Programming Assignment #3 : Solving Incompressible Energy Equation

Jerin Roberts
December 21, 2016

Supervisor: Dr. Carl Ollivier-Gooch
Locations: University of British Columbia

CONTENTS

LIST OF FIGURES

LIST OF TABLES

1. Introduction

The are many physical phenomenons in physics and engineering that require linear and non-linear partial differential equations to described the true nature of the system. Solving these systems analytically and finding exact solutions for these equations can be difficult and often require simplifications that ultimately don't fully represent the problem being investigated. Numerical methods for solving PDE's provides a means for finding approximations to the exact solutions without having to make sacrificial simplifications. With recent advancements in computational technology numerical methods can now be easily applied to large and difficult problems that would otherwise be impossible to solve.

1.1. Problem Overview

Numerical problems are essentially solved by breaking the entire solution domain into small discrete points (mesh) and finding the solution at or around these areas. Each point requires the solving the differential equations that represent physical phenomenon being investigated. Since the exact solution cannot be computed, it is instead approximated using various techniques and methods. In this assignment the 2D in-compressible Navier-Stokes equations are applied to a lid-driven cavity problem. The program will employ an second ordered centered flux calculation with an implicit euler time advance method. The implicit method will solve using approximate factorization and Gauss-Jordan elimination via the Block Thomas algorithm. Boundary conditions will be implement using ghost cells allowing the interior scheme to remain the same during calculation of bordering cells. The ghost cells are calculated such the boundary condition is enforced at $i = 1/2$ or $j = 1/2$.

2. Implementation

2.1. Program Overview

The C/C++language was selected for this programming assignment. The scripts were compiled using g++/gcc version 5.4.0 on Ubuntu 16.04.02 and are available in the attached zip or for clone via the link provided: `https://github.com/j16out/cfd510` . The program itself is broken into 3 pieces and two levels to produce a modular set that makes it easier to apply to different problems. The highest level contains the "macro" or the main function which can be modified for different problems. The numerical directory contains numerical.cpp script and its header file numerical.hpp. This set contains all the functions necessary for solving the problem numerically. The appendix contains the .hpp file which lists all functions with a short description of each. The vroot directory contains the scripts necessary for drawing data. These scripts make use of the ROOT-v6 libraries. ROOT is a popular data analysis framework primarily written in C++ and Phython. For more information on ROOT libraries visit the link provided: `https://root.cern.ch/` .

2.2. Solution Method

There are three general routes that can be taken to solve the Navier-Stokes equations each of which is characterized by the way it satisfies the continuity equation; Stream function-vorticity methods take and express the velocity field in terms of derivatives of the stream function. Because of this the momentum equations are replaced by a single vorticity transport equation, while a Poisson equation relates the stream function and vorticity. This method works fine for 2D flows, but can be difficult to use in three dimensions as stream function is a 2D entity. The 2nd type is Pressure Poisson methods, which take the divergence of the momentum equations and manipulate this to get a Poisson equation for the pressure. This equation can be simplified by using the continuity equation to eliminate terms. These methods are well-known in the field, but have a reputation for being difficult to program. The method selected for this project is the artificial compressibility method. This method adds a physical time derivative of pressure to the continuity equation. This derivative is scaled by a parameter β that effectively sets the pseudo-compressibility of the fluid.The addition of a time derivative of pressure enables the coupling of the continuity equation with the momentum equations which ideally allows us to advance pressure and velocity in time together.

The non-dimensional Navier-Stokes equations in artificial compressibility form can be written as equations 2.1 to 2.3

$$\frac{\partial P}{\partial t} + \frac{1}{\beta}\frac{\partial u}{\partial x} + \frac{1}{\beta}\frac{\partial v}{\partial y} = 0 \tag{2.1}$$

$$\frac{\partial u}{\partial t} + \frac{\partial u^2}{\partial x} + \frac{\partial vu}{\partial y} = -\frac{\partial P}{\partial x} + \frac{1}{Re}\left(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2}\right) \tag{2.2}$$

$$\frac{\partial v}{\partial t} + \frac{\partial vu}{\partial x} + \frac{\partial v^2}{\partial y} = -\frac{\partial P}{\partial y} + \frac{1}{Re}\left(\frac{\partial^2 v}{\partial x^2} + \frac{\partial^2 v}{\partial y^2}\right) \tag{2.3}$$

We can take this and write it in our standard form (eq 2.4 which enables to integrate over the computational cell by applying gauss's Theorem and dividing over the size of the finite volume.

$$\frac{\partial U}{\partial t} + \frac{\partial F}{\partial x} + \frac{\partial G}{\partial y} = 0 \tag{2.4}$$

where

$$U = \begin{bmatrix} P \\ u \\ v \end{bmatrix} \tag{2.5}$$

$$F = \begin{bmatrix} \frac{u}{\beta} \\ u^2 + P - \frac{1}{Re}\frac{\partial u}{\partial x} \\ uv - \frac{1}{Re}\frac{\partial v}{\partial x} \end{bmatrix} \tag{2.6}$$

$$G = \begin{bmatrix} \frac{v}{\beta} \\ uv - \frac{1}{Re}\frac{\partial u}{\partial y} \\ v^2 + P - \frac{1}{Re}\frac{\partial v}{\partial y} \end{bmatrix} \tag{2.7}$$

Which leads to navier stokes in finite volume form as shown in equation 2.8

$$\frac{dU_{ij}}{dt} = -\frac{F_{i+\frac{1}{2},j} - F_{i-\frac{1}{2},j}}{\triangle x} - \frac{G_{i,j+\frac{1}{2}} - G_{i,j-\frac{1}{2}}}{\triangle y} \tag{2.8}$$

Where the right hand side (RHS) represents the 2nd order approximation to the flux integral, where U, G, and F as before are represented as 3x1 matrices of Pressure and velocity components.

3. Validation

3.1. Flux Computation

The program was built and tested in small components to ensure its correctness. The flux integral was run and compared to the exact solution for the flux integral. The problem will employ a second ordered centered flux calculation which is found on the RHS of the discretized energy equation.

The flux was calculated using the function $update\_flux()$, which is shown below. The function calls two other functions which grab the necessary data using the $surr$ struct from the current solution. The flux calculated using $calc\_newcell()$ is then stored on the $carray$ struct under the $f1$ array.

```
void update_flux(carray & myarray){
surr mysurr;
vec ftemp;
for(int j = 1; j < myarray.sizey-1; ++j)
{
    for(int i = 1; i < myarray.sizex-1; ++i)
    {
    //----get surrounding cells and compute new cell----//
    get_nsurcells(myarray, i, j, mysurr);
    calc_flux(myarray, mysurr, ftemp);

    //-----update current cell----//
    myarray.f1[i][j] = ftemp;
    }
}
}
```

The correctness of the flux integral was validated using the exact computed flux. The L_2 norms for the flux calculation are displayed in table 5.3. The order of accuracy for the scheme was determined using the log log method as displayed in figure 3.1. The order of accuracy

for this scheme was estimated to be 2nd order accurate based on the L_2 data from 8 different meshes.

