implicit y
    ↪   ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~!'
253         ! do i = 1,N*M
254         !     print *, i, dyy(i)
255         ! end do
256         call tdsolve(ay,by,cy,dyy,1,N*M)
257         call sortYy(Ynext,dyy,N,M)
258         call
    ↪   boundaryY(Ynext,Yinletconditions,xy,yy,numleftinlets,numtopinlets,numrightinlets,Yall_loc,inlet_leng
259         ! Store Ynext in Y to be used for next time step
260         Y = Ynext;
261
262         ! ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ Phi ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~!
263
264         call
    ↪   Lagrange(u,v,unext,vnext,deltax,deltay,dt,xv,phi,Linf,nItermax,dphidn,N,M,n1,Veps)
265
266         if(time+dt>=timefinal) then
267             dt = timefinal - time
268         end if
269
270         time = time + dt
271         iterations = iterations + 1;
272
273         if (mod(iterations,100)==(0)) then
274           if (n1 > nItermax) then
275             n1 = n1 - 1
276           end if
277           call diagnostics(iterations,start,time,timesrequested(totaltimes),Linf(n1))
278         end if
279
280         ! Interpolate to calculate the probe value
281         ! call interpolate3D(xu,yu,iu,ju,1.0d0,0.5d0,M,N+1,uprobe,u)
282         ! call interpolate3D(xv,yv,iv,jv,1.0d0,1.5d0,M+1,N,vprobe,v)
283         ! call interpolate3DY(xy,yy,iy,jy,2.0d0,0.5d0,M+3,N+3,Yprobe,Y)
284         ! write (11,*) time, uprobe; write (12,*) time, vprobe; write (13,*) time, Yprobe;
285
286         uprev = uprev1; vprev = vprev1; Yprev = Yprev1;
```

```
287
288       end do
289
290       call savedata(u,xu,yu,v,xv,yv,Y,xy,yy,N,M,time)
291
292     end do
293
294     print '("Simulation time =",d20.10," " )', time
295     print '("Expected time =",f12.10," " )', timefinal
296     print '("Iterations =",I0," ")', iterations
297     print '("Delta x =",f12.10," ")', deltax
298     print '("Delta y =",f12.10," ")', deltay
299     print '("Uprobe =",f20.10," ")', uprobe
300     print '("Vprobe =",f20.10," ")', vprobe
301     print '("Yprobe =",f20.10," ")', Yprobe
302     print '("Rprobe =",f20.10," ")', R
303
304
305     call cpu_time(finish)
306     print '("Computational Time = ",f20.10," seconds.")',finish-start
307
308
309   end program Final
```

"AdamsB.f95":

```
1   module AdamsB
2
3     implicit none
4     private
5     public :: crunchQu, crunchQv, AdamsHu, AdamsHv, du2dx, duvdy, dv2dy, duvdx, uij,
    ↪  ui12j12, vi12j12, vij
6
7     contains
8
9     subroutine crunchQu(u,v,uprev,vprev,Qu,deltax,deltay,N,M)
10      implicit none
11      integer :: i, j, M, N
12      real (kind = 8) :: deltax, deltay
13      real (kind = 8), dimension(0:M,0:N+1) :: u, uprev, Qu
14      real (kind = 8), dimension(0:M+1,0:N) :: v, vprev
15      Qu(:,:) = 0.0d0;
16      do i = 1,M-1
17        do j = 1,N
18          Qu(i,j) = (3.0d0/2.0d0)*(-1.0d0*AdamsHu(u,v,i,j,deltax,deltay,N,M)) -&
19           (1.0d0/2.0d0)*(-1.0d0*AdamsHu(uprev,vprev,i,j,deltax,deltay,N,M));
20        end do
21      end do
22    end subroutine crunchQu
23
24    subroutine crunchQv(u,v,uprev,vprev,Qv,deltax,deltay,N,M)
25      implicit none
26      integer :: i, j, M, N
27      real (kind = 8) :: deltax, deltay
28      real (kind = 8), dimension(0:M,0:N+1) :: u, uprev
```

```
29      real (kind = 8), dimension(0:M+1,0:N) :: v, vprev, Qv
30      Qv(:,:) = 0.0d0;
31      do i = 1,M
32        do j = 1,N-1
33          Qv(i,j) = (3.0d0/2.0d0)*(-1.0d0*AdamsHv(u,v,i,j,deltax,deltay,N,M)) -&
34           (1.0d0/2.0d0)*(-1.0d0*AdamsHv(uprev,vprev,i,j,deltax,deltay,N,M));
35        end do
36      end do
37    end subroutine crunchQv
38
39    function AdamsHu(u,v,i,j,deltax,deltay,N,M)
40    implicit none
41    integer :: i, j, N, M
42    real (kind = 8) :: deltax, deltay, AdamsHu
43    real (kind = 8), dimension (0:M,0:N+1) :: u
44    real (kind = 8), dimension (0:M+1,0:N) :: v
45    AdamsHu = du2dx(u,i,j,deltax,N,M) + duvdy(u,v,i,j,deltay,N,M);
46    end function AdamsHu
47
48    function AdamsHv(u,v,i,j,deltax,deltay,N,M)
49    implicit none
50    integer :: i, j, N, M
51    real (kind = 8) :: deltax, deltay, AdamsHv
52    real (kind = 8), dimension (0:M,0:N+1) :: u
53    real (kind = 8), dimension (0:M+1,0:N) :: v
54    AdamsHv = duvdx(u,v,i,j,deltax,N,M) + dv2dy(v,i,j,deltay,N,M);
55    end function AdamsHv
56
57    function du2dx(u,i,j,deltax,N,M)
58    implicit none
59    integer :: i, j, N, M
60    real (kind = 8) :: deltax, du2dx
61    real (kind = 8), dimension (0:M,0:N+1) :: u
62    du2dx = ((uij(u,i+1,j,N,M))**2-(uij(u,i,j,N,M))**2)/deltax;
63    end function du2dx
64
65    function duvdy(u,v,i,j,deltay,N,M)
66    implicit none
67    integer :: i, j, N, M
68    real (kind = 8) :: deltay, duvdy
69    real (kind = 8), dimension (0:M,0:N+1) :: u
70    real (kind = 8), dimension (0:M+1,0:N) :: v
71    duvdy = ((ui12j12(u,i,j,N,M)*vi12j12(v,i,j,N,M)) -
    ↪  (ui12j12(u,i,j-1,N,M)*vi12j12(v,i,j-1,N,M)))/deltay
72    end function duvdy
73
74    function dv2dy(v,i,j,deltay,N,M)
75    implicit none
76    integer :: i, j, N, M
77    real (kind = 8) :: deltay, dv2dy
78    real (kind = 8), dimension (0:M+1,0:N) :: v
79    dv2dy = ((vij(v,i,j+1,N,M))**2-(vij(v,i,j,N,M))**2)/deltay;
80    end function dv2dy
81
```

```
  function duvdx(u,v,i,j,deltax,N,M)
  implicit none
  integer :: i, j, N, M
  real (kind = 8) :: deltax, duvdx
  real (kind = 8), dimension (0:M,0:N+1) :: u
  real (kind = 8), dimension (0:M+1,0:N) :: v
  duvdx = ((ui12j12(u,i,j,N,M)*vi12j12(v,i,j,N,M)) -
 ↪  (ui12j12(u,i-1,j,N,M)*vi12j12(v,i-1,j,N,M)))/deltax
  end function duvdx

  function uij(u,i,j,N,M)
  implicit none
  integer :: i, j, N, M
  real (kind = 8) :: uij
  real (kind = 8), dimension(0:M,0:N+1) :: u
  uij = (0.5d0)*(u(i,j)+u(i-1,j));
  end function uij

  function vij(v,i,j,N,M)
  implicit none
  integer :: i, j, N, M
  real (kind = 8) :: vij
  real (kind = 8), dimension(0:M+1,0:N) :: v
  vij = (0.5d0)*(v(i,j)+v(i,j-1));
  end function vij

  function ui12j12(u,i,j,N,M)
  implicit none
  integer :: i, j, N, M
  real (kind = 8) :: ui12j12
  real (kind = 8), dimension (0:M,0:N+1) :: u
  ui12j12 = (0.5d0)*(u(i,j)+u(i,j+1));
  end function ui12j12

  function vi12j12(v,i,j,N,M)
  implicit none
  integer :: i, j, N, M
  real (kind = 8) :: vi12j12
  real (kind = 8), dimension (0:M+1,0:N) :: v
  vi12j12 = (0.5d0)*(v(i,j)+v(i+1,j));
  end function vi12j12


end module AdamsB
```

"CN.f95":

```
module CN

  use swissarmy
  implicit none

  private
  public :: tdsolve, setux, setuy, setvx, setvy, setYx, setYy, sortux, sortuy, sortvx,
 ↪  sortvy, sortYx, sortYy
```

```
8
9   contains
10
11    subroutine sortux(unext,ub,ut,ul,ur,du,N,M)
12      integer :: i, j, N, M
13      real (kind = 8), dimension(0:M,0:N+1) :: unext
14      real (kind = 8), dimension(0:M) :: ub, ut
15      real (kind = 8), dimension(0:N+1) :: ul, ur
16      real (kind = 8), dimension(1:(M-1)*N) :: du
17      ! Sort the values of the 1D vector d into the 2D matrix unext
18      unext(0,:) = ul; unext(M,:) = ur; unext(:,N+1) = ut; unext(:,0) = ub;
19      do j = 1,N
20          do i = 1,M-1
21              unext(i,j) = du((j-1)*(M-1)+i)
22          end do
23      end do
24
25    end subroutine sortux
26
27    subroutine sortuy(unext,ub,ut,ul,ur,du,N,M)
28      integer :: i, j, N, M
29      real (kind = 8), dimension(0:M,0:N+1) :: unext
30      real (kind = 8), dimension(0:M) :: ub, ut
31      real (kind = 8), dimension(0:N+1) :: ul, ur
32      real (kind = 8), dimension(1:(M-1)*N) :: du
33      ! Sort the values of the 1D vector d into the 2D matrix unext
34      unext(0,:) = ul; unext(M,:) = ur; unext(:,N+1) = ut; unext(:,0) = ub;
35      do i = 1,M-1
36          do j = 1,N
37              unext(i,j) = du((i-1)*N+j)
38          end do
39      end do
40      ! Apply boundary conditions to bottom and top walls via second order interpolation
41      unext(:,0) = -unext(:,1); unext(:,N+1) = -unext(:,N);
42    end subroutine sortuy
43
44    subroutine sortvx(vnext,dv,xv,vt,N,M)
45      integer :: i, j, N, M
46      real (kind = 8), dimension(0:M+1,0:N) :: vnext
47      real (kind = 8), dimension(1:(N-1)*M) :: dv
48      real (kind = 8), dimension(0:M+1) :: xv, vt
49      ! Sort the values of the 1D vector d into the 2D matrix vnext
50      do j = 1,N-1
51          do i = 1,M
52              vnext(i,j) = dv((j-1)*M+i)
53          end do
54      end do
55
56      j = 1; ! Set bottom cells (Neumann condition)
57      do i = 0,M+1
58          if ((xv(i)>1.5d0).and.(xv(i)<2.5d0)) then
59              vnext(i,0) = (4.0d0/3.0d0)*vnext(i,j)-(1.0d0/3.0d0)*vnext(i,j+1)
60          else
61              vnext(i,0) = 0.0d0
```

```
62          end if
63      end do
64
65      ! Boundary conditions for left and right walls plus top wall
66      vnext(0,:) = -vnext(1,:); vnext(M+1,:) = -vnext(M,:); vnext(:,N) = vt;
67  end subroutine sortvx
68
69  subroutine sortvy(vnext,dv,xv,vt,N,M)
70      implicit none
71      integer :: i, j, N, M
72      real (kind = 8), dimension(0:M+1,0:N) :: vnext
73      real (kind = 8), dimension(1:(N-1)*M) :: dv
74      real (kind = 8), dimension(0:M+1) :: xv, vt
75      ! Sort the values of the 1D vector d into the 2D matrix vnext
76
77      do i = 1,M
78          do j = 1,N-1
79              vnext(i,j) = dv((i-1)*(N-1)+j)
80          end do
81      end do
82
83      ! Boundary conditions for left and right walls plus top wall
84      vnext(:,N) = vt; vnext(0,:) = -vnext(1,:); vnext(M+1,:) = -vnext(M,:);
85
86      j = 1; ! Set bottom cells (Neumann condition)
87      do i = 0,M+1
88          if ((xv(i)>1.5d0).and.(xv(i)<2.5d0)) then
89              vnext(i,0) = (4.0d0/3.0d0)*vnext(i,j)-(1.0d0/3.0d0)*vnext(i,j+1);
90          else
91              vnext(i,0) = 0.0d0
92          end if
93      end do
94  end subroutine sortvy
95
96  subroutine sortYx(Ynext,dyy,N,M)
97      integer :: i, j, N, M
98      real (kind = 8), dimension(1:M*N) :: dyy
99      real (kind = 8), dimension(-2:M+3,-2:N+3) :: Ynext
100     ! Sort the values of the 1D vector into the 2D matrix Ynext
101     do j = 1,N
102         do i = 1,M
103             Ynext(i,j) = dyy((j-1)*M+i);
104         end do
105     end do
106
107 end subroutine sortYx
108
109 subroutine sortYy(Ynext,dyy,N,M)
110     integer :: i, j, N, M
111     real (kind = 8), dimension(1:M*N) :: dyy
112     real (kind = 8), dimension(-2:M+3,-2:N+3) :: Ynext
113     do i = 1,M
114         do j = 1,N
115             Ynext(i,j) = dyy((i-1)*N+j)
```