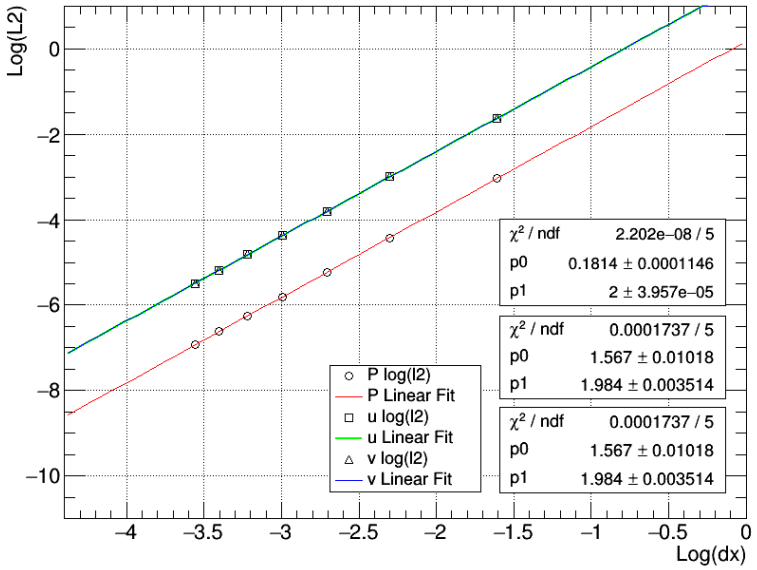


Figure 3.1: Order of Accuracy of Flux Integral for Pressure and Velocity components

Mesh Size	**Pressure L^2norm**	**Velocity L^2norm**	$\triangle x$	**Order**
10 x 10	$4.7946 * 10^{-2}$	$1.9505 * 10^{-1}$	0.1	-
20 x 20	$1.1983 * 10^{-2}$	$5.0025 * 10^{-2}$	0.05	1.952
40 x 40	$2.9957 * 10^{-3}$	$1.2585 * 10^{-2}$	0.025	1.980
80 x 80	$7.4892 * 10^{-4}$	$3.1514 * 10^{-3}$	0.0125	2.002

Table 3.1: Table of L^2 norms for increasing mesh size of flux integral for RHS

This matches what was expect by implementing a 2nd order interior flux evaluation and boundary schemes. It should be noted as seen in figure 3.1 the error decreases when refining the mesh. This is expect as the numerical solution is a discrete representation of the continuous exact solution.

3.2. Flux Jacobian

For the time discetization of the Navier-Stokes equations a implicit euler method was selected. The fully implicit time discretization of NS equations can be written as

$$\begin{aligned} &\delta U_{ij} + \triangle t A_x \delta U_{i-1,j} + \triangle t B_x \delta U_{i,j} + \triangle t C_x \delta U_{i+1,j} \\ &+\triangle t A_y \delta U_{i,j-1} + \triangle t B_y \delta U_{i,j} + \triangle t A_y \delta U_{i,j+1} \\ &= -\triangle t \frac{F_{i+\frac{1}{2},j} - F_{i-\frac{1}{2},j}}{\triangle x} - \triangle t \frac{G_{i,j+\frac{1}{2}} - G_{i,j-\frac{1}{2}}}{\triangle y} \end{aligned} \tag{3.1}$$

The implementation of the left hand side (LHS) can be validated by implementing a very small known change in solution for one cell between time steps. The difference between the flux calculated on the RHS from time level n to n+1 should be roughly equivalent to the LHS of the implementation (minus δU_{ij}).

$$\begin{aligned} &\left(\frac{F_{i+\frac{1}{2},j} - F_{i-\frac{1}{2},j}}{\triangle x} - \frac{G_{i,j+\frac{1}{2}} - G_{i,j-\frac{1}{2}}}{\triangle y} \right)^{n+1} \\ &-\left(\frac{F_{i+\frac{1}{2},j} - F_{i-\frac{1}{2},j}}{\triangle x} - \frac{G_{i,j+\frac{1}{2}} - G_{i,j-\frac{1}{2}}}{\triangle y} \right)^{n} \\ &= A_x \delta U_{i-1,j} + B_x \delta U_{i,j} + C_x \delta U_{i+1,j} \\ &+ A_y \delta U_{i,j-1} + B_y \delta U_{i,j} + A_y \delta U_{i,j+1} \end{aligned} \tag{3.2}$$

Since we've only selected one cell to change only 5 cells in total will see a calculated change which corresponds each to one term on the LHS being non-zero. This provides a great method for narrowing down which term was incorrect. If the error was found to be greater then 10^{-10} then the term was considered to be incorrect. The errors for each corresponding component of change are tabulated in table 3.2.

Component (i, j)	**Pressure Error**	**u Error**	**v Error**
Ax (11, 10)	$3.000 * 10^{-15}$	$5.001 * 10^{-12}$	$5.001 * 10^{-12}$
Cx (9, 10)	$1.000 * 10^{-15}$	$5.002 * 10^{-12}$	$5.001 * 10^{-12}$
Bx By (10, 10)	$1.000 * 10^{-15}$	$2.500 * 10^{-13}$	$2.500 * 10^{-13}$
Cy (10, 9)	$1.000 * 10^{-15}$	$5.001 * 10^{-12}$	$5.000 * 10^{-12}$
Ay (10, 11)	$3.000 * 10^{-15}$	$4.995 * 10^{-12}$	$4.995 * 10^{-12}$

Table 3.2: Table of L^2 norms for increasing mesh size of flux integral

3.3. Block Thomas

We can re-write Equation 3.1 in a approximate factored form by combining the various small matrices (A, B, and C) into larger matrices as shown in equation 3.3.

$$\left(\frac{F^n_{i+\frac{1}{2},j} - F^n_{i-\frac{1}{2},j}}{\triangle x} - \frac{G^n_{i,j+\frac{1}{2}} - G^n_{i,j-\frac{1}{2}}}{\triangle y} \right) = [I + \triangle t D_x][I + \triangle t D_y]\delta\overline{U} \tag{3.3}$$

This approximately factored system can be solved by breaking the problem into two sets of line problems. In this case each line problem is a 3×3 block tridiagonal problem. An extension of the Thomas algorithm is used to solve this system of equations. The process for solving the system each time step is shown below.

```
void solve_LinSys(carray & myarray, double tstep, double & mdiff)
{
//--linear system num 1---//
crow myrow;
for(int j = 1; j < myarray.sizey-1; ++j)
{
load_row(myarray, myrow, j, tstep);
solve_block_thomas(myarray, myrow, myarray.sizex, j);
}
ccol mycol;
for(int i = 1; i < myarray.sizex-1; ++i)
{
load_col(myarray, mycol, i, tstep);
solve_block_thomas(myarray, mycol, myarray.sizey, i);
}
}
```

The $load\_row()$ function is used to retrieve the required Jacobian values and load them into a format to be solved by the Block Thomson algorithm. Once all the rows have been solved, the same process is completed for the columns. The result is the change in solution for one time step, which is used to update the current solution. The data for the final iteration over the columns on the first time step is displayed in the appendix for validation.