```
116           end do
117       end do
118
119     end subroutine sortYy
120
121     subroutine tdsolve(a,b,c,d,st,ed)
122         implicit none
123         real (kind=8), dimension(st:ed) :: a, b, c, d
124         integer :: i, st, ed
125
126     !     print *, 'Before forward elimination'
127     !     ! Print tri-diagonal vectors before forward elimination
128     !     do i = st,ed
129     !         print *, 'a(',i,') =', a(i), 'b(',i,') =',
    ↪  b(i),'c(',i,')=',c(i),'d(',i,')=',d(i)
130     !     end do
131     ! ! for loop to go through rows and eliminate
132         do i = st+1,ed
133             b(i) = b(i) - c(i-1)*(a(i)/b(i-1))
134             d(i) = d(i) - d(i-1)*(a(i)/b(i-1))
135         end do
136
137     !     print *, 'After forward elimination'
138         ! Print tri-diagonal vectors after forward elimination
139     !     do i = st,ed
140     !         print *, 'a(',i,') =', a(i), 'b(',i,') =',
    ↪  b(i),'c(',i,')=',c(i),'d(',i,')=',d(i)
141     !     end do
142
143
144     ! Back-substitute
145         d(ed) = d(ed)/b(ed)
146         do i = ed-1, st, -1 ! loop through rows
147             d(i) = (d(i) - c(i)*d(i+1))/b(i)
148         end do
149
150     !     print *, 'After back substitution'
151         ! Print tri-diagonal vectors after back substitution
152     !     do i = st,ed
153     !         print *, 'd(',i,')=',d(i)
154     !     end do
155
156     end subroutine tdsolve
157
158     subroutine setux(a,b,c,d,Qu,d1,d2,dt,ul,ur,u,N,M)
159     implicit none
160     integer :: i, j, ii, M, N
161     real (kind = 8) :: d1, d2, dt
162     real (kind = 8), dimension (1:(M-1)*N) :: a, b, c, d
163     real (kind = 8), dimension (0:N+1) :: ul, ur
164     real (kind = 8), dimension (0:M,0:N+1) :: u, Qu
165
166     ii = 1;
167     ! Set bottom corner i = 1, j = 1
```

```
168     a(1) = 0.0d0; b(1) = 1.0d0 + 2.0d0*d1; c(1) = -d1; d(1) =
    ↪  d2*u(1,2)+(1.0d0-3.0d0*d2)*u(1,1)+(dt/2.0d0)*Qu(1,1);
169     ii = ii + 1;
170
171     ! Fill up entries on the bottom row (i.e. j = 1 & 1 < i < M-2 )
172     j = 1;
173     do i = 2,M-2
174         a(ii) = -d1; b(ii) = 1.0d0 + 2.0d0*d1; c(ii) = -d1; d(ii) =
    ↪  d2*u(i,j+1)+(1.0d0-3.0d0*d2)*u(i,j)+(dt/2.0d0)*Qu(i,j);
175         ii = ii + 1;
176     end do
177
178     ! Set bottom corner i = M, j = 1
179     a(ii) = -d1; b(ii) = 1.0d0 + 2.0d0*d1; c(ii) = 0.0d0; d(ii) =
    ↪  d2*u(i,j+1)+(1.0d0-3.0d0*d2)*u(i,j)+(dt/2.0d0)*Qu(i,j);
180     ii = ii + 1
181
182     ! Fill up entries from the 2nd row to the N-1 row
183     do j = 2,N-1
184         ! Start with the left wall
185         i = 1
186         a(ii) = 0.0d0; b(ii) = 1.0d0 + 2.0d0*d1; c(ii) = -d1
187         d(ii) =
    ↪  d2*u(i,j+1)+(1.0d0-2.0d0*d2)*u(i,j)+d2*u(i,j-1)+ul(j)*d1+(dt/2.0d0)*Qu(i,j);
188         ii = ii + 1;
189         ! Compute the row, up to the right hand wall ( 1 < i < M-1 )
190         do i = 2,M-2
191             a(ii) = -d1; b(ii) = 1.0d0 + 2.0d0*d1; c(ii) = -d1;
192             d(ii) = d2*u(i,j+1)+(1.0d0-2.0d0*d2)*u(i,j)+d2*u(i,j-1)+(dt/2.0d0)*Qu(i,j);
193             ii = ii + 1;
194         end do
195         ! Compute the right wall
196         a(ii) = -d1; b(ii) = 1.0d0 + 2.0d0*d1; c(ii) = 0.0d0;
197         d(ii) =
    ↪  d2*u(i,j+1)+(1.0d0-2.0d0*d2)*u(i,j)+d2*u(i,j-1)+ur(j)*d1+(dt/2.0d0)*Qu(i,j);
198         ii = ii + 1;
199     end do
200
201     ! Set top left corner i = 1, j = N
202     i = 1; j = N;
203     a(ii) = 0.0d0; b(ii) = 1.0d0+2.0d0*d1; c(ii) = -d1; d(ii) = (1.0d0 -
    ↪  3.0d0*d2)*u(i,j)+d2*u(i,j-1)+(dt/2.0d0)*Qu(i,j);
204     ii = ii + 1;
205
206     ! Set the top row ( j = N & 1 < i < M-2 )
207     do i = 2,M-2
208         a(ii) = -d1; b(ii) = 1.0d0 + 2.0d0*d1; c(ii) = -d1; d(ii) = (1.0d0 -
    ↪  3.0d0*d2)*u(i,j) + d2*u(i,j-1) + (dt/2.0d0)*Qu(i,j);
209         ii = ii + 1;
210     end do
211
212     ! Set top right corner i = M-1, j = N
213     i = M-1; j = N;
214     a(ii) = -d1; b(ii) = 1.0d0 + 2.0d0*d1; c(ii) = 0.0d0;
```

```
215     d(ii) = (1.0d0 - 3.0d0*d2)*u(i,j) + d2*u(i,j-1) + (dt/2.0d0)*Qu(i,j);
216
217     end subroutine setux
218
219     subroutine setuy(a,b,c,d,Qu,d1,d2,dt,ul,ur,u,N,M)
220     implicit none
221     integer :: i, j, jj, M, N
222     real (kind = 8) :: d1, d2, dt
223     real (kind = 8), dimension (1:(M-1)*N) :: a, b, c, d
224     real (kind = 8), dimension (0:N+1) :: ul, ur
225     real (kind = 8), dimension (0:M,0:N+1) :: u, Qu
226
227     jj = 1; i = 1; j = 1;
228     ! Set bottom left corner i = 1, j = 1
229     a(jj) = 0.0d0; b(jj) = 1.0d0 + 3.0d0*d2; c(jj) = -d2;
230     d(jj) = d1*u(i+1,j)+(1.0d0-2.0d0*d1)*u(i,j)+(dt/2.0d0)*Qu(i,j);
231     jj = jj + 1;
232
233     ! Fill up entries for first column ( i = 1, 1 < j < N )
234     do j = 2,N-1
235         a(jj) = -d2; b(jj) = 1.0d0 + 2.0d0*d2; c(jj) = -d2;
236         d(jj) = d1*u(i+1,j)+(1.0d0-2.0d0*d1)*u(i,j)+ul(j)*d1+(dt/2.0d0)*Qu(i,j);
237         jj = jj + 1;
238     end do
239
240     ! Set top right corner i = 1, j = N
241     i = 1; j = N;
242     a(jj) = -d2; b(jj) = 1.0+3.0d0*d2; c(jj) = 0.0d0;
243     d(jj) = d1*u(i+1,j)+(1.0d0-2.0d0*d1)*u(i,j)+(dt/2.0d0)*Qu(i,j);
244     jj = jj + 1;
245
246     ! Fill up 2nd column to M-2 column
247     do i = 2,M-2
248         ! Start with the bottom wall
249         j = 1
250         a(jj) = 0.0d0; b(jj) = 1.0d0 + 3.0d0*d2; c(jj) = -d2;
251         d(jj) = d1*u(i+1,j)+(1.0d0-2.0d0*d1)*u(i,j)+d1*u(i-1,j)+(dt/2.0d0)*Qu(i,j);
252         jj = jj + 1;
253         ! Compute the columns, up to the top wall ( 1 < j < N-1 )
254         do j = 2,N-1
255             a(jj) = -d2; b(jj) = 1.0d0+2.0d0*d2; c(jj) = -d2;
256             d(jj) = d1*u(i+1,j)+(1.0d0-2.0d0*d1)*u(i,j)+d1*u(i-1,j)+(dt/2.0d0)*Qu(i,j);
257             jj = jj + 1;
258         end do
259         ! Finish with the top wall
260         a(jj) = -d2; b(jj) = 1.0d0+3.0d0*d2; c(jj) = 0.0d0;
261         d(jj) = d1*u(i+1,j)+(1.0d0-2.0d0*d1)*u(i,j)+d1*u(i-1,j)+(dt/2.0d0)*Qu(i,j)
262         jj = jj + 1;
263     end do
264
265     ! Set bottom right corner i = M-1, j = 1
266     i = M-1; j = 1;
267     a(jj) = 0.0d0; b(jj) = 1.0d0 + 3.0d0*d2; c(jj) = -d2;
268     d(jj) = (1.0d0-2.0d0*d1)*u(i,j)+d1*u(i-1,j)+(dt/2.0d0)*Qu(i,j);
```

```
269     jj = jj + 1;
270
271     ! Fill the final column i = M-1, 1 < j < N
272     do j = 2,N-1
273         a(jj) = -d2; b(jj) = 1.0d0+2.0d0*d2; c(jj) = -d2;
274         d(jj) = ur(j)*d1+(1.0d0-2.0d0*d1)*u(i,j)+d1*u(i-1,j)+(dt/2.0d0)*Qu(i,j);
275         jj = jj + 1;
276     end do
277
278     ! Set top right corner i = M-1, j = N
279     a(jj) = -d2; b(jj) = 1.0d0+3.0d0*d2; c(jj) = 0.0d0;
280     d(jj) = (1.0d0-2.0d0*d1)*u(i,j) + d1*u(i-1,j)+(dt/2.0d0)*Qu(i,j);
281
282     end subroutine setuy
283
284     subroutine setvx(a,b,c,d,Qv,d1,d2,dt,vt,v,N,M,x)
285     implicit none
286     integer :: i, j, ii, N, M
287     real (kind = 8) :: d1, d2, dt
288     real (kind = 8), dimension(1:(N-1)*M) :: a, b, c, d
289     real (kind = 8), dimension(0:M+1) :: vt, x
290     real (kind = 8), dimension(0:M+1,0:N) :: v, Qv
291
292     ii = 1; i = 1; j = 1;
293     ! Set bottom left corner i = 1, j = 1
294     a(ii) = 0.0d0; b(ii) = 1.0d0+3.0d0*d1; c(ii) = -d1;
295     d(ii) = d2*v(i,j+1)+(1.0d0-2.0d0*d2)*v(i,j)+(dt/2.0d0)*Qv(i,j);
296     ii = ii + 1;
297
298     ! Fill up entries on the bottom row ( j = 1 & 1 < i < M-1 )
299     do i = 2,M-1
300         if ((x(i)>1.5).and.(x(i)<2.5))then
301             a(ii) = -d1; b(ii) = 1.0d0 + 2.0d0*d1; c(ii) = -d1;
302             d(ii) =
    ↪  (2.0d0/3.0d0)*d2*v(i,j+1)+(1.0d0-(2.0d0/3.0d0)*d2)*v(i,j)+(dt/2.0d0)*Qv(i,j);
303             ii = ii + 1;
304         else
305         a(ii) = -d1; b(ii) = 1.0d0+2.0d0*d1; c(ii) = -d1;
306         d(ii) = d2*v(i,j+1)+(1.0d0-2.0d0*d2)*v(i,j)+(dt/2.0d0)*Qv(i,j);
307         ii = ii + 1;
308         end if
309     end do
310
311     ! Set bottom right corner i = M, j = 1
312     i = M; j = 1;
313     a(ii) = -d1; b(ii) = 1.0d0+3.0d0*d1; c(ii) = 0.0d0;
314     d(ii) = d2*v(i,j+1)+(1.0d0-2.0d0*d2)*v(i,j)+(dt/2.0d0)*Qv(i,j);
315     ii = ii + 1;
316
317     ! Fill up entries from the 2nd row to the N-2 row
318     do j = 2,N-2
319         ! Start with the left wall
320         i = 1;
321         a(ii) = 0.0d0; b(ii) = 1.0d0+3.0d0*d1; c(ii) = -d1;
```