3.4. Boundary Conditions

The boundary condition are divided into two types; explicit and implicit. The explicit ones are used to enforce the boundary conditions during the flux calculation for the RHS. They are set using the $set\_ghost\_cell()$ function which is displayed below. For this problem only wall boundaries are considered. For stationary walls, both the normal and tangential velocity are zero. Therefore this means that for a wall at j = 1/2, we can set values in ghost cells using equations below 3.4.

$$\begin{aligned} u_{i,0} &= -u_{i,1} \\ v_{i,0} &= -v_{i,1} \\ P_{i,0} &= P_{i,1} \end{aligned} \tag{3.4}$$

For moving walls, the normal velocity is zero and the tangential velocity matches the wall velocity. An example of a wall moving along x is shown below.

$$\begin{aligned} u_{i,0} &= 2u_{wall} - u_{i,1} \\ v_{i,0} &= -v_{i,1} \\ P_{i,0} &= P_{i,1} \end{aligned} \tag{3.5}$$

Where the wall velocity is constant through out time. The implicit boundaries need not be altered for this case as the velocity of the wall does not change with time. There all four implicit wall boundaries implemented using the Jacobian matrix shown below which are loaded into the Block Thomas algorithm.

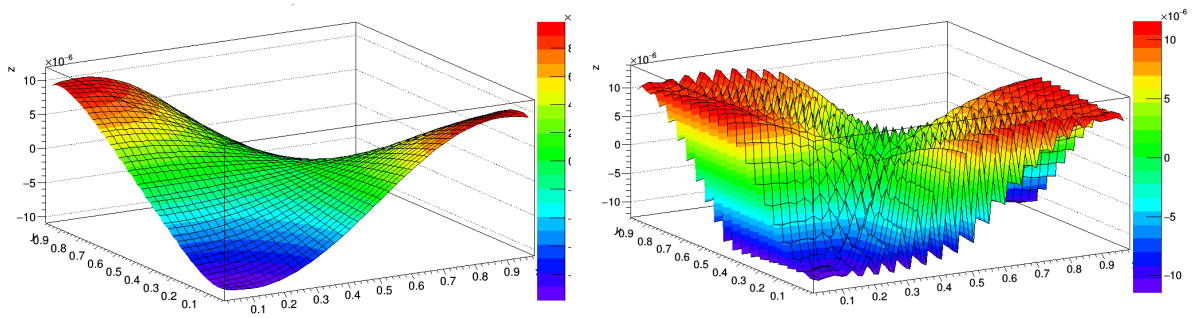
$$\begin{bmatrix} -1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} \tag{3.6}$$

3.5. Pressure Oscillations

It was found that steady-state pressure distributions have some oscillations. These were present because of decoupling between pressure in alternate lines of the mesh. To fix this problem a term was added to the right-hand side of the pressure equation which is displayed as 3.7.

$$A\left(\frac{\overline{P}_{i+1,j} - 2\overline{P}_{i,j} + \overline{P}_{i-1,j}}{\triangle x^2} + \frac{\overline{P}_{i,j+1} - 2\overline{P}_{i,j} + \overline{P}_{i,j-1}}{\triangle y^2}\right)\triangle x \triangle y \tag{3.7}$$

This essentially introduces an artificial component which for a properly selected A can smooth out the pressure fluctuations while maintaining 2nd order accuracy. A value of 0.01 was selected for A. The effect of the correction is displayed in figure 3.2



(a) (b)

Figure 3.2: Correcting pressure oscillations for 40x40 mesh (a) corrected A = 0.01 (b) uncorrected A = 0

4. Program Testing

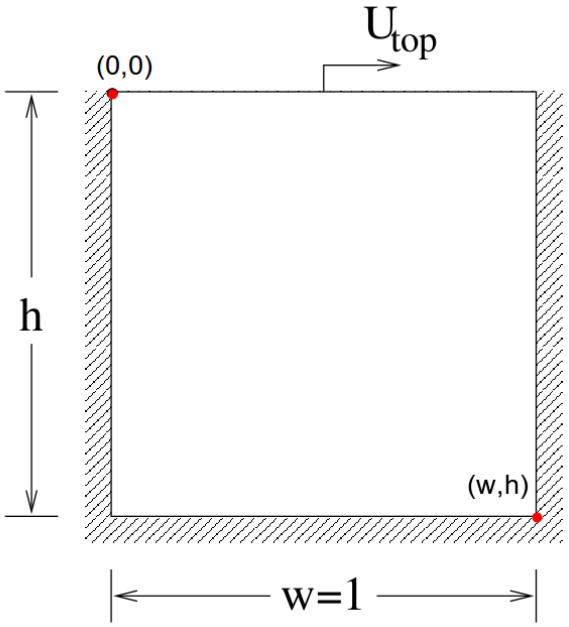


Figure 4.1: diagram of box modeled in problem

To test the program a problem concerning the flow in an enclosed box with a moving top was evaluated. A diagram of the problem is displayed in figure 4.1 The walls and the lid of the box also has a no-slip boundary condition, where the lid is at constant velocity U top. As the top of the box moves to the right, the fluid in the box circulates clockwise (for a square box). For all cases in this section a Reynolds number of 100 and $\beta = 1$ was used. The initial conditions for the problem are displayed in equation 4.1

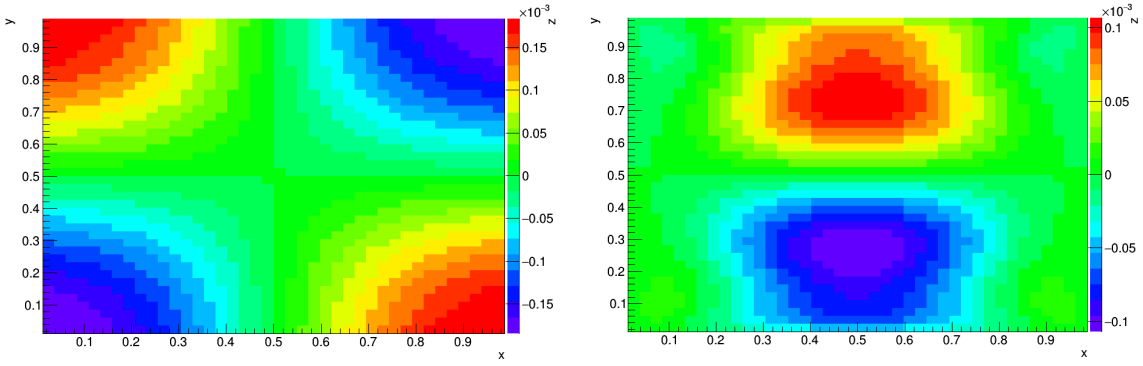
$$\begin{bmatrix} P \\ u \\ v \end{bmatrix} = \begin{bmatrix} P_0 \cos(\pi x)\cos(\pi y) \\ u_0 \sin(\pi x)\sin(2\pi y) \\ v_0 \sin(2\pi x)\sin(\pi y) \end{bmatrix} \tag{4.1}$$

The origin for the computational domain is set at top left corner of box. Therefore the lid is found along the y=0 plane which will be the convention for the remainder of report.

4.1. Stability

For stability test the top plate velocity was set to zero and the problem was solve to steady state. The convergence was logged for each iteration, its history is displayed in figure 5.3. A 20x20 mesh was used with a 0.05 second time step. The convergence exhibits some oscillatory behavior which seems to be mediated through the use of larger time steps. The oscillations seen in the convergence history can physically be thought of as pressure waves bouncing

around inside the domain which will inherently induce velocity changes that are out of phase with the pressure changes. The solution for pressure and velocity components converged toward zero gauge pressure and velocity which confirms the stability of the code. This is demonstrated in figures 4.2 where solutions are shown approaching zero and equalizing.



(a) (b)

Figure 4.2: Solutions approaching zero for Pressure (a) and Velocity u (b). Note the z scale is order $\approx 10^{-3}$

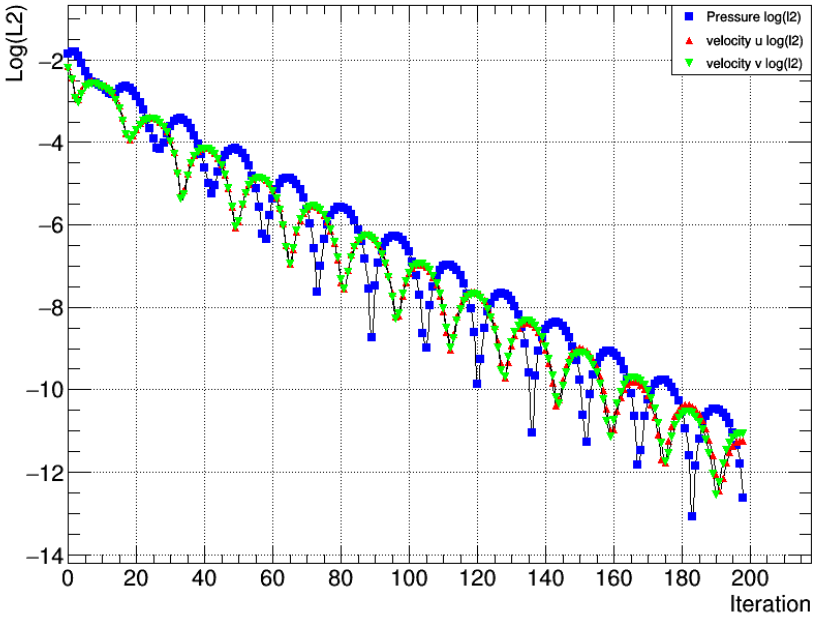


Figure 4.3: convergence history for flow inside enclosed box.

4.2. SOLUTION $U_{top} = 1$

The solution for the flow inside the box was calculated when the speed of the lid was set to $U_{top} = 1$. The height was set to $h = 1$ and time step set to $\triangle t = 0.1$. No over relaxation was used during this evaluation. Figure 4.7 displays the convergence history during the solving of the solution. Again we see oscillatory behavior which begins to smooth out as the solution refines. The velocity profile for the x-velocity u was plotted at $x = 0.5$ for different values of height inside the box which displays the u-velocity along the line of vertical symmetry. Since the mesh is even value points at x = 0.5 had to be calculated using surrounding cell data. For each y cell (row) 2 cells on each side of x = 0.5 were used (total = 4) to fit a 2nd degree polynomial function which was then used to calculate the value at $x = 0.5$. The u-velocity along the vertical line of symmetry for $U_{top} = 1$ is displayed in figure 4.4.

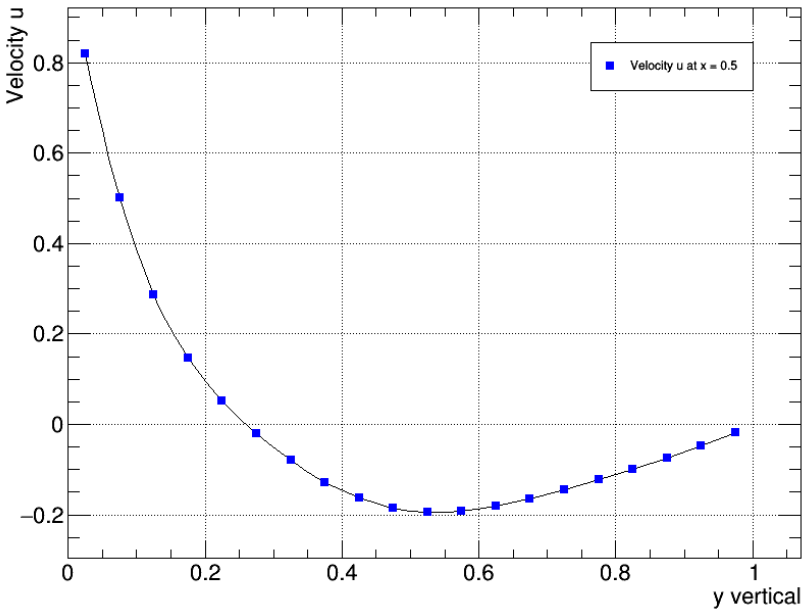
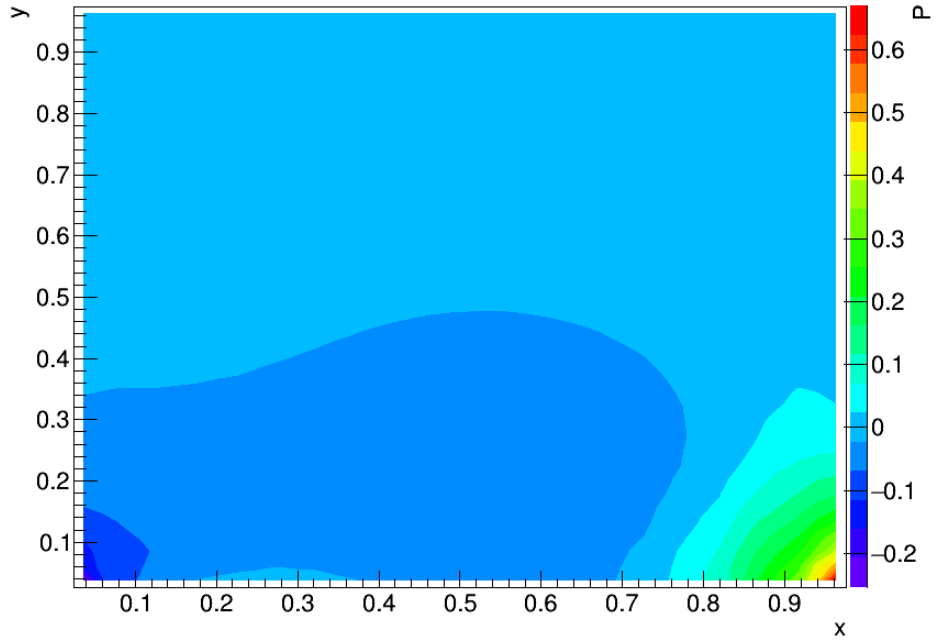
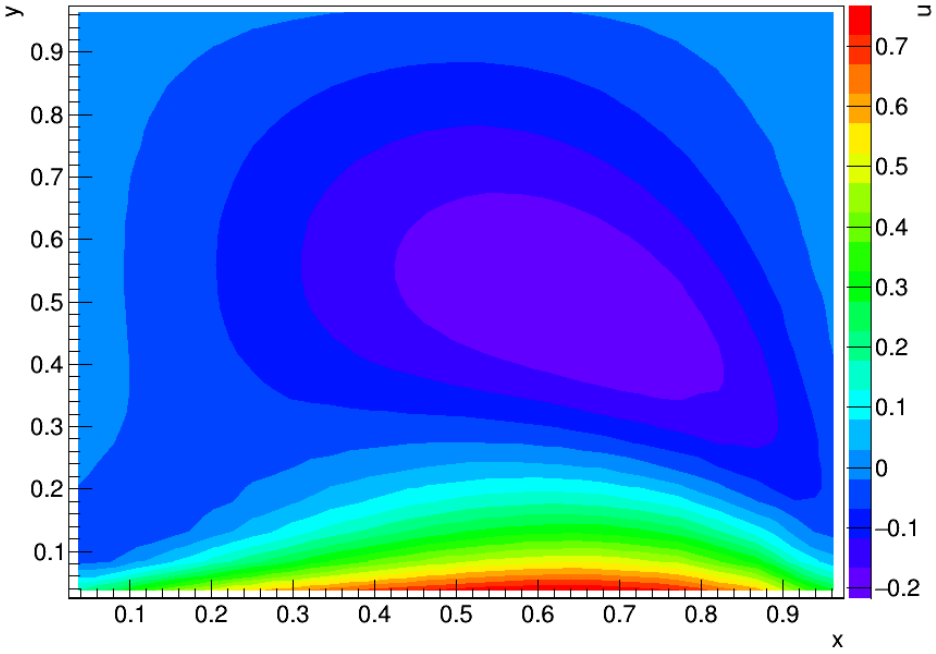