```
322         d(ii) = d2*v(i,j+1)+(1.0d0-2.0d0*d2)*v(i,j)+d2*v(i,j-1)+(dt/2.0d0)*Qv(i,j);
323         ii = ii + 1;
324         ! Compute the row, up to right hand wall ( 1 < i < M )
325         do i = 2,M-1
326             a(ii) = -d1; b(ii) = 1.0d0+2.0d0*d1; c(ii) = -d1;
327             d(ii) = d2*v(i,j+1)+(1.0d0-2.0d0*d2)*v(i,j)+d2*v(i,j-1)+(dt/2.0d0)*Qv(i,j);
328             ii = ii + 1;
329         end do
330         ! Compute the right wall
331         a(ii) = -d1; b(ii) = 1.0d0+3.0d0*d1; c(ii) = 0.0d0;
332         d(ii) = d2*v(i,j+1)+(1.0d0-2.0d0*d2)*v(i,j)+d2*v(i,j-1)+(dt/2.0d0)*Qv(i,j);
333         ii = ii + 1;
334     end do
335 
336     ! Set top left corner i = 1, j = N-1
337     i = 1; j = N-1;
338     a(ii) = 0.0d0; b(ii) = 1.0d0+3.0d0*d1; c(ii) = -d1;
339     d(ii) = (1.0d0-2.0d0*d2)*v(i,j)+d2*v(i,j-1)+(dt/2.0d0)*Qv(i,j);
340     ii = ii + 1;
341 
342     ! Fill up entries on the top row ( j = N-1, 1 < i < M-1 )
343     do i = 2,M-1
344         a(ii) = -d1; b(ii) = 1.0d0+2.0d0*d1; c(ii) = -d1;
345         d(ii) = vt(i)*d2+(1.0d0-2.0d0*d2)*v(i,j)+d2*v(i,j-1)+(dt/2.0d0)*Qv(i,j);
346         ii = ii + 1;
347     end do
348 
349     ! Set top right corner i = M, j = N-1
350     i = M;
351     a(ii) = -d1; b(ii) = 1.0d0+3.0d0*d1; c(ii) = 0.0d0;
352     d(ii) = (1.0d0-2.0d0*d2)*v(i,j) + d2*v(i,j-1)+(dt/2.0d0)*Qv(i,j);
353 
354     end subroutine setvx
355 
356     subroutine setvy(a,b,c,d,Qv,d1,d2,dt,vt,v,N,M,x)
357     implicit none
358     integer :: i, j, jj, N, M
359     real (kind = 8) :: d1, d2, dt
360     real (kind = 8), dimension(1:(N-1)*M) :: a, b, c, d
361     real (kind = 8), dimension(0:M+1) :: vt, x
362     real (kind = 8), dimension(0:M+1,0:N) :: v, Qv
363 
364     jj = 1; i = 1; j = 1;
365     ! Set bottom left corner i = 1, j = 1
366     a(jj) = 0.0d0; b(jj) = 1.0d0+2.0d0*d2; c(jj) = -d2;
367     d(jj) = d1*v(i+1,j)+(1.0d0-3.0d0*d1)*v(i,j)+(dt/2.0d0)*Qv(i,j)
368     jj = jj + 1;
369 
370     ! Fill up entries for first column ( i = 1, 1 < j < N-1 )
371     do j = 2,N-2
372         a(jj) = -d2; b(jj) = 1.0d0+2.0d0*d2; c(jj) = -d2;
373         d(jj) = d1*v(i+1,j)+(1.0d0-3.0d0*d1)*v(i,j)+(dt/2.0d0)*Qv(i,j)
374         jj = jj + 1
375     end do
```

```
376
377     ! Set top left corner i = 1, j = N-1
378     j = N-1;
379     a(jj) = -d2; b(jj) = 1.0d0+2.0d0*d2; c(jj) = 0.0d0;
380     d(jj) = d1*v(i+1,j)+(1.0d0-3.0d0*d1)*v(i,j)+(dt/2.0d0)*Qv(i,j)
381     jj = jj + 1
382
383     ! Fill the 2nd column to the M-1 column
384     do i = 2,M-1
385         ! Start with the bottom wall
386         j = 1;
387         if ((x(i)>1.5d0).and.(x(i)<2.5d0)) then
388             a(jj) = 0.0d0; b(jj) = 1.0d0+(2.0d0/3.0d0)*d2; c(jj) = -(2.0d0/3.0d0)*d2;
389             d(jj) = d1*v(i+1,j)+(1.0d0-2.0d0*d1)*v(i,j)+d1*v(i-1,j)+(dt/2.0d0)*Qv(i,j);
390         else
391             a(jj) = 0.0d0; b(jj) = 1.0d0+2.0d0*d2; c(jj) = -d2;
392             d(jj) = d1*v(i+1,j)+(1.0d0-2.0d0*d1)*v(i,j)+d1*v(i-1,j)+(dt/2.0d0)*Qv(i,j);
393         end if
394         jj = jj + 1;
395         ! Fill the interior rows 1 < j < N-1
396         do j = 2,N-2
397             a(jj) = -d2; b(jj) = 1.0d0+2.0d0*d2; c(jj) = -d2;
398             d(jj) = d1*v(i+1,j)+(1.0d0-2.0d0*d1)*v(i,j)+d1*v(i-1,j)+(dt/2.0d0)*Qv(i,j);
399             jj = jj + 1;
400         end do
401         ! Finish with the top wall
402         a(jj) = -d2; b(jj) = 1.0d0+2.0d0*d2; c(jj) = 0.0d0;
403         d(jj) = d1*v(i+1,j)+(1.0d0-2.0d0*d1)*v(i,j)+d1*v(i-1,j) +
    ↪  vt(i)*d2+(dt/2.0d0)*Qv(i,j);
404         jj = jj + 1;
405     end do
406
407     ! Set bottom right corner i = M, j = 1
408     i = M; j = 1;
409     a(jj) = 0.0d0; b(jj) = 1.0d0+2.0d0*d2; c(jj) = -d2;
410     d(jj) = (1.0d0-3.0d0*d1)*v(i,j)+d1*v(i-1,j)+(dt/2.0d0)*Qv(i,j);
411     jj = jj + 1;
412
413     ! Set final column i = M, 1 < j < N-1
414     do j = 2,N-2
415         a(jj) = -d2; b(jj) = 1.0d0+2.0d0*d2; c(jj) = -d2;
416         d(jj) = (1.0d0-3.0d0*d1)*v(i,j)+d1*v(i-1,j)+(dt/2.0d0)*Qv(i,j);
417         jj = jj + 1;
418     end do
419
420     ! Set top right corner i = M, j = N-1
421     j = N-1;
422     a(jj) = -d2; b(jj) = 1.0d0+2.0d0*d2; c(jj) = 0.0d0;
423     d(jj) = (1.0d0-3.0d0*d1)*v(i,j)+d1*v(i-1,j)+(dt/2.0d0)*Qv(i,j);
424
425     end subroutine setvy
426
427     subroutine setYx(a,b,c,d,Qy,d1,d2,dt,Y,N,M,yy,xy,concentration,nl,nr,nt,loc,width)
428     implicit none
```

```
  integer :: i, j, ii, M, N, nl, nr, nt, dummyn
  real (kind = 8) :: d1, d2, dt, width
  real (kind = 8), dimension(1:M*N) :: a, b, c, d
  real (kind = 8), dimension(-2:M+3,-2:N+3) :: Y, Qy
  real (kind = 8), dimension(-2:N+3) :: yy
  real (kind = 8), dimension(-2:M+3) :: xy
  real (kind = 8), dimension(1:nl+nr+nt) :: concentration, loc
  logical :: noinlet
  ii = 1;
  ! Set bottom left corner i = 1, j = 1
  i = 1; j = 1;
  a(ii) = 0.0d0; b(ii) = 1.0d0 + d1; c(ii) = -d1;
  d(ii) = d2*Y(i,j+1)+(1.0d0-d2)*Y(i,j)+(dt/2.0d0)*Qy(i,j);
  ii = ii + 1;

  ! Set first row j = 1, 1 < i < M
  do i = 2,M-1
      a(ii) = -d1; b(ii) = 1.0d0+2.0d0*d1; c(ii) = -d1;
      d(ii) = d2*Y(i,j+1)+(1.0d0-d2)*Y(i,j)+(dt/2.0d0)*Qy(i,j);
      ii = ii + 1;
  end do

  ! Set the bottom right corner j = 1, i = M
  i = M; j = 1;
  a(ii) = -d1; b(ii) = 1.0d0+d1; c(ii) = 0.0d0;
  d(ii) = d2*Y(i,j+1)+(1.0d0-d2)*Y(i,j)+(dt/2.0d0)*Qy(i,j);
  ii = ii + 1;

  noinlet = .TRUE.
  ! Fill up entries from the 2nd row to the N-1 row
  do j = 2,N-1
      ! Compute the left wall
      i = 1;
      do dummyn = 1,nl
        call
↪ inletCNYLx(a,b,c,d,d1,d2,dt,yy,i,j,ii,concentration(dummyn),loc(dummyn),width,N,M,noinlet,Y,Qy)
      end do
      if (noinlet) then
        a(ii) = 0.0d0; b(ii) = 1.0d0 + d1; c(ii) = -d1;
        d(ii) = d2*Y(i,j+1)+(1.0d0-2.0d0*d2)*Y(i,j)+d2*Y(i,j-1)+(dt/2.0d0)*Qy(i,j);
      end if
      ii = ii + 1;
      ! Compute the row up to the right hand wall ( 1 < i < M )
      do i = 2,M-1
          a(ii) = -d1; b(ii) = 1.0d0+2.0d0*d1; c(ii) = -d1;
          d(ii) = d2*Y(i,j+1)+(1.0d0-2.0d0*d2)*Y(i,j)+d2*Y(i,j-1)+(dt/2.0d0)*Qy(i,j);
          ii = ii + 1;
      end do
      ! Compute the right wall
      do dummyn = nl+nt+1,nl+nr+nt
        call
↪ inletCNYRx(a,b,c,d,d1,d2,dt,yy,i,j,ii,concentration(dummyn),loc(dummyn),width,N,M,noinlet,Y,Qy)
      end do
      if (noinlet) then
```

```
481         a(ii) = -d1; b(ii) = 1.0d0+d1; c(ii) = 0.0d0;
482         d(ii) = d2*Y(i,j+1)+(1.0d0-2.0d0*d2)*Y(i,j)+d2*Y(i,j-1)+(dt/2.0d0)*Qy(i,j);
483       end if
484       ii = ii + 1;
485   end do
486
487   ! Set top left corner i = 1, j = N
488   i = 1; j = N;
489   a(ii) = 0.0d0; b(ii) = 1.0d0+d1; c(ii) = -d1;
490   d(ii) = (1.0d0-d2)*Y(i,j)+d2*Y(i,j-1)+(dt/2.0d0)*Qy(i,j);
491   ii = ii + 1;
492
493   ! Set last row j = N, 1 < i < M
494   do i = 2,M-1
495     do dummyn = nl+1,nl+nt
496       call
    ↪  inletCNYTx(a,b,c,d,d1,d2,dt,xy,i,j,ii,concentration(dummyn),loc(dummyn),width,N,M,noinlet,Y,Qy)
497     end do
498     if (noinlet) then
499       a(ii) = -d1; b(ii) = 1.0d0+2.0d0*d1; c(ii) = -d1;
500       d(ii) = (1.0d0-d2)*Y(i,j)+d2*Y(i,j-1)+(dt/2.0d0)*Qy(i,j);
501     end if
502       ii = ii + 1;
503   end do
504
505   ! Set top right corner i = M, j = N
506   i = M; j = N
507   a(ii) = -d1; b(ii) = 1.0d0+d1; c(ii) = 0.0d0;
508   d(ii) = (1.0d0-d2)*Y(i,j)+d2*Y(i,j-1)+(dt/2.0d0)*Qy(i,j);
509
510   end subroutine setYx
511
512   subroutine
    ↪  inletCNYLx(a,b,c,d,d1,d2,dt,yy,i,j,ii,concentration,loc,width,N,M,noinlet,Y,Qy)
513     implicit none
514     integer :: i, j, ii, N, M
515     real (kind = 8) :: d1, d2, dt, loc, width, concentration
516     real (kind = 8), dimension(1:M*N) :: a, b, c, d
517     real (kind = 8), dimension(-2:M+3,-2:N+3) :: Y, Qy
518     real (kind = 8), dimension(-2:N+3) :: yy
519     logical :: noinlet
520     if ((yy(j)>loc-(width/2.0d0)).and.(yy(j)<loc+(width/2.0d0))) then
521         a(ii) = 0.0d0; b(ii) = 1.0d0+3.0d0*d1; c(ii) = -d1;
522         d(ii) =
    ↪  d2*Y(i,j+1)+(1.0d0-2.0d0*d2)*Y(i,j)+d2*Y(i,j-1)+2.0d0*concentration*d1+(dt/2.0d0)*Qy(i,j)
523         noinlet = .FALSE.
524     else
525         noinlet = .TRUE.
526     end if
527   end subroutine inletCNYLx
528
529   subroutine
    ↪  inletCNYRx(a,b,c,d,d1,d2,dt,yy,i,j,ii,concentration,loc,width,N,M,noinlet,Y,Qy)
530     implicit none
```

```
531       integer :: i, j, ii, N, M
532       real (kind = 8) :: d1, d2, dt, loc, width, concentration
533       real (kind = 8), dimension(1:M*N) :: a, b, c, d
534       real (kind = 8), dimension(-2:M+3,-2:N+3) :: Y, Qy
535       real (kind = 8), dimension(-2:N+3) :: yy
536       logical :: noinlet
537       if ((yy(j)>loc-(width/2.0d0)).and.(yy(j)<loc+(width/2.0d0))) then
538         a(ii) = -d1; b(ii) = 1.0d0+3.0d0*d1; c(ii) = 0.0d0;
539         d(ii) =
    ↪  d2*Y(i,j+1)+(1.0d0-2.0d0*d2)*Y(i,j)+d2*Y(i,j-1)+2.0d0*concentration*d1+(dt/2.0d0)*Qy(i,j);
540         noinlet = .FALSE.
541       else
542         noinlet = .TRUE.
543       end if
544     end subroutine inletCNYRx
545
546     subroutine
    ↪  inletCNYTx(a,b,c,d,d1,d2,dt,x,i,j,ii,concentration,loc,width,N,M,noinlet,Y,Qy)
547       implicit none
548       integer :: i, j, ii, N, M
549       real (kind = 8) :: d1, d2, dt, loc, width, concentration
550       real (kind = 8), dimension(1:M*N) :: a, b, c, d
551       real (kind = 8), dimension(-2:M+3,-2:N+3) :: Y, Qy
552       real (kind = 8), dimension(-2:M+3) :: x
553       logical :: noinlet
554       if ((x(i)>loc-(width/2.0d0)).and.(x(i)<loc+(width/2.0d0))) then
555         a(ii) = -d1; b(ii) = 1.0d0+2.0d0*d1; c(ii) = -d1;
556         d(ii) = 2.0d0*concentration*d2+
    ↪  (1.0d0-3.0d0*d2)*Y(i,j)+d2*Y(i,j-1)+(dt/2.0d0)*Qy(i,j);
557         noinlet = .FALSE.
558       else
559         noinlet = .TRUE.
560       end if
561     end subroutine inletCNYTx
562
563     subroutine setYy(a,b,c,d,Qy,d1,d2,dt,Y,N,M,yy,xy,concentration,nl,nr,nt,loc,width)
564     implicit none
565     integer :: i, j, jj, N, M, nl, nr, nt, dummyn
566     real (kind = 8) :: d1, d2, dt, width
567     real (kind = 8), dimension(1:M*N) :: a, b, c, d
568     real (kind = 8), dimension(-2:M+3,-2:N+3) :: Y, Qy
569     real (kind = 8), dimension(-2:N+3) :: yy
570     real (kind = 8), dimension(-2:M+3) :: xy
571     real (kind = 8), dimension(1:nl+nr+nt) :: concentration, loc
572     logical :: noinlet
573
574     jj = 1; i = 1; j = 1;
575     ! Set bottom left corner i = 1, j = 1
576     a(jj) = 0.0d0; b(jj) = 1.0d0+d2; c(jj) = -d2;
577     d(jj) = d1*Y(i+1,j)+(1.0d0-d1)*Y(i,j)+(dt/2.0d0)*Qy(i,j);
578     jj = jj + 1;
579
580     noinlet = .TRUE.;
581     ! Set first column i = 1, 1 < j < N
```