Figure 4.4: Velocity u profile at x = 0.5 for $U_{top} = 1$. Lid found at y = 0.



(a)



(b)

Figure 4.5: Pressure Distribution (a) and u-Velocity (b) inside box for Solution $U_{top} = 1$ and A = 0.5

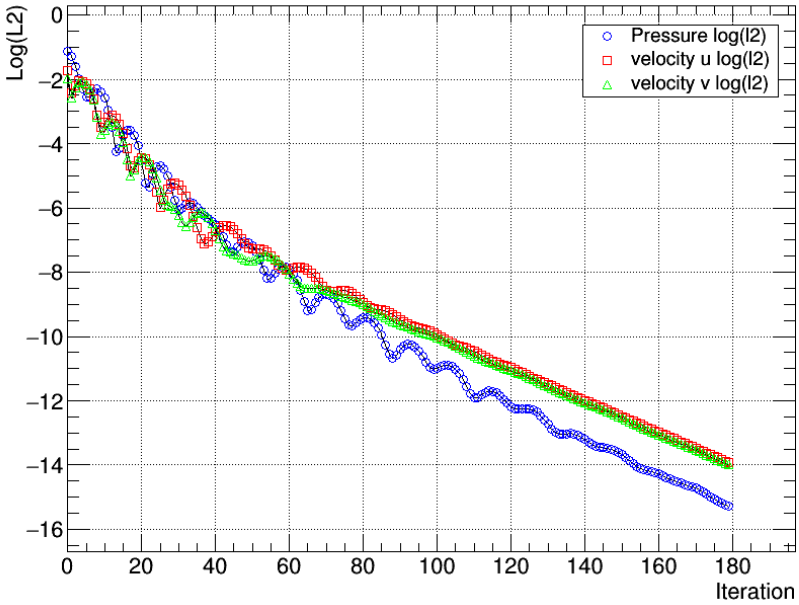


Figure 4.6: convergence history of flow inside enclosed box for $\triangle t = 0.1$

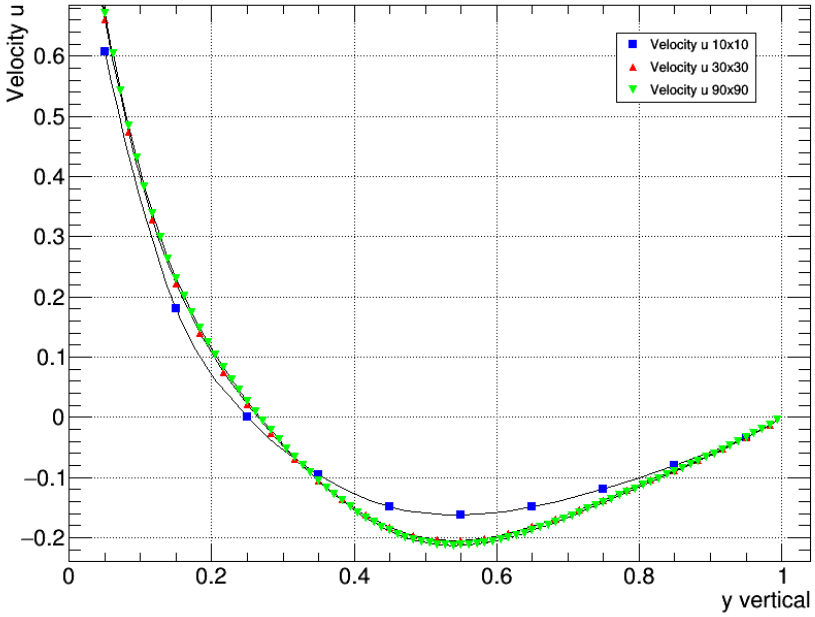


Figure 4.7: convergence history of flow inside enclosed box for $\triangle t = 0.1$

4.3. Solution $U_{top} = -1$

Additionally the solution for the flow inside the box was calculated when the speed of the lid was set to $U_{top} = -1$. The height was set to $h = 1$ and time step set to $\triangle t = 0.1$. No over relaxation was used during this evaluation. This was then contrasted with the previous solution at $U_{top} = 1$ by flipping $U_{top} = -1$ over the vertical axis of symmetry and adding it to $U_{top} = 1$.

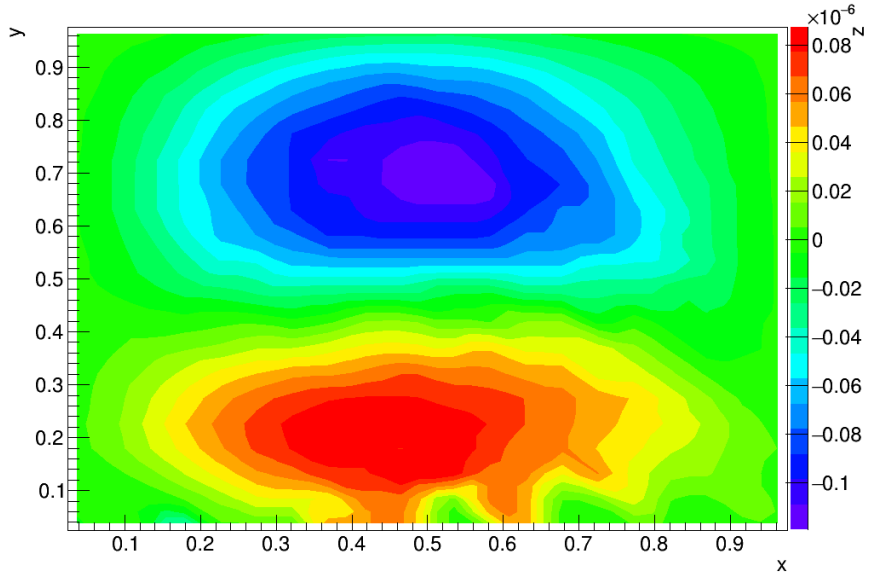


Figure 4.8: A contour of $u_{U_{top}=1}(x, y) + u_{U_{top}=-1}(1 - x, y)$ displaying the program symmetry

More specifically a contour of $u_{U_{top}=1}(x, y) + u_{U_{top}=-1}(1 - x, y)$ was found and displayed in figure 4.8. This shows the continuity of the program by comparing the error for opposite symmetry problems. Because the error seen is really small its attributed to a lack of iterative convergence.

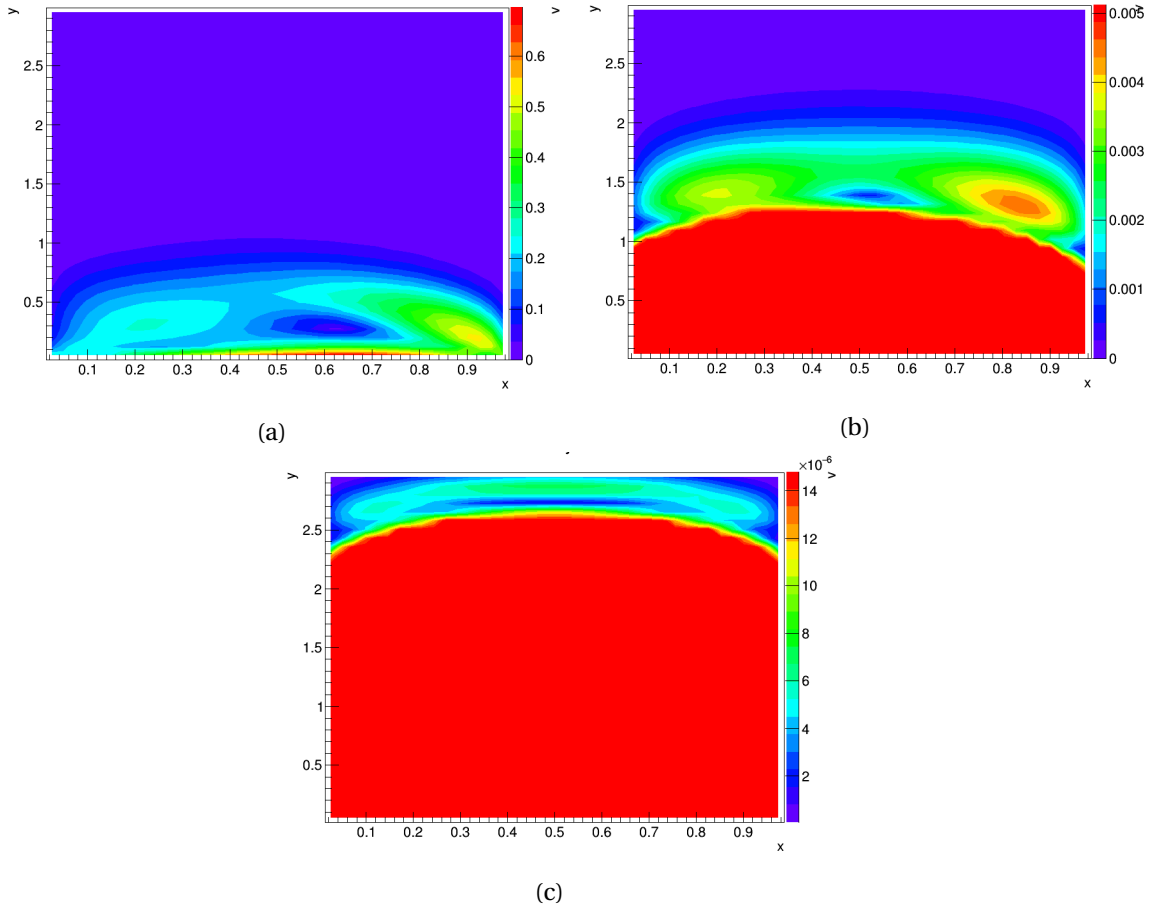
5. Lid-Driven Cavity Problem

For the Lid-driven cavity problem the height h of the box was increased. The single vortex seen in previous examples eventually becomes unstable, and a second vortex forms below it and eventually a third, and so on. The exploratory part of this problem will focus on this formation of additional vortices with an increase in the height of the cavity.

5.1. Vortex Location

When the height was increased to $h = 3$ the cavity was observed to produce a total of three vortices. The vortices were discovered using a contour plot of the magnitude of the velocity components u and v. By generating a contour plot of the magnitude of u and v one could easily

search out locations where the magnitude was zero, suggesting a potential area for a vortex. Figure 5.1 displays three vortices identified for a cavity of $h = 3$.