```
582   do j = 2,N-1
583     do dummyn = 1,nl
584       call
    ↪  inletCNYLy(a,b,c,d,d1,d2,dt,yy,i,j,jj,concentration(dummyn),loc(dummyn),width,N,M,noinlet,Y,Qy)
585     end do
586     if (noinlet) then
587       a(jj) = -d2; b(jj) = 1.0d0+2.0d0*d2; c(jj) = -d2;
588       d(jj) = d1*Y(i+1,j)+(1.0d0-d1)*Y(i,j)+(dt/2.0d0)*Qy(i,j);
589     end if
590       jj = jj + 1;
591   end do
592
593   ! Set the top left corner i = 1, j = N
594   a(jj) = -d2; b(jj) = 1.0d0+d2; c(jj) = 0.0d0;
595   d(jj) = d1*Y(i+1,j)+(1.0d0-d1)*Y(i,j)+(dt/2.0d0)*Qy(i,j);
596   jj = jj + 1;
597
598   ! Fill up entries from the 2nd column to the M-1 column
599   do i = 2,M-1
600       ! Compute the bottom wall
601       j = 1;
602       a(jj) = 0.0d0; b(jj) = 1.0d0+d2; c(jj) = -d2;
603       d(jj) = d1*Y(i+1,j)+(1.0d0-2.0d0*d1)*Y(i,j)+d1*Y(i-1,j)+(dt/2.0d0)*Qy(i,j);
604       jj = jj + 1;
605       ! Compute the column up to the top wall 1 < j < N
606       do j = 2,N-1
607           a(jj) = -d2; b(jj) = 1.0d0+2.0d0*d2; c(jj) = -d2;
608           d(jj) = d1*Y(i+1,j)+(1.0d0-2.0d0*d1)*Y(i,j)+d1*Y(i-1,j)+(dt/2.0d0)*Qy(i,j);
609           jj = jj + 1;
610       end do
611       ! Compute the top wall
612       do dummyn = nl+1,nl+nt
613         call
    ↪  inletCNYTy(a,b,c,d,d1,d2,dt,xy,i,j,jj,concentration(dummyn),loc(dummyn),width,N,M,noinlet,Y,Qy)
614       end do
615       if (noinlet) then
616         a(jj) = -d2; b(jj) = 1.0d0+d2; c(jj) = 0.0d0;
617         d(jj) = d1*Y(i+1,j)+(1.0d0-2.0d0*d1)*Y(i,j)+d1*Y(i-1,j)+(dt/2.0d0)*Qy(i,j);
618       end if
619       jj = jj + 1;
620   end do
621
622   ! Set the bottom right corner i = M, j = 1
623   i = M; j = 1;
624   a(jj) = 0.0d0; b(jj) = 1.0d0+d2; c(jj) = -d2;
625   d(jj) = (1.0d0-d1)*Y(i,j)+d1*Y(i-1,j)+(dt/2.0d0)*Qy(i,j);
626   jj = jj + 1;
627
628   ! Set the final column i = M,  1 < j < N
629   do j = 2,N-1
630     do dummyn = nl+nt+1,nl+nt+nr
631       call
    ↪  inletCNYRy(a,b,c,d,d1,d2,dt,yy,i,j,jj,concentration(dummyn),loc(dummyn),width,N,M,noinlet,Y,Qy)
632       ! print *, dummyn, concentration(dummyn), loc(dummyn), xy(i), yy(j)
```

```
633       end do
634       if (noinlet) then
635         a(jj) = -d2; b(jj) = 1.0d0+2.0d0*d2; c(jj) = -d2;
636         d(jj) = (1.0d0-d1)*Y(i,j)+d1*Y(i-1,j)+(dt/2.0d0)*Qy(i,j);
637       end if
638         jj = jj + 1;
639     end do
640
641     ! Set the top right corner i = M, j = N
642     a(jj) = -d2; b(jj) = 1.0d0+d2; c(jj) = 0.0d0;
643     d(jj) = (1.0d0-d1)*Y(i,j)+d1*Y(i-1,j)+(dt/2.0d0)*Qy(i,j);
644
645     end subroutine setYy
646
647     subroutine
    ↪  inletCNYLy(a,b,c,d,d1,d2,dt,yy,i,j,jj,concentration,loc,width,N,M,noinlet,Y,Qy)
648       implicit none
649       integer :: i, j, jj, N, M
650       real (kind = 8) :: d1, d2, dt, loc, width, concentration
651       real (kind = 8), dimension(1:M*N) :: a, b, c, d
652       real (kind = 8), dimension(-2:M+3,-2:N+3) :: Y, Qy
653       real (kind = 8), dimension(-2:N+3) :: yy
654       logical :: noinlet
655       if ((yy(j)>loc-(width/2.0d0)).and.(yy(j)<loc+(width/2.0d0))) then
656         a(jj) = -d2; b(jj) = 1.0d0+2.0d0*d2; c(jj) = -d2;
657         d(jj) =
    ↪  d1*Y(i+1,j)+(1.0d0-3.0d0*d1)*Y(i,j)+2.0d0*concentration*d1+(dt/2.0d0)*Qy(i,j);
658           noinlet = .FALSE.
659       else
660           noinlet = .TRUE.
661       end if
662     end subroutine inletCNYLy
663
664     subroutine
    ↪  inletCNYRy(a,b,c,d,d1,d2,dt,yy,i,j,jj,concentration,loc,width,N,M,noinlet,Y,Qy)
665       implicit none
666       integer :: i, j, jj, N, M
667       real (kind = 8) :: d1, d2, dt, loc, width, concentration
668       real (kind = 8), dimension(1:M*N) :: a, b, c, d
669       real (kind = 8), dimension(-2:M+3,-2:N+3) :: Y, Qy
670       real (kind = 8), dimension(-2:N+3) :: yy
671       logical :: noinlet
672       if ((yy(j)>loc-(width/2.0d0)).and.(yy(j)<loc+(width/2.0d0))) then
673         a(jj) = -d2; b(jj) = 1.0d0+2.0d0*d2; c(jj) = -d2;
674         d(jj) =
    ↪  2.0d0*concentration*d1+(1.0d0-3.0d0*d1)*Y(i,j)+d1*Y(i-1,j)+(dt/2.0d0)*Qy(i,j);
675         noinlet = .FALSE.
676       else
677         noinlet = .TRUE.
678       end if
679     end subroutine inletCNYRy
680
681     subroutine
    ↪  inletCNYTy(a,b,c,d,d1,d2,dt,x,i,j,jj,concentration,loc,width,N,M,noinlet,Y,Qy)
```

```
    implicit none
    integer :: i, j, jj, N, M
    real (kind = 8) :: d1, d2, dt, loc, width, concentration
    real (kind = 8), dimension(1:M*N) :: a, b, c, d
    real (kind = 8), dimension(-2:M+3,-2:N+3) :: Y, Qy
    real (kind = 8), dimension(-2:M+3) :: x
    logical :: noinlet
    if ((x(i)>loc-(width/2.0d0)).and.(x(i)<loc+(width/2.0d0))) then
      a(jj) = -d2; b(jj) = 1.0d0+3.0d0*d2; c(jj) = 0.0d0;
      d(jj) = 2.0d0*concentration*d2 +
↪  d1*Y(i+1,j)+(1.0d0-2.0d0*d1)*Y(i,j)+d1*Y(i-1,j)+(dt/2.0d0)*Qy(i,j);
      noinlet = .FALSE.
    else
      noinlet = .TRUE.
    end if
  end subroutine inletCNYTy

end module CN
```

"probe.f95":

```
module probe


  implicit none
  private
  public :: findindex, findindexY, interpolate3D, interpolate3DY

contains

  subroutine findindex(x,y,i,j,targetx,targety,sizex,sizey)
  implicit none
  integer :: i, j, sizex, sizey
  real (kind = 8) :: targetx, targety
  real (kind = 8), dimension(0:sizex) :: x
  real (kind = 8), dimension(0:sizey) :: y

  do i = 0,sizex
      if (x(i)>targetx) then
          EXIT
      end if
  end do

  do j = 0,sizey
      if (y(j)>targety) then
          EXIT
      end if
  end do

  end subroutine findindex

  subroutine findindexY(x,y,i,j,targetx,targety,sizex,sizey)
  implicit none
  integer :: i, j, sizex, sizey
  real (kind = 8) :: targetx, targety
```

```
35    real (kind = 8), dimension(-2:sizex) :: x
36    real (kind = 8), dimension(-2:sizey) :: y
37
38    do i = 0,sizex
39        if (x(i)>targetx) then
40            EXIT
41        end if
42    end do
43
44    do j = 0,sizey
45        if (y(j)>targety) then
46            EXIT
47        end if
48    end do
49
50    end subroutine findindexY
51
52    subroutine interpolate3D(x,y,i,j,targetx,targety,sizex,sizey,probe,z)
53    implicit none
54    integer :: i, j, sizex, sizey
55    real (kind = 8) :: targetx, targety, probe, p2, p1
56    real (kind = 8), dimension(0:sizex) :: x
57    real (kind = 8), dimension(0:sizey) :: y
58    real (kind = 8), dimension(0:sizex,0:sizey) :: z
59
60    if (targetx == x(i-1)) then
61        p2 = z(i-1,j); p1 = z(i-1,j-1)
62    else
63        ! interpolate at y2
64        p2 = (z(i,j)-z(i-1,j))*(targetx-x(i-1))/(x(i)-x(i-1))+z(i-1,j)
65
66        ! interpolate at y1
67        p1 = (z(i,j-1)-z(i-1,j-1))*(targetx-x(i-1))/(x(i)-x(i-1))+z(i-1,j-1)
68
69    end if
70
71    if (targety == y(j-1)) then
72        probe = p1
73    else
74        ! interpolate between p1 and p2
75        probe = (p2-p1)*(targety-y(j-1))/(y(j)-y(j-1))+p1
76    end if
77
78    if ((targetx == x(i-1)).and.(targety == y(j-1))) then
79        probe = z(i-1,j-1)
80    end if
81
82    end subroutine interpolate3D
83
84    subroutine interpolate3DY(x,y,i,j,targetx,targety,sizex,sizey,probe,z)
85    implicit none
86    integer :: i, j, sizex, sizey
87    real (kind = 8) :: targetx, targety, probe, p2, p1
88    real (kind = 8), dimension(-2:sizex) :: x
```

```
  real (kind = 8), dimension(-2:sizey) :: y
  real (kind = 8), dimension(-2:sizex,-2:sizey) :: z

  if (targetx == x(i-1)) then
      p2 = z(i-1,j); p1 = z(i-1,j-1)
  else
      ! interpolate at y2
      p2 = (z(i,j)-z(i-1,j))*(targetx-x(i-1))/(x(i)-x(i-1))+z(i-1,j)

      ! interpolate at y1
      p1 = (z(i,j-1)-z(i-1,j-1))*(targetx-x(i-1))/(x(i)-x(i-1))+z(i-1,j-1)

  end if

  if (targety == y(j-1)) then
      probe = p1
  else
      ! interpolate between p1 and p2
      probe = (p2-p1)*(targety-y(j-1))/(y(j)-y(j-1))+p1
  end if

  if ((targetx == x(i-1)).and.(targety == y(j-1))) then
      probe = z(i-1,j-1)
  end if

  end subroutine interpolate3DY

end module probe
```

"swissarmy.f95":

```
module swissarmy

  implicit none

  private
  public :: popu, iboundaryu, popv, iboundaryv, popY, iboundaryY, popPhi, ICu, ICv, ICY,
 ↪  boundaryu, boundaryv, boundaryY, &
  calcR, diagnostics, savedata, calcS, Volflow, Volin

contains

  subroutine popu(xu,yu,deltax,deltay,x0,y0,N,M)
    implicit none
    integer :: i, j, N, M
    real (kind = 8) :: deltax, deltay, x0, y0
    real (kind = 8), dimension(0:M) :: xu
    real (kind = 8), dimension(0:N+1) :: yu
    do i = 0,M
        xu(i) = x0 + dble(i)*deltax;
    end do

    do j = 0,N+1
        yu(j) = y0 + (dble(j)-0.5d0)*deltay;
    end do
```

```
24    end subroutine popu
25
26    subroutine popv(xv,yv,deltax,deltay,x0,y0,N,M)
27      implicit none
28      integer :: i, j, N, M
29      real (kind = 8) :: deltax, deltay, x0, y0
30      real (kind = 8), dimension(0:M+1) :: xv
31      real (kind = 8), dimension(0:N) :: yv
32      do i = 0,M+1
33          xv(i) = x0 + (dble(i)-0.5d0)*deltax;
34      end do
35
36      do j = 0,N
37          yv(j) = y0 + dble(j)*deltay;
38      end do
39    end subroutine popv
40
41    subroutine popY(xy,yy,deltax,deltay,x0,y0,N,M)
42      implicit none
43      integer :: i, j, N, M
44      real (kind = 8) :: deltax, deltay, x0, y0
45      real (kind = 8), dimension(-2:M+3) :: xy
46      real (kind = 8), dimension(-2:N+3) :: yy
47      do i = -2,M+3
48          xy(i) = x0 + (dble(i)-0.50d0)*deltax
49      end do
50
51      do j = -2,N+3
52          yy(j) = y0 + (dble(j)-0.50d0)*deltay
53      end do
54    end subroutine popY
55
56    subroutine popPhi(xphi,yphi,x0,y0,deltax,deltay,N,M)
57      integer :: i, j, N, M
58      real (kind = 8) :: x0, y0, deltax, deltay
59      real (kind = 8), dimension(0:M+1) :: xphi
60      real (kind = 8), dimension(0:N+1) :: yphi
61      do i = 0,M+1
62          xphi(i) = x0 + (dble(i)-0.5d0)*deltax;
63      end do
64
65      do j = 0,N+1
66          yphi(j) = y0 + (dble(j)-0.5d0)*deltay;
67      end do
68    end subroutine popPhi
69
70    subroutine iboundaryu(u,ual,uar,yu,ul,ur,ub,ut,nl,nr,locleft,locright,width,N,M)
71      implicit none
72      integer :: j, N, M, nl, nr
73      real (kind = 8) :: width
74      real (kind = 8), dimension(0:N+1) :: yu, ul, ur
75      real (kind = 8), dimension(0:M) :: ub, ut
76      real (kind = 8), dimension(0:M,0:N+1) :: u
77      real (kind = 8), dimension(1:nl) :: ual, locleft
```

```
78      real (kind = 8), dimension(1:nr) :: uar, locright
79
80      ul(:) = 0.0d0; ur(:) = 0.0d0;
81
82      do j = 1,nl
83        call addinletu(ul,yu,ual(j),width,locleft(j),N)
84      end do
85
86      do j = 1,nr
87        call addinletu(ur,yu,uar(j),width,locright(j),N)
88      end do
89
90      ! set all cells; left cells; right cells; top cells; bottom cells;
91      u(:,:) = 0.0d0; u(0,:) = ul; u(M,:) = ur; u(:,N+1) = ut; u(:,0) = ub;
92    end subroutine iboundaryu
93
94    subroutine addinletu(us,yu,magnitude,width,loc,N)
95      implicit none
96      integer :: j, N
97      real (kind = 8) :: magnitude, width, loc, ad, kd, hd, wd
98      real (kind = 8), dimension(0:N+1) :: yu, us
99
100     do j = 0,N+1
101       ! Set values for inlet 1 on the left hand wall
102       if ( (yu(j) > loc-width/2.0d0).and.(yu(j) < loc+width/2.0d0) ) then
103           ! Set dummy variables for the vertex form of a parabola
104           kd = 1.5d0*magnitude; hd = loc; wd = width; ad = -kd/(0.5d0*wd)**2;
105           ! Vertex form of parabola y = a(x-h)^2 + k
106           us(j) = ad*(yu(j)-hd)**2+kd;
107       end if
108     end do
109   end subroutine addinletu
110
111   subroutine iboundaryv(v,vat,xv,vt,nt,loctop,width,N,M)
112     implicit none
113     integer :: i, N, M, nt
114     real (kind = 8) :: width
115     real (kind = 8), dimension(0:M+1) :: xv, vt
116     real (kind = 8), dimension(0:M+1,0:N) :: v
117     real (kind = 8), dimension(1:nt) :: vat, loctop
118
119     vt(:) = 0.0d0
120
121     do i = 1,nt
122       call addinletu(vt,xv,vat(i),width,loctop(i),M)
123     end do
124
125     ! Set all cells; top cells
126     v(:,:) = 0.0d0; v(:,N) = vt;
127   end subroutine iboundaryv
128
129   ! subroutine addinletv(vs,xv,magnitude,)
130   !
131   ! end subroutine addinletv
```

```
132
133   subroutine iboundaryY(Y,Yinletconditions,xy,yy,nl,nt,nr,loc,width,N,M)
134     implicit none
135     integer :: j, N, M, nl, nt, nr
136     real (kind = 8) :: width
137     real (kind = 8), dimension(-2:N+3) :: yy, Yt
138     real (kind = 8), dimension(-2:M+3) :: xy, Yl, Yr
139     real (kind = 8), dimension(-2:M+3,-2:N+3) :: Y
140     real (kind = 8), dimension(1:nl+nt+nr) :: Yinletconditions, loc
141
142     ! Set intial condition for domain
143     Y(:,:) = 0.0d0;
144     ! Add inlet conditions for Y on each inlet
145     do j = 1,nl+nt+nr
146       if (j <= nl) then
147         call addinletY(Yl,yy,Yinletconditions(j),loc(j),width,N)
148         Y(0,:) = Yl(:)-Y(1,:)
149         Y(-1,:) = Yl(:)-Y(2,:)
150         Y(-2,:) = Yl(:)-Y(3,:)
151       else if(j <= nt+nl) then
152         call addinletY(Yt,xy,Yinletconditions(j),loc(j),width,M)
153         Y(:,N+1) = Yt(:)-Y(:,N)
154         Y(:,N+2) = Yt(:)-Y(:,N-1)
155         Y(:,N+3) = Yt(:)-Y(:,N-2)
156       else
157         call addinletY(Yr,yy,Yinletconditions(j),loc(j),width,N)
158         Y(M+1,:) = Yr(:)-Y(M,:)
159         Y(M+2,:) = Yr(:)-Y(M-1,:)
160         Y(M+3,:) = Yr(:)-Y(M-2,:)
161       end if
162     end do
163
164   end subroutine iboundaryY
165
166   subroutine addinletY(Ys,x,concentration,loc,width,M)
167     implicit none
168     integer :: M, i
169     real (kind = 8) :: width, concentration, loc
170     real (kind = 8), dimension(-2:M+3) :: Ys, x
171     ! Set initial boundary condtions for inlets on walls
172     do i = 1,M
173       if ((x(i)>loc-width/2.0d0).and.(x(i)<loc+width/2.0d0)) then
174           Ys(i) = 2.0d0*concentration
175       end if
176     end do
177   end subroutine addinletY
178
179   subroutine ICu(u,xu,yu,N,M)
180     implicit none
181     integer :: i, j, N, M
182     real (kind = 8), dimension(0:M,0:N+1) :: u
183     real (kind = 8), dimension(0:M) :: xu
184     real (kind = 8), dimension(0:N+1) :: yu
185     do i = 1,M-1
```

```
186         do j = 1,N
187           u(i,j) = uinitial(xu(i),yu(j));
188         end do
189       end do
190     end subroutine ICu
191
192     subroutine ICv(v,xv,yv,N,M)
193       implicit none
194       integer :: i, j, N, M
195       real (kind = 8), dimension(0:M+1,0:N) :: v
196       real (kind = 8), dimension(0:M+1) :: xv
197       real (kind = 8), dimension(0:N) :: yv
198       do i = 1,M
199         do j = 1,N-1
200           v(i,j) = vinitial(xv(i),yv(j))
201         end do
202       end do
203     end subroutine ICv
204
205     subroutine ICY(Y,xy,yy,N,M)
206       implicit none
207       integer :: i, j, N, M
208       real (kind = 8), dimension(-2:M+3,-2:N+3) :: Y
209       real (kind = 8), dimension(-2:M+3) :: xy
210       real (kind = 8), dimension(-2:N+3) :: yy
211       do i = 1,M
212         do j = 1,N
213           Y(i,j) = Yinitial(xy(i),yy(j))
214         end do
215       end do
216     end subroutine ICY
217
218     function uinitial(x,y)
219       implicit none
220       real (kind = 8) :: x, y, uinitial
221       uinitial = dsin(1.2d0*x+0.1d0) + dsin(1.4d0*y)
222     end function uinitial
223
224     function vinitial(x,y)
225       implicit none
226       real (kind = 8) :: x, y, vinitial
227       vinitial = dcos(1.4d0*y) + dcos(1.3d0*x)
228     end function vinitial
229
230     function Yinitial(x,y)
231       implicit none
232       real (kind = 8) :: x, y, Yinitial
233       Yinitial = dsin(0.7d0*x) + dcos(1.2d0*y)
234     end function Yinitial
235
236
237     subroutine boundaryu(u,ur,ul,N,M)
238       implicit none
239       integer :: N, M
```

```
240       real (kind = 8), dimension(0:N+1) :: ul, ur
241       real (kind = 8), dimension(0:M,0:N+1) :: u
242       u(0,:) = ul; u(M,:) = ur;
243       u(:,0) = -u(:,1); u(:,N+1) = -u(:,N);
244     end subroutine boundaryu
245
246     subroutine boundaryv(v,xv,vt,N,M)
247       implicit none
248       integer :: j, i, N, M
249       real (kind = 8), dimension(0:M+1) :: vt, xv
250       real (kind = 8), dimension(0:M+1,0:N) :: v
251       j = 1; ! Set bottom cells (Neumann condition)
252       do i = 0,M+1
253           if ((xv(i)>1.5d0).and.(xv(i)<2.5d0)) then
254               v(i,0) = (4.0d0/3.0d0)*v(i,j)-(1.0d0/3.0d0)*v(i,j+1)
255           else
256               v(i,0) = 0.0d0
257           end if
258       end do
259
260       ! Boundary conditions for left and right walls plus top wall
261       v(0,:) = -v(1,:); v(M+1,:) = -v(M,:); v(:,N) = vt;
262     end subroutine boundaryv
263
264     subroutine calcR(R,Y,xM,yN,deltax,deltay,N,M)
265       implicit none
266       integer :: i, j, N, M
267       real (kind = 8) :: R, xM, yN, deltax, deltay
268       real (kind = 8), dimension(-3:M+2,-3:N+2) :: Y
269       R = 0.0d0
270       do i = 1,M
271         do j = 1,N
272           R = R + Y(i,j)*(1.0d0-Y(i,j))*deltax*deltay
273         end do
274       end do
275       R = R/(xM*yN)
276     end subroutine calcR
277
278     subroutine calcS(S,Y,v,deltax,Lout,N,M)
279       implicit none
280       integer :: i, j, N, M
281       real (kind = 8) :: S, deltax, Lout
282       real (kind = 8), dimension(-3:M+2,-3:N+2) :: Y
283       real (kind = 8), dimension(0:M+1,0:N) :: v
284       S = 0.0d0
285       j = 0;
286       do i = 1,M
287         S = S + (-1.0d0*v(i,j))*Yij12(i,j-1,Y,N,M)*(1.0d0-Yij12(i,j-1,Y,N,M))
288       end do
289       S = S*deltax/Lout;
290     end subroutine calcS
291
292     function Yij12(i,j,Y,N,M)
293       implicit none
```

```
294       integer :: i, j, N, M
295       real (kind = 8) :: Yij12
296       real (kind = 8), dimension(-3:M+2,-3:N+2) :: Y
297       Yij12 = (0.5d0)*(Y(i,j) + Y(i,j+1));
298     end function Yij12
299
300     subroutine diagnostics(iterations,start,time,timefinal,LinfR)
301       implicit none
302       integer :: iterations
303       real (kind = 8) :: start, time, finish, LinfR, timeremaining, timefinal
304       print '("Simulation time =",d20.10," " )', time
305       print '("Iterations =",I0," ")', iterations
306       call cpu_time(finish)
307       print '("Computational Time = ",f20.10," seconds.")',finish-start
308       print '("Infinity Norm (Poisson) = ", d15.10," ")', LinfR
309       timeremaining = -(finish-start) + (finish-start)/time*timefinal;
310       if (timeremaining > 3600) then
311         print '("Estimated Time Remaining = ",f20.10," hours.")',timeremaining/3600
312       else if (timeremaining > 60) then
313         print '("Estimated Time Remaining = ",f20.10," minutes.")',timeremaining/60
314       else
315         print '("Estimated Time Remaining = ",f20.10," seconds.")',timeremaining
316       end if
317     end subroutine diagnostics
318
319     subroutine boundaryY(Y,Yinletconditions,xy,yy,nl,nt,nr,loc,width,N,M)
320       implicit none
321       integer :: j, N, M, nl, nt, nr
322       real (kind = 8) :: width
323       real (kind = 8), dimension(-2:N+3) :: yy
324       real (kind = 8), dimension(-2:M+3) :: xy
325       real (kind = 8), dimension(-2:M+3,-2:N+3) :: Y
326       real (kind = 8), dimension(1:nl+nt+nr) :: loc, Yinletconditions
327       logical :: l, r, t
328       ! Set initial values for ghost cells
329       ! Left Wall
330       Y(0,:) = Y(1,:); Y(-1,:) = Y(2,:); Y(-2,:) = Y(3,:)
331
332       ! Right Wall
333       Y(M+1,:) = Y(M,:); Y(M+2,:) = Y(M-1,:); Y(M+3,:) = Y(M-2,:)
334
335       ! Bottom Wall
336       Y(:,0) = Y(:,1); Y(:,-1) = Y(:,2); Y(:,-2) = Y(:,3)
337
338       ! Top Wall
339       Y(:,N+1) = Y(:,N); Y(:,N+2) = Y(:,N-1); Y(:,N+3) = Y(:,N-2)
340
341       ! Add inlet boundary condtions:
342       do j = 1,nl+nt+nr
343         if (j <= nl) then
344           l = .TRUE.; r = .FALSE.; t = .FALSE.;
345           call boundaryinletY(Y,yy,xy,Yinletconditions(j),loc(j),width,N,M,l,r,t)
346         else if(j <= nt+nl) then
347           l = .FALSE.; r = .FALSE.; t = .TRUE.;
```