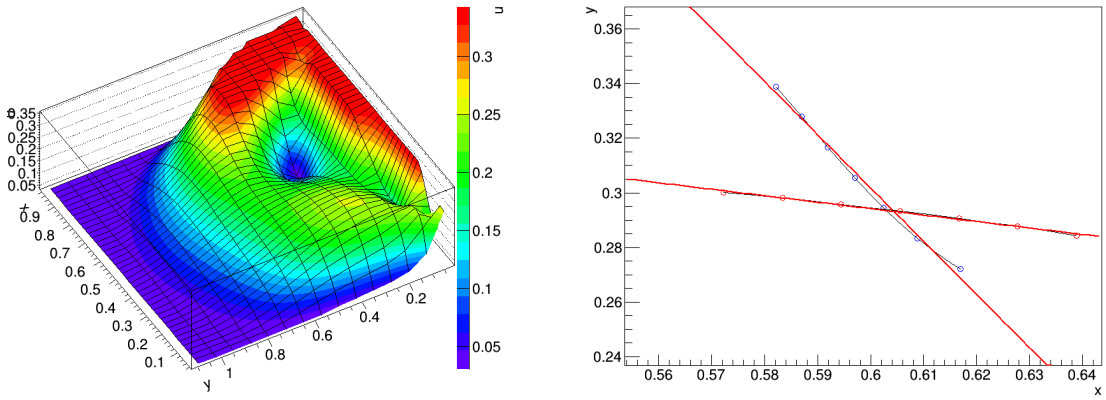


(a) (b)

(c)

Figure 5.1: Identifying vortices using magnitude of v and u contour plot

Once a vortex was identified a curve fitting method was employed to calculated its center. The method works by approximating the dip in velocity magnitude or the 'hole of the vortex' if you will, and approximates the center based on the fit of a parabolic arc. The program looks at individual slices for a set range fitting approx 10-20 points per slice. Each slice of the hole will contain a parabolic arc. The vertex of each arc for each slice is calculated and plotted; one plot for the y direction sweep and the other for the x sweep. Since the programs sweeps over both the x and y direction of the vortex hole, the lines mapping the vertex locations will eventually cross. This crossing point is used as the estimate for the center of the vortex. An example of the crossing point is shown in figure 5.2. The crossing points were calculated for three different vortices for 5 different grid sizes. The data is displayed in table **??**



(a) (b)

Figure 5.2: Displays an example of a velocity void due to a vortex (a) and a plot of the vertex centers crossing (b)

Mesh Size V#1	**x coordinate**	**y coordinate**	**Relative Error**	GCI_{fine}
40 x 120	0.618863	0.284613	-	-
60 x 180	0.607763	0.291379	0.0111	-
80 x 240	0.605493	0.292715	0.0027	0.2730
90 x 270	0.604404	0.293177	0.001089	0.3014
Mesh Size V#2	**x coordinate**	**y coordinate**	**Relative Error**	GCI_{fine}
40 x 120	0.614811	1.352830	-	-
60 x 180	0.521546	1.365390	0.0529	-
80 x 240	0.516247	1.370399	0.005	0.2730
90 x 270	0.515219	1.371391	0.001	0.2978
Mesh Size V#3	**x coordinate**	**y coordinate**	**Relative Error**	GCI_{fine}
40 x 120	0.500413	2.740107	-	-
60 x 180	0.496878	2.732679	0.00548	-
80 x 240	0.497947	2.727510	0.0031	0.354
90 x 270	0.497706	2.729092	0.0064	0.327

Table 5.1: Calculated coordinates for each vortex 1-3 using curve fit method

Values	udA
N_1, N_2, N_3	80, 60, 40
r_{21}	1.5
r_{32}	1.33
$u_1 dA$	0.097330
$u_2 dA$	0.095894
$u_3 dA$	0.091769
p	2.014
P_{ext}^{21}	0.099158
e_a^{21}	0.014754
e_{ext}^{21}	0.018438
GCI_{fine}^{21}	0.023480

Table 5.2: Table of calculation of uncertainty in the fine-grid solution

Relative Strength	40 x 120	60 x 180	80 x 240	$\overline{Ratio}$	GCI_{fine}
V1	0.091769	0.095894	0.097330	1.00	-
V2	0.000551	0.000823	0.000919	129.65	-
V3	0.000387	0.000435	0.000448	231.42	0.023480

Table 5.3: Calculated coordinates for each vortex 1-3 using curve fit method

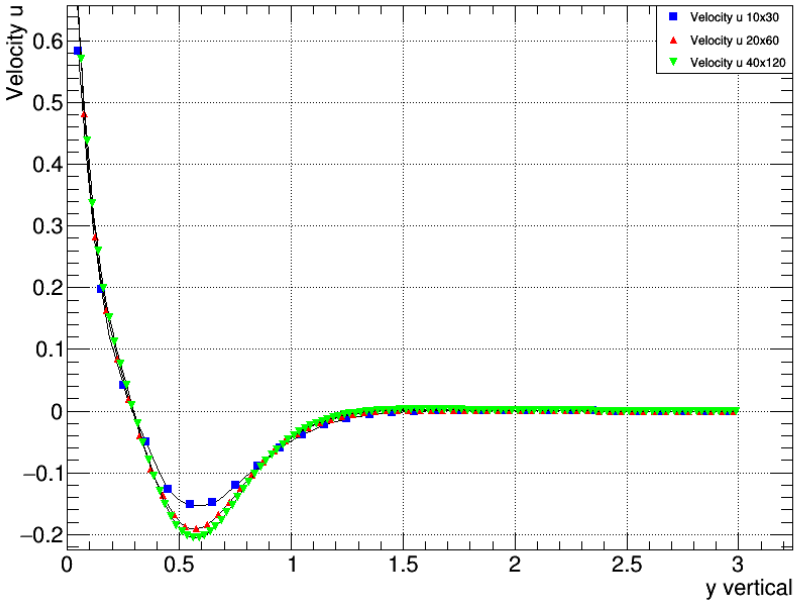


Figure 5.3: A u velocity profile for varying grid sizes illustrating grid convergence for lid-driven cavity

6. CONCLUSION

This project provides great insight into the internal algorithms used for calculating numerical solutions for time varying problems. The Energy Equation applied to a steady state channel provided a great platform for developing and testing the program. Having the analytic solution really help fine tune the program and helped given an idea on how accurate the solutions could be. Investigating the error and how it changes relative to boundaries and mesh sizes will provides insight on how the methods should be applied to bigger problems to minimize time and error. Stability analysis provided a great means for understanding the computational limits of the program in terms of speed. Numerical methods provide a great way of solving difficult problems and until new methods are discovered for finding exact solution will remain the main method for finding solutions to difficult problems.

A. Appendix

A.1. Thomas Block Data Last Pass

```
--------------------------------------------------------------------------------
Solution Pressure:
       |   i:  0|   i:  1|   i:  2|   i:  3|   i:  4|   i:  5|   i:  6|   i:  7|   i:  8|   i:  9|   i: 10|   i: 11|

j: 0|  |0.975528|0.975528|0.880037|0.698401|0.448401|0.154508|-0.15451|-0.44840|-0.69840|-0.88004|-0.97553|-0.97553|
j: 1|  |0.975528|-0.12097|-0.15591|-0.16377|-0.14228|-0.08996|-0.00555|0.125265|0.263872|0.304414|0.204633|-0.97553|
j: 2|  |0.880037|-0.15548|-0.18291|-0.18645|-0.16572|-0.11925|-0.03663|0.093823|0.228599|0.319403|0.293180|-0.88004|
j: 3|  |0.698401|-0.16458|-0.18823|-0.18715|-0.16136|-0.10623|-0.00496|0.104755|0.167101|0.226886|0.271163|-0.69840|
j: 4|  |0.448401|-0.14388|-0.16796|-0.16272|-0.12821|-0.04799|0.073081|0.124639|0.101237|0.091979|0.140944|-0.44840|
j: 5|  |0.154508|-0.09046|-0.12069|-0.10768|-0.04729|0.062648|0.128193|0.071260|-0.00635|-0.03708|0.000628|-0.15451|
j: 6|  |-0.15451|0.000628|-0.03708|-0.00635|0.071260|0.128193|0.062648|-0.04729|-0.10768|-0.12069|-0.09046|0.154508|
j: 7|  |-0.44840|0.140944|0.091979|0.101237|0.124639|0.073081|-0.04799|-0.12821|-0.16272|-0.16796|-0.14388|0.448401|
j: 8|  |-0.69840|0.271163|0.226886|0.167101|0.104755|-0.00496|-0.10623|-0.16136|-0.18715|-0.18823|-0.16458|0.698401|
j: 9|  |-0.88004|0.293180|0.319403|0.228599|0.093823|-0.03663|-0.11925|-0.16572|-0.18645|-0.18291|-0.15548|0.880037|
j:10|  |-0.97553|0.204633|0.304414|0.263872|0.125265|-0.00555|-0.08996|-0.14228|-0.16377|-0.15591|-0.12097|0.975528|
j:11|  |-0.97553|-0.97553|-0.88004|-0.69840|-0.44840|-0.15451|0.154508|0.448401|0.698401|0.880037|0.975528|0.975528|
Solution u:
       |   i:  0|   i:  1|   i:  2|   i:  3|   i:  4|   i:  5|   i:  6|   i:  7|   i:  8|   i:  9|   i: 10|   i: 11|

j: 0|  |0.048341|-0.048341|-0.140291|-0.218508|-0.275336|-0.305212|-0.30521|-0.27534|-0.21851|-0.14029|-0.04834|0.04834|
j: 1|  |-0.048341|0.01929|0.03328|0.02778|0.02598|0.03857|0.08024|0.162847|0.225790|0.184515|0.059112|-0.04834|
j: 2|  |-0.126558|-0.00376|-0.02894|-0.06383|-0.08717|-0.07806|0.00911|0.189838|0.335648|0.319562|0.114325|-0.12656|
j: 3|  |-0.156434|-0.01540|-0.05679|-0.09984|-0.12419|-0.10196|0.00440|0.117312|0.173958|0.205763|0.105926|-0.15643|
j: 4|  |-0.126558|-0.00840|-0.03170|-0.05239|-0.04706|0.00463|0.046172|-0.016635|-0.061023|-0.016065|0.029257|-0.12656|
j: 5|  |-0.048341|0.01181|0.04059|0.06941|0.10811|0.106676|-0.025096|-0.153912|-0.16344|-0.10171|-0.019104|-0.04834|
j: 6|  |0.04834|0.019104|0.10171|0.16344|0.153912|0.025096|-0.106676|-0.10811|-0.06941|-0.04059|-0.01181|0.048341|
j: 7|  |0.12656|-0.029257|0.016065|0.061023|0.016635|-0.046172|-0.00463|0.04706|0.05239|0.03170|0.00840|0.126558|
j: 8|  |0.15643|-0.105926|-0.205763|-0.173958|-0.117312|-0.00440|0.10196|0.12419|0.09984|0.05679|0.01540|0.156434|
j: 9|  |0.12656|-0.114325|-0.319562|-0.335648|-0.189838|-0.00911|0.07806|0.08717|0.06383|0.02894|0.00376|0.126558|
j:10|  |0.04834|-0.059112|-0.184515|-0.225790|-0.162847|-0.08024|-0.03857|-0.02598|-0.02778|-0.03328|-0.01929|0.048341|
j:11|  |-0.04834|0.04834|0.14029|0.21851|0.27534|0.30521|0.305212|0.275336|0.218508|0.140291|0.048341|-0.048341|
Solution v:
       |   i:  0|   i:  1|   i:  2|   i:  3|   i:  4|   i:  5|   i:  6|   i:  7|   i:  8|   i:  9|   i: 10|   i: 11|

j: 0|  |0.048341|-0.048341|-0.126558|-0.156434|-0.126558|-0.048341|0.04834|0.12656|0.15643|0.12656|0.04834|-0.04834|
j: 1|  |-0.048341|0.01604|-0.00732|-0.01777|-0.00896|0.01371|0.02761|-0.010619|-0.094134|-0.121034|-0.068592|0.04834|
j: 2|  |-0.140291|0.02948|-0.03398|-0.06048|-0.03312|0.04411|0.11029|0.027418|-0.195288|-0.313856|-0.182487|0.14029|
j: 3|  |-0.218508|0.02710|-0.06555|-0.10086|-0.05202|0.07045|0.15687|0.046938|-0.174510|-0.323373|-0.210378|0.21851|
j: 4|  |-0.275336|0.02796|-0.08228|-0.11891|-0.04729|0.09731|0.131845|0.004158|-0.109454|-0.186289|-0.154520|0.27534|
j: 5|  |-0.305212|0.04211|-0.06599|-0.09779|-0.00840|0.089258|0.018080|-0.030590|0.00387|-0.01941|-0.080965|0.30521|
j: 6|  |-0.30521|0.080965|0.01941|-0.00387|0.030590|-0.018080|-0.089258|0.00840|0.09779|0.06599|-0.04211|0.305212|
j: 7|  |-0.27534|0.154520|0.186289|0.109454|-0.004158|-0.131845|-0.09731|0.04729|0.11891|0.08228|-0.02796|0.275336|
j: 8|  |-0.21851|0.210378|0.323373|0.174510|-0.046938|-0.15687|-0.07045|0.05202|0.10086|0.06555|-0.02710|0.218508|
j: 9|  |-0.14029|0.182487|0.313856|0.195288|-0.027418|-0.11029|-0.04411|0.03312|0.06048|0.03398|-0.02948|0.140291|
j:10|  |-0.04834|0.068592|0.121034|0.094134|0.010619|-0.02761|-0.01371|0.00896|0.01777|0.00732|-0.01604|0.048341|
j:11|  |0.04834|-0.04834|-0.12656|-0.15643|-0.12656|-0.04834|0.048341|0.126558|0.156434|0.126558|0.048341|-0.048341|
```

A.2. FinalProb.cpp

References

[Celik, 2006] Ismail B. Celik1, Urmila Ghia, Patrick J.Roache and Christopher J. Freitas "Procedure for Esitmation and Reporting Uncertainty Due to Discretization in CFD apllications", West Virginia University, Morgantown WV, USA