```
348           call boundaryinletY(Y,xy,xy,Yinletconditions(j),loc(j),width,N,M,l,r,t)
349         else
350           l = .FALSE.; r = .TRUE.; t = .FALSE.;
351           call boundaryinletY(Y,yy,xy,Yinletconditions(j),loc(j),width,N,M,l,r,t)
352         end if
353       end do
354
355     end subroutine boundaryY
356
357     subroutine boundaryinletY(Y,yy,x,concentration,loc,width,N,M,left,right,top)
358       implicit none
359       integer :: j, N, M
360       logical :: left, right, top
361       real (kind = 8) :: concentration, loc, width
362       real (kind = 8), dimension(-2:N+3) :: yy
363       real (kind = 8), dimension(-2:M+3) :: x
364       real (kind = 8), dimension(-2:M+3,-2:N+3) :: Y
365
366       ! Boundary conditions on left wall
367       if (left) then
368         do j = 1,N
369           if ((yy(j)>loc-(width/2.0d0)).and.(yy(j)<loc+(width/2.0d0))) then
370             Y(0,j) = 2.0d0*concentration-Y(1,j)
371             Y(-1,j) = 2.0d0*concentration-Y(2,j)
372             Y(-2,j) = 2.0d0*concentration-Y(3,j)
373           end if
374         end do
375       end if
376
377       ! Boundary conditions on right wall
378       if (right) then
379         do j = 1,N
380           if ((yy(j)>loc-(width/2.0d0)).and.(yy(j)<loc+(width/2.0d0))) then
381             Y(M+1,j) = 2.0d0*concentration-Y(M,j)
382             Y(M+2,j) = 2.0d0*concentration-Y(M-1,j)
383             Y(M+3,j) = 2.0d0*concentration-Y(M-2,j)
384           end if
385         end do
386       end if
387
388       ! Boundary conditions on top wall
389       if (top) then
390         do j = 1,M
391           if ((x(j)>loc-(width/2.0d0)).and.(x(j)<loc+(width/2.0d0))) then
392             Y(j,N+1) = 2.0d0*concentration-Y(j,N)
393             Y(j,N+2) = 2.0d0*concentration-Y(j,N-1)
394             Y(j,N+3) = 2.0d0*concentration-Y(j,N-2)
395           end if
396         end do
397       end if
398
399     end subroutine boundaryinletY
400
401     subroutine savedata(u,xu,yu,v,xv,yv,Y,xy,yy,N,M,time)
```

```
402       implicit none
403       integer :: i, j, N, M
404       character(len=1024) :: ufile, vfile, Yfile, Ypfile, append
405       real (kind = 8) :: time
406       real (kind = 8), dimension(-2:N+3) :: yy
407       real (kind = 8), dimension(-2:M+3) :: xy
408       real (kind = 8), dimension(-2:M+3,-2:N+3) :: Y
409       real (kind = 8), dimension(0:M+1,0:N) :: v
410       real (kind = 8), dimension(0:M+1) :: xv
411       real (kind = 8), dimension(0:N) :: yv
412       real (kind = 8), dimension(0:M,0:N+1) :: u
413       real (kind = 8), dimension(0:M) :: xu
414       real (kind = 8), dimension(0:N+1) :: yu
415       if (time < 10.0d0) then
416         write(append,'(I1)') int(time)
417       else
418         write(append,'(I2)') int(time)
419       end if
420       ufile = trim('u_' // append);
421       vfile = trim('v_' // append);
422       Yfile = trim('Y_' // append);
423       Ypfile = trim('Yp_' // append);
424       open(1, file = ufile); open(2, file = vfile); open(3, file = Yfile); open(4, file =
      ↪  Ypfile)
425       ! Write data for u
426       do j = 0,N+1
427           do i = 0,M
428               write(1,*)  xu(i), yu(j), u(i,j)
429           end do
430       end do
431       ! Write data for v
432       do j = 0,N
433           do i = 0,M+1
434               write(2,*) xv(i), yv(j), v(i,j)
435           end do
436       end do
437       ! Write data for Y
438       do j = -2,N+3
439           do i = -2,M+3
440               write(3,*) xy(i), yy(j), Y(i,j)
441               write(4,*) xy(i), yy(j), Y(i,j)*(1.0d0-Y(i,j))
442           end do
443       end do
444       close(1); close(2); close(3); close(4);
445     end subroutine savedata
446
447     function Volin(velocity,x,loc,width,size,dx,normal)
448       implicit none
449       integer :: i, size
450       real (kind = 8) :: Volin, loc, width, dx, normal
451       real (kind = 8), dimension(0:size+1) :: velocity, x
452       Volin = 0.0d0
453       do i = 1,size
454         if ((x(i)>loc-(width/2.0d0)).and.(x(i)<loc+(width/2.0d0))) then
```

```
        Volin = Volin + velocity(i)
      end if
    end do
    Volin = Volin*dx*normal;
  end function Volin

  subroutine
 ↪  Volflow(VinY1,VinY0,nl,nt,nr,concentration,locations,width,N,M,dx,dy,ul,ur,vt,x,y)
    implicit none
    integer :: nl, nt, nr, N, M, dummyn
    real (kind = 8) :: VinY1, VinY0, width, dx, dy
    real (kind = 8), dimension(1:nl+nt+nr) :: locations, concentration
    real (kind = 8), dimension(0:N+1) :: ul, ur, y
    real (kind = 8), dimension(0:M+1) :: vt, x
    VinY0 = 0.0d0; VinY1 = 0.0d0;
    do dummyn = 1,nl+nt+nr
      if (dummyn <= nl) then
        if (concentration(dummyn)>0.9d0) then
          VinY1 = VinY1 + Volin(ul,y,locations(dummyn),width,N,dx,1.0d0)
        else
          VinY0 = VinY0 + Volin(ul,y,locations(dummyn),width,N,dx,1.0d0)
        end if
      else if (dummyn <= nl+nt) then
        if (concentration(dummyn)>0.9d0) then
          VinY1 = VinY1 + Volin(vt,x,locations(dummyn),width,M,dy,-1.0d0)
        else
          VinY0 = VinY0 + Volin(vt,x,locations(dummyn),width,M,dy,-1.0d0)
        end if
      else
        if (concentration(dummyn)>0.9d0) then
          VinY1 = VinY1 + Volin(ur,y,locations(dummyn),width,N,dx,-1.0d0)
        else
          VinY0 = VinY0 + Volin(ur,y,locations(dummyn),width,N,dx,-1.0d0)
        end if
      end if
    end do
  end subroutine Volflow

end module swissarmy
```

”vcycle.f95”:

```
module Vcycle
implicit none


private
public :: poisson, boundary, resid, Linf2d, L12d, L22d, Lagrange

real (kind=8), parameter :: pi = 4.0d0*datan(1.0d0)


contains

subroutine poisson(phi,f,dphidn,h,nIterMax,eps,LinfR,M,N,n1)
```

```
14  implicit none
15  ! integer, dimension(0:1) :: a;
16  integer :: nIterMax, n1, N, M, i,j
17  real (kind = 8), dimension(0:M+1,0:N+1) :: phi, f, r
18  real (kind = 8), dimension(0:3) :: dphidn
19  real (kind = 8) :: h, eps, avg
20  real (kind = 8), dimension(0:nIterMax) :: LinfR
21  ! real (kind = 8), dimension(:,:), allocatable :: r
22
23  ! a = shape(phi); M = a(0) - 2;allocate(r(0:M+1,0:M+1));
24  r(:,:) = 0.0d0;
25  r = resid(phi,f,h,M,N)
26  LinfR(0) = Linf2d(r,M,N);
27
28  do n1 = 1,nIterMax
29      phi = multigrid(phi,f,dphidn,h,M,N);
30      ! print *, '!~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ multigrid done...
    ↪  ~~~~~~~~~~~~~~~~~~~~~~~~~~~!';
31  ! do j = 0,N+1
32      ! do i = 0,M+1
33          ! print *,i+1,j+1, phi(i,j)
34      ! end do
35  ! end do
36  avg = 0.0d0
37  do j = 0,N+1
38      do i = 0,M+1
39          avg = avg+phi(i,j)
40      end do
41  end do
42  avg = 1/(dble(M)*dble(N))*avg; phi(:,:) = phi(:,:) - avg;
43
44      r = resid(phi,f,h,M,N); LinfR(n1) = Linf2d(r,M,N);! print *, n1, LinfR(n1)
45      if (LinfR(n1) < eps) then
46          exit
47      end if
48  end do
49
50  ! do j = 0,N+1
51      ! do i = 0,M+1
52          ! print *, i,j,r(i,j)
53      ! end do
54  ! end do
55
56  ! print *, n1, LinfR(n1)
57
58  end subroutine
59
60  function GaussSeidel(phi,f,dphidn,h,n1,M,N)
61  implicit none
62  integer :: n1, i, j, iterations, M, N
63  real (kind = 8), dimension (0:M+1,0:N+1) :: phi, f, GaussSeidel
64  real (kind = 8), dimension (0:3) :: dphidn
65  real (kind = 8) :: h
66
```

```
67
68  do iterations = 1,n1
69      do j = 1,N
70          do i = 1,M
71              phi(i,j) = 0.25d0*(phi(i-1,j)+phi(i+1,j)+phi(i,j-1)+phi(i,j+1)) - &
72              0.25d0*h**2*f(i,j)
73          end do
74      end do
75      call boundary(phi,h,dphidn,M,N)
76  end do
77
78  GaussSeidel = phi
79
80  end function GaussSeidel
81
82  function resid(phi,f,h,M,N)
83  implicit none
84  integer :: i, j, M, N
85  real (kind = 8), dimension (0:M+1,0:N+1) :: phi, f, resid
86  real (kind = 8) :: h
87
88  resid(:,:) = 0.0d0;
89  !~ residual(:,0) residual(:,ed) residual(0,:) residual(ed,:)
90  ! row end           row 0        left column   right column
91
92
93  ! Calculate residual on interior points
94  do j = 1,N ! Interior points
95      do i = 1,M ! Interior points
96          resid(i,j) = f(i,j) - cdif2d(phi(i-1:i+1,j-1:j+1),h)
97      end do
98  end do
99
100 resid(:,0) = resid(:,1); resid(:,N+1) = resid(:,N); resid(0,:) = resid(1,:);
101 resid(M+1,:) = resid(M,:);
102
103
104 end function resid
105
106 function restrict(rh,M,N)
107 implicit none
108 integer :: i, j, i_f, j_f, M, N
109 real (kind = 8), dimension (0:M+1,0:N+1) :: rh
110 real (kind = 8), dimension (:,:), allocatable :: restrict
111 real (kind = 8) :: sum1 = 0.0d0
112
113
114 ! Get the dimension of rh and allocate space for r2h, which has half as many cells as rh
115 allocate(restrict(0:M/2+1,0:N/2+1)); restrict(:,:) = 0.0d0;
116
117
118 ! Perform the restriction operation, take the average of four cells and use it as the
    ↪  value for the new coarse cell
119 do j = 1,N/2 ! Loop through the y-dir cells (coarse)
```

```
120     do i = 1,M/2 ! Loop through the x-dir cells (coarse)
121         do j_f = (2*j)-1,2*j ! Loop through the adjacent  y-dir cells (fine)
122             do i_f = (2*i)-1,2*i ! Loop through the adjacent x-dir cells (fine)
123                 sum1 = sum1 + rh(i_f,j_f) ! Add up each cell center
124             end do
125         end do
126     ! Set the i,j cell to the average of the 4 surronding cells and reset sum1
127     restrict(i,j) = 0.25d0*sum1; sum1 = 0.0d0
128     end do
129 end do
130 
131 
132 end function restrict
133 
134 function prolong(e2h,M,N)
135 implicit none
136 integer :: i, j, i_f, j_f, M, N
137 real (kind = 8), dimension (0:M/2+1,0:N/2+1) :: e2h
138 real (kind = 8), dimension (0:M+1,0:N+1) :: prolong
139 prolong(:,:) = 0.0d0
140 
141 ! Perform the prolongation operation, take the value of four cells and use it as the
    ↪  value for the new fine cells
142 do j = 1,N/2 ! Loop through the y-dir cells (coarse)
143     do i = 0,M/2 ! Loop through the x-dir cells (coarse)
144         do j_f = 2*j-1,2*j ! Loop through the adjacent y-dir cells (fine)
145             do i_f = 2*i-1,2*i ! Loop through the adjacent x-dir cells (fine)
146                 prolong(i_f,j_f) = e2h(i,j) ! Set the 4 fine cells equal to the
    ↪  overlaying coarse cell
147             end do
148         end do
149     end do
150 end do
151 
152 end function prolong
153 
154 recursive function multigrid(phi,f,dphidn,h,M,N) result (phir)
155 implicit none
156 integer :: n1, N, M!, i, j
157 real (kind = 8), dimension (0:M+1,0:N+1) :: phi, f
158 real (kind = 8), dimension (0:3) :: dphidn
159 real (kind = 8) :: h
160 real (kind = 8), dimension (:,:), allocatable :: e2h, rh, r2h, eh, phir
161 
162 n1 = 10;
163 allocate(phir(0:M+1,0:N+1),rh(0:M+1,0:N+1),r2h(0:M/2+1,0:N/2+1))
164 ! On the fine mesh, perform n1 iteration of Gauss-Seidel with phi as the initial guess
165 phir = GaussSeidel(phi,f,dphidn,h,n1,M,N)
166 
167 ! If there are more than 2 elements in the mesh, then perform the V-cycle
168 if ( (M > 2).and.(N > 2) ) then
169     rh = resid(phi,f,h,M,N) ! Calculate the residual at the current phi
170     r2h = restrict(rh,M,N); call boundary(r2h,2*h,dphidn,M/2,N/2) ! Restrict the
    ↪  residuals to a coarse mesh of 2h
```

```
171     allocate(e2h(0:M/2+1,0:N/2+1)); e2h(:,:) = 0.0d0; ! Allocate space for e2h and set
    ↪  inital values to 0
172     e2h = multigrid(e2h,r2h,dphidn,2*h,M/2,N/2) ! Perform multigrid on the error on the
    ↪  mesh with the residuals as the inital guess
173     allocate(eh(0:M+1,0:N+1))
174     eh = prolong(e2h,M,N); call boundary(eh,h,dphidn,M,N) ! Prolong the error to the fine
    ↪  mesh
175     phi = phi + eh; call boundary(phi,h,dphidn,M,N) ! Add the error to the guess phi
176     phir = GaussSeidel(phi,f,dphidn,h,1,M,N) ! Perform Gauss-Seidel with the new phi
177 end if
178
179
180 end function multigrid
181
182 function cdif2d(phi,h)
183 implicit none
184 real (kind=8) :: h, cdif2d
185 real (kind=8), dimension(-1:1,-1:1) :: phi
186 cdif2d =
    ↪  (phi(-1,0)-2.0d0*phi(0,0)+phi(1,0))/h**2+(phi(0,-1)-2.0d0*phi(0,0)+phi(0,1))/h**2
187 end function cdif2d
188
189 function Linf2d(error,M,N)
190     implicit none
191     integer :: M, N, i, j
192     real(kind=8), dimension(0:M+1,0:N+1) :: error
193     real(kind=8) :: Linf2d
194     Linf2d = dabs(error(1,1));
195     do j = 1,N
196         do i = 1,M
197             if (dabs(error(i,j)) > Linf2d) then
198                 Linf2d = dabs(error(i,j))
199             end if
200         end do
201     end do
202 end function Linf2d
203
204 subroutine boundary(phi,h,dphidn,M,N)
205 implicit none
206 real (kind = 8), dimension (0:M+1,0:N+1) :: phi
207 real (kind = 8), dimension (0:3) :: dphidn
208 real (kind = 8) :: h
209 integer :: i, j, M, N
210
211 do j = 0,N+1
212     phi(0,j) = phi(1,j) - h*dphidn(0)
213 end do
214
215 do j = 0,N+1
216     phi(M+1,j) = phi(M,j) + h*dphidn(1)
217 end do
218
219 do i = 0,M+1
220     phi(i,0) = phi(i,1) - h*dphidn(2)
```

```
221 end do
222
223 do i = 0,M+1
224     phi(i,N+1) = phi(i,N) + h*dphidn(3)
225 end do
226
227
228 end subroutine boundary
229
230 function L12d(error,M,N)
231     implicit none
232     integer :: M, N, i, j
233     real(kind=8), dimension(0:M+1,0:N+1) :: error
234     real(kind=8) :: L12d
235     L12d = 0.0d0
236     do j = 0,N+1
237         do i = 0,M+1
238             L12d = L12d + dabs(error(i,j))
239         end do
240     end do
241     L12d = ((1.0d0)/(M+N+2.0d0))*L12d
242 end function L12d
243
244 function L22d(error,M,N)
245     implicit none
246     integer :: M, N, i, j
247     real(kind=8), dimension(0:M+1,0:N+1) :: error
248     real(kind=8) :: L22d
249     L22d = 0.0d0
250     do j = 0,N+1
251         do i = 0,M+1
252             L22d = L22d + (error(i,j)*error(i,j))
253         end do
254     end do
255     L22d = dsqrt(((1.0d0)/(M+N+2.0d0))*L22d)
256 end function
257
258 subroutine
    ↪ Lagrange(u,v,unext,vnext,deltax,deltay,dt,xv,phi,Linf,nItermax,dphidn,N,M,n1,eps)
259   implicit none
260   integer :: i, j, N, M, nItermax, n1
261   real (kind = 8) :: deltax, deltay, ucorr, dt, q, eps
262   real (kind = 8), dimension(0:M,0:N+1) :: u, unext
263   real (kind = 8), dimension(0:M+1,0:N) :: v, vnext
264   real (kind = 8), dimension(0:M+1,0:N+1) :: phi, f
265   real (kind = 8), dimension(0:M+1) :: xv
266   real (kind = 8), dimension(0:nIterMax) :: Linf
267   real (kind = 8), dimension(0:3) :: dphidn
268   ! Correct outlet velocities to ensure volume conservation
269   ! Calculate volumetric flow rate into the control volume
270   q = deltay*(sum(u(0,:))-sum(u(M,:)))+deltax*(sum(v(:,0))-sum(v(:,N))); ucorr = q;
271
272
273   ! print *, 'Correctional Velocity', ucorr
```

```
274
275     ! Add the correction velocity to the outlet only
276     do i = 0,M+1
277         if ((xv(i)>1.5d0).and.(xv(i)<2.5d0)) then
278             v(i,0) = v(i,0) - ucorr;
279         end if
280     end do
281
282     ! do j = 0,N
283     !     do i = 0,M+1
284     !         print *, v(i,j)
285     !     end do
286     ! end do
287
288     ! Calculate right hand side of Poisson equation
289     f(:,:) = 0.0d0;
290     do i = 1,M
291         do j = 1,N
292             f(i,j) = (1.0d0/dt)*((u(i,j)-u(i-1,j))/deltax+(v(i,j)-v(i,j-1))/deltay)
293         end do
294     end do
295
296
297     ! do j = 0,N+1
298     !     do i = 0,M+1
299     !         print *, f(i,j)
300     !     end do
301     ! end do
302
303     ! Set initial values of phi to 0
304
305     call poisson(phi,f,dphidn,deltax,nItermax,eps,Linf,M,N,n1)
306
307     ! do j = 0,N+1
308     !     do i = 0,M+1
309     !         print *, phi(i,j)
310     !     end do
311     ! end do
312
313
314     ! Correct velocities using Lagrange multiplier
315     do j = 1,N
316         do i = 1,M-1
317             unext(i,j) = u(i,j) - dt*(phi(i+1,j)-phi(i,j))/deltax
318         end do
319     end do
320
321     do j = 1,N-1
322         do i = 1,M
323             vnext(i,j) = v(i,j) - dt*(phi(i,j+1)-phi(i,j))/deltay
324         end do
325     end do
326
327     ! Apply velocity boundary conditions to ghost cells only
```

```
  unext(:,N+1) = -unext(:,N); unext(:,0) = -unext(:,1);
  vnext(0,:) = -vnext(1,:); vnext(M+1,:) = -vnext(M,:);

  ! Set node boundary conditions from u and v to unext and vnext
  unext(0,:) = u(0,:); unext(M+1,:) = u(M+1,:)
  vnext(:,0) = v(:,0); vnext(:,N+1) = v(:,N+1)

  u = unext; v = vnext;

  ! print *, '!~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ u After Correction
  ↪  ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~!';
  ! do j = 0,N+1
      ! do i = 0,M
          ! print *, u(i,j)
      ! end do
  ! end do

  ! print *, '!~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ v After Correction
  ↪  ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~!';
  ! do j = 0,N
      ! do i = 0,M+1
          ! print *, v(i,j)
      ! end do
  ! end do
end subroutine Lagrange

end module Vcycle
```

"WENO5.f95":

```
module WENO5
use swissarmy
use AdamsB
  implicit none

  private
  public :: TVDRK3

contains

  function WENO(a,b,c,d)
  implicit none
  real (kind = 8) :: a, b, c, d, IS0, IS1, IS2, eps, alpha0, alpha1, alpha2, omega0,
  ↪  omega2, WENO
  ! real (kind = 8), intent(out) WENO
  ! Calculate smoothness indicators
  eps = 1.0d-6; IS0 = 13.0d0*(a-b)**2+3.0d0*(a-3.0d0*b)**2; IS1 =
  ↪  13.0d0*(b-c)**2+3.0d0*(b+c)**2;
  IS2 = 13.0d0*(c-d)**2+3.0d0*(3.0d0*c-d)**2;
  ! Calculate alphas
  alpha0 = 1.0d0/(eps+IS0)**2; alpha1 = 6.0d0/(eps+IS1)**2; alpha2 = 3.0d0/(eps+IS2)**2;
  ! Calculate omegas
  omega0 = alpha0/(alpha0+alpha1+alpha2); omega2 = alpha2/(alpha0+alpha1+alpha2);
  WENO =
  ↪  (1.0d0/3.0d0)*omega0*(a-2.0d0*b+c)+(1.0d0/6.0d0)*(omega2-(1.0d0/2.0d0))*(b-2.0d0*c+d);
```

```
23    end function WENO
24
25    function dplus(phi,i,j,N,M)
26    implicit none
27    integer :: i, j, N, M
28    real (kind = 8) :: dplus
29    real (kind = 8), dimension(-2:M+3,-2:N+3) :: phi
30    dplus = phi(i+1,j) - phi(i,j)
31    end function dplus
32
33    function dplusy(phi,i,j,N,M)
34    implicit none
35    integer :: i, j, N, M
36    real (kind = 8) :: dplusy
37    real (kind = 8), dimension(-2:M+3,-2:N+3) :: phi
38    dplusy = phi(i,j+1) - phi(i,j)
39    end function dplusy
40
41    function dminusdplus(phi,i,j,N,M)
42    implicit none
43    integer :: i, j, N, M
44    real (kind = 8) :: dminusdplus
45    real (kind = 8), dimension(-2:M+3,-2:N+3) :: phi
46    dminusdplus = phi(i+1,j) - 2.0d0*phi(i,j) + phi(i-1,j)
47    end function dminusdplus
48
49    function dminusdplusy(phi,i,j,N,M)
50    implicit none
51    integer :: i, j, N, M
52    real (kind = 8) :: dminusdplusy
53    real (kind = 8), dimension(-2:M+3,-2:N+3) :: phi
54    dminusdplusy = phi(i,j+1) - 2.0d0*phi(i,j) + phi(i,j-1)
55    end function dminusdplusy
56
57    function dphidxminus(phi,i,j,deltax,N,M)
58    implicit none
59    integer :: i, j, N, M
60    real (kind = 8) :: dphidxminus, deltax
61    ! real (kind = 8) :: dplus, dminusdplus, deltax, WENO
62    real (kind = 8), dimension(-2:M+3,-2:N+3) :: phi
63
64    dphidxminus =
    ↪  (1.0d0/(12.0d0*deltax))*(-dplus(phi,i-2,j,N,M)+7.0d0*dplus(phi,i-1,j,N,M)&
65
    ↪  +7.0d0*dplus(phi,i,j,N,M)-dplus(phi,i+1,j,N,M))-WENO(dminusdplus(phi,i-2,j,N,M)/deltax,&
66
    ↪  dminusdplus(phi,i-1,j,N,M)/deltax,dminusdplus(phi,i,j,N,M)/deltax,dminusdplus(phi,i+1,j,N,M)/deltax)
67
68    end function dphidxminus
69
70    function dphidxplus(phi,i,j,deltax,N,M)
71    implicit none
72    integer :: i, j, N, M
73    ! real (kind = 8) :: dplus, dminusdplus, deltax, WENO
```

```
74    real (kind = 8) :: dphidxplus, deltax
75    real (kind = 8), dimension(-2:M+3,-2:N+3) :: phi
76
77    dphidxplus = (1.0d0/(12.0d0*deltax))*(-dplus(phi,i-2,j,N,M)+7.0d0*dplus(phi,i-1,j,N,M)&
78
      ↪  +7.0d0*dplus(phi,i,j,N,M)-dplus(phi,i+1,j,N,M))+WENO(dminusdplus(phi,i+2,j,N,M)/deltax,&
79
      ↪  dminusdplus(phi,i+1,j,N,M)/deltax,dminusdplus(phi,i,j,N,M)/deltax,dminusdplus(phi,i-1,j,N,M)/deltax)
80
81    end function dphidxplus
82
83    function dphidyminus(phi,i,j,deltay,N,M)
84    implicit none
85    integer :: i, j, N, M
86    real (kind = 8) :: dphidyminus, deltay
87    real (kind = 8), dimension(-2:M+3,-2:N+3) :: phi
88
89    dphidyminus =
      ↪  (1.0d0/(12.0d0*deltay))*(-dplusy(phi,i,j-2,N,M)+7.0d0*dplusy(phi,i,j-1,N,M)&
90
      ↪  +7.0d0*dplusy(phi,i,j,N,M)-dplusy(phi,i,j+1,N,M))-WENO(dminusdplusy(phi,i,j-2,N,M)/deltay,&
91
      ↪  dminusdplusy(phi,i,j-1,N,M)/deltay,dminusdplusy(phi,i,j,N,M)/deltay,dminusdplusy(phi,i,j+1,N,M)/deltay)
92
93    end function dphidyminus
94
95    function dphidyplus(phi,i,j,deltay,N,M)
96    implicit none
97    integer :: i, j, N, M
98    ! real (kind = 8) :: dplus, dminusdplus, WENO
99    real (kind = 8) :: dphidyplus, deltay
100   real (kind = 8), dimension(-2:M+3,-2:N+3) :: phi
101
102   dphidyplus =
      ↪  (1.0d0/(12.0d0*deltay))*(-dplusy(phi,i,j-2,N,M)+7.0d0*dplusy(phi,i,j-1,N,M)&
103
      ↪  +7.0d0*dplusy(phi,i,j,N,M)-dplusy(phi,i,j+1,N,M))+WENO(dminusdplusy(phi,i,j+2,N,M)/deltay,&
104
      ↪  dminusdplusy(phi,i,j+1,N,M)/deltay,dminusdplusy(phi,i,j,N,M)/deltay,dminusdplusy(phi,i,j-1,N,M)/deltay)
105
106   end function
107
108   subroutine TVDRK3(phi,u,v,N,M,x,y,deltax,deltay,dt,loc,width,Yinletconditions,nl,nt,nr)
109   implicit none
110   integer :: N, M, i, j, nl, nt, nr
111   real (kind = 8) :: deltax, deltay, dt, a, b, width
112   real (kind = 8), dimension(0:M,0:N+1) :: u
113   real (kind = 8), dimension(0:M+1,0:N) :: v
114   real (kind = 8), dimension(-2:M+3) :: x
115   real (kind = 8), dimension(-2:N+3) :: y
116   real (kind = 8), dimension(-2:M+3,-2:N+3) :: phi0, phi1, phi2, phi3, phi
117   real (kind = 8), dimension(1:3,0:2) :: alpha
118   real (kind = 8), dimension(1:nl+nt+nr) :: loc, Yinletconditions
119
```

```
120     alpha(1,0) = 1.0d0; alpha(2,0) = -3.0d0/4.0d0; alpha(2,1) = 1.0d0/4.0d0;
121     alpha(3,0) = -1.0d0/12.0d0; alpha(3,1) = -1.0d0/12.0d0; alpha(3,2) = 2.0d0/3.0d0;
122
123     ! Set phi0 to the phi passed in
124     phi0 = phi;
125
126     ! do j = -2,N+3
127     !   do i = -2,M+3
128     !     print *, phi(i,j)
129     !   end do
130     ! end do
131
132     ! Calculate phi1
133     do j = 1,N
134       do i = 1,M
135           a = uij(u,i,j,N,M)
136           if (a>0.0d0)then
137               phi1(i,j) = phi0(i,j) - alpha(1,0)*(a*dt*dphidxminus(phi0,i,j,deltax,N,M))
138           else
139               phi1(i,j) = phi0(i,j) - alpha(1,0)*(a*dt*dphidxplus(phi0,i,j,deltax,N,M))
140           end if
141           b = vij(v,i,j,N,M)
142           if (b>0.0d0)then
143               phi1(i,j) = phi1(i,j) - alpha(1,0)*(b*dt*dphidyminus(phi0,i,j,deltay,N,M))
144           else
145               phi1(i,j) = phi1(i,j) - alpha(1,0)*(b*dt*dphidyplus(phi0,i,j,deltay,N,M))
146           end if
147       end do
148     end do
149
150     ! Apply boundary conditions to phi1
151     call boundaryY(phi1,Yinletconditions,x,y,nl,nt,nr,loc,width,N,M)
152
153     ! Calculate phi2
154     do j =1,N
155       do i = 1,M
156           a = uij(u,i,j,N,M)
157           if (a>0.0d0)then
158               phi2(i,j) = phi1(i,j) - alpha(2,0)*(a*dt*dphidxminus(phi0,i,j,deltax,N,M))&
159               - alpha(2,1)*(a*dt*dphidxminus(phi1,i,j,deltax,N,M))
160           else
161               phi2(i,j) = phi1(i,j) - alpha(2,0)*(a*dt*dphidxplus(phi0,i,j,deltax,N,M))&
162               - alpha(2,1)*(a*dt*dphidxplus(phi1,i,j,deltax,N,M))
163           end if
164           b = vij(v,i,j,N,M)
165           if (b>0.0d0)then
166               phi2(i,j) = phi2(i,j) - alpha(2,0)*(b*dt*dphidyminus(phi0,i,j,deltay,N,M))&
167                - alpha(2,1)*(b*dt*dphidyminus(phi1,i,j,deltay,N,M))
168           else
169               phi2(i,j) = phi2(i,j) - alpha(2,0)*(b*dt*dphidyplus(phi0,i,j,deltay,N,M))&
170              - alpha(2,1)*(b*dt*dphidyplus(phi1,i,j,deltay,N,M))
171           end if
172       end do
173     end do
```

```
174
175    ! Apply boundary conditions to phi2
176    call boundaryY(phi2,Yinletconditions,x,y,nl,nt,nr,loc,width,N,M)
177
178    ! Calculate phi3
179    do j = 1,N
180      do i = 1,M
181          a = uij(u,i,j,N,M)
182          if (a>0.0d0)then
183              phi3(i,j) = phi2(i,j) - alpha(3,0)*(a*dt*dphidxminus(phi0,i,j,deltax,N,M))&
184              - alpha(3,1)*(a*dt*dphidxminus(phi1,i,j,deltax,N,M))&
185              - alpha(3,2)*(a*dt*dphidxminus(phi2,i,j,deltax,N,M))
186          else
187              phi3(i,j) = phi2(i,j) - alpha(3,0)*(a*dt*dphidxplus(phi0,i,j,deltax,N,M))&
188              - alpha(3,1)*(a*dt*dphidxplus(phi1,i,j,deltax,N,M))&
189              - alpha(3,2)*(a*dt*dphidxplus(phi2,i,j,deltax,N,M))
190          end if
191          b = vij(v,i,j,N,M)
192          if (b>0.0d0)then
193              phi3(i,j) = phi3(i,j) - alpha(3,0)*(b*dt*dphidyminus(phi0,i,j,deltax,N,M))&
194              - alpha(3,1)*(b*dt*dphidyminus(phi1,i,j,deltax,N,M))&
195              - alpha(3,2)*(b*dt*dphidyminus(phi2,i,j,deltax,N,M))
196          else
197              phi3(i,j) = phi3(i,j) - alpha(3,0)*(b*dt*dphidyplus(phi0,i,j,deltax,N,M))&
198              - alpha(3,1)*(b*dt*dphidyplus(phi1,i,j,deltax,N,M))&
199              - alpha(3,2)*(b*dt*dphidyplus(phi2,i,j,deltax,N,M))
200          end if
201      end do
202    end do
203
204    ! Apply boundary conditions to phi3
205    call boundaryY(phi3,Yinletconditions,x,y,nl,nt,nr,loc,width,N,M)
206
207    phi = phi3;
208    ! do j = -2,N+3
209    !   do i = -2,M+3
210    !     print *, phi(i,j)
211    !  end do
212    ! end do
213
214    end subroutine TVDRK3
215
216  end module WENO5
```

"contourplotsu.m":

```
1  %~~~~~~~~~~~~~~~~~~~~~~~~~ u ~~~~~~~~~~~~~~~~~~~~~~~~~%
2  i = 1; Mu = 256 + 1; Nu = 128 + 2;
3
4  filename = 'u_1';
5  data = importdata(filename);
6  [x,y,z] = myreshape(data(:,:), Mu, Nu);
7  figure(i)
8  myplotu(x,y,z,i)
9  % print(i,'-dpng','u_1'); i = i + 1;
```

```
10
11
12  filename = 'u_3';
13  data = importdata(filename);
14  [x,y,z] = myreshape(data(:,:), Mu, Nu);
15  figure(i)
16  myplotu(x,y,z,i)
17  % print(i,'-dpng','u_3'); i = i + 1;
18
19
20  filename = 'u_5';
21  data = importdata(filename);
22  [x,y,z] = myreshape(data(:,:), Mu, Nu);
23  figure(i)
24  myplotu(x,y,z,i)
25  % print(i,'-dpng','u_5'); i = i + 1;
26
27
28  filename = 'u_10';
29  data = importdata(filename);
30  [x,y,z] = myreshape(data(:,:), Mu, Nu);
31  myplotu(x,y,z,i)
32  % print(i,'-dpng','u_10'); i = i + 1;
33
34
35  filename = 'u_15';
36  data = importdata(filename);
37  [x,y,z] = myreshape(data(:,:), Mu, Nu);
38  figure(i)
39  myplotu(x,y,z,i)
40  % print(i,'-dpng','u_15'); i = i + 1;
41
42
43  filename = 'u_20';
44  data = importdata(filename);
45  [x,y,z] = myreshape(data(:,:), Mu, Nu);
46  figure(i)
47  myplotu(x,y,z,i)
48  % print(i,'-dpng','u_20'); i = i + 1;
49
50
51
52   close all
```

"myplotu.m":

```
1  function [complete] = myplotu (x,y,z,i)
2  % figure(i, 'pos' ,[10 10 900 600])
3  pcolor(x,y,z); shading interp;
4  colorbar; xlim([0 4]); ylim([0 2])
5  xlabel('x','Fontsize',20)
6  ylabel('y','Fontsize',20)
7  zlabel('v','Fontsize',20)
8  set(gca,'fontsize',15,'LineWidth',1);
9  complete = 1;
```

```
end
```

"contourplotsv.m":

```
%~~~~~~~~~~~~~~~~~~~~~~~~~ v ~~~~~~~~~~~~~~~~~~~~~~~~~%
i = 1; Mv = 256 + 2; Nv = 128 + 1; i = 1;

filename = 'v_1';
data = importdata(filename);
[x,y,z] = myreshape(data(:,:), Mv, Nv);
figure(i)
myplotv(x,y,z,i)
% print(i,'-dpng','v_1'); i = i + 1;


filename = 'v_3';
data = importdata(filename);
[x,y,z] = myreshape(data(:,:), Mv, Nv);
figure(i)
myplotv(x,y,z,i)
% print(i,'-dpng','v_3'); i = i + 1;


filename = 'v_5';
data = importdata(filename);
[x,y,z] = myreshape(data(:,:), Mv, Nv);
figure(i)
myplotv(x,y,z,i)
% print(i,'-dpng','v_5'); i = i + 1;


filename = 'v_10';
data = importdata(filename);
[x,y,z] = myreshape(data(:,:), Mv, Nv);
figure(i)
myplotv(x,y,z,i)
% print(i,'-dpng','v_10'); i = i + 1;


filename = 'v_15';
data = importdata(filename);
[x,y,z] = myreshape(data(:,:), Mv, Nv);
figure(i)
myplotv(x,y,z,i)
% print(i,'-dpng','v_15'); i = i + 1;


filename = 'v_20';
data = importdata(filename);
[x,y,z] = myreshape(data(:,:), Mv, Nv);
figure(i)
myplotv(x,y,z,i)
% print(i,'-dpng','v_20'); i = i + 1;

% close all;
```

”myplotv.m”:

```
function [complete] = myplotv (x,y,z,i)
% figure(i, 'pos' ,[10 10 900 600])
pcolor(x,y,z); shading interp;
colorbar; caxis([-1.5,0]); xlim([0 4]); ylim([0 2])
xlabel('x','Fontsize',20)
ylabel('y','Fontsize',20)
zlabel('v','Fontsize',20)
set(gca,'fontsize',15,'LineWidth',1);
complete = 1;
end
```

”contourplotsY.m”:

```
%~~~~~~~~~~~~~~~~~~~~~~~~~ Y ~~~~~~~~~~~~~~~~~~~~~~~~~%
i = 1; MY = 256 + 6; NY = 128 + 6;

filename = 'Y_1';
data = importdata(filename);
[x,y,z] = myreshape(data(:,:), MY, NY);
figure(i)
myplotY(x,y,z,i)
% print(i,'-dpng','Y_1'); i = i + 1;


filename = 'Y_3';
data = importdata(filename);
[x,y,z] = myreshape(data(:,:), MY, NY);
figure(i)
myplotY(x,y,z,i)
% print(i,'-dpng','Y_3'); i = i + 1;


filename = 'Y_5';
data = importdata(filename);
[x,y,z] = myreshape(data(:,:), MY, NY);
figure(i)
myplotY(x,y,z,i)
% print(i,'-dpng','Y_5'); i = i + 1;


filename = 'Y_10';
data = importdata(filename);
[x,y,z] = myreshape(data(:,:), MY, NY);
figure(i)
myplotY(x,y,z,i)
% print(i,'-dpng','Y_10'); i = i + 1;


filename = 'Y_15';
data = importdata(filename);
[x,y,z] = myreshape(data(:,:), MY, NY);
figure(i)
myplotY(x,y,z,i)
```

```
% print(i,'-dpng','Y_15'); i = i + 1;


filename = 'Y_20';
data = importdata(filename);
[x,y,z] = myreshape(data(:,:), MY, NY);
figure(i)
myplotY(x,y,z,i)
% print(i,'-dpng','Y_20'); i = i + 1;


% close all
```

"myplotY.m":

```
function [complete] = myplotY (x,y,z,i)
% figure(i, 'pos' ,[10 10 900 600])
pcolor(x,y,z); shading interp;
colorbar; caxis([-0.1,1.1]); xlim([0 4]); ylim([0 2])
xlabel('x','Fontsize',20)
ylabel('y','Fontsize',20)
zlabel('v','Fontsize',20)
set(gca,'fontsize',15,'LineWidth',1);
complete = 1;
end
```

"myreshape.m":

```
function [x,y,z] = myreshape(A, M, N)
  z = zeros(M,N); x = zeros(M,N); y = zeros(M,N);
for i = 1:N
  z(:,i) = A(M*(i-1)+1:M*i,3);
  x(:,i) = A(M*(i-1)+1:M*i,1);
  y(:,i) = A(M*(i-1)+1:M*i,2);
end
end
```

"uprobeplot.plt":

```
set terminal epslatex "enhanced"
set output "uprobe.tex"
set xlabel '${\Large t}$'
set ylabel '${\Large u}$'
data = './uprobe'
set key top right box 2
set grid
set size 1.2,1.5
plot data using 1:2 title '$u$' with lines
```

"vprobeplot.plt":

```
set terminal epslatex "enhanced"
set output "vprobe.tex"
set xlabel '${\Large t}$'
set ylabel '${\Large v}$'
data = './vprobe'
set key top right box 2
```

```
set grid
set size 1.2,1.5
plot data using 1:2 title '$v$' with lines
```

"Yprobeplot.plt":

```
set terminal epslatex "enhanced"
set output "Yprobe.tex"
set xlabel '${\Large t}$'
set ylabel '${\Large Y}$'
data = './Yprobe'
set key top right box 2
set grid
set size 1.2,1.5
plot data using 1:2 title '$Y$' with lines
```

"Rprobeplot.plt":

```
set terminal epslatex "enhanced"
set output "Rprobe.tex"
set xlabel '${\Large t}$'
set ylabel '${\Large R}$'
data = './Rprobe'
set key top right box 2
set grid
set size 1.2,1.5
plot data using 1:2 title '$R(t)$' with lines
```