Computer Based Modeling 1

Part 2 Fluid Flow

MENG11511

Harry Morgan

April 22, 2016

Abstract

The objective of the set exercise was to determine the result of using Laplace's equation to approximate the nature of fluid flow through a complex chamber. This application would be simulated and visualised through the production of a streamlined plot for a 2-D system, through a relaxation method.

Contents

List of Figures

0.1 Introduction

Laplaces Equation, $\nabla^2 u = 0$[1], is a second-order partial differential equation capable of being adapted to provide an accurate approximation of various properties within different systems. This report will address the method and outcome of applying a relaxation method to the simulation of a flow of a fluid through a chamber whilst obstructed by several cylinders across its path. The simulation produced was modeled in MathWorks Matlab ®, and produces a visible representation of the streamlines in the system. The extent of the efficiency to which this method works will also be considered, through the analysation of a convergence plot, where the size of the error is seen.

The aim of the simulation is to obtain, through iteration, a reasonable approximation of the shape and velocity of the flow around the cylinders. The dimensions of the chamber will be of specified height and length by the user, or of a preset value, stated in the beginning of the code. However, due to the simplistic nature of the problem, the depth of the chamber remains unspecified, and the problem is analysed in 2 dimensions. In the essence of the simplicity of this problem, any drag between the fluid and the surface of the cylinder and boundaries are excluded from any calculation. The flow in this problem is considered to be entirely ideal and laminar.

Whilst preset values are included in this code, it was also important that the user could apply their own variables.

0.2 Methodology

0.2.1 User Option

The code begins with the option to use default variables, which are later defined, or to set as custom. This put in place by giving the user the option, upon running the code, of "Use Preset Variables? yes/no" appearing in the command window. An input of *yes*, into the command window, will run a code with defined variables for initial velocity of flow, resolution in the x and y directions, channel dimensions and the position and size of the cylinders. A *no* response, will bring up the instructions to input the desired values in the command window. This is achieved with an **if** statement, to determine which part of the code should be executed following the corresponding yes/no response. An **elseif** component to this, allows the code to be restarted if question is answered incorrectly, the user will be responded to with an "*error please retry message*". In this code the user has the option to set the circles one-by-one, rather than defining all of the respective characteristics in separate arrays (i.e. an array for the x values of circle centres, y values of circle centres and radius of each circle), as shown below, in order to allow each circle to be defined as separate entities, providing a more intuitive input sequence for the user.

```
U=1;
X=4;
Y=2;
dx=0.0125;
dy=0.0125;
C1=[0.5,0.7,0.2];
C2=[1,1.4,0.3];
C3=[1.5,0.5,0.3];
C4=[1.9,1.1,0.2];
C5=[2.5,0.4,0.2];
C6=[2.3,1.6,0.2];
```

Figure 1: Default Settings for Code

0.2.2 Defining Required Values and Matrices

Following this loop, the circles values are put into corresponding arrays, depending on their characteristic, in order to improve the ease

of access from later parts of the code. The resolutions, determined at the start of code are used to make arrays of values, progressing at increments equal to this resolution, up to the relative length/width of the system. The vector size of this is then calculated and used to create a zeros matrix of the correct size in which to begin inputting steam function values. Before any physics is used, the arrays created, are used to create a mesh grid, which will allow the access of selection of each element within the created matrix.

0.2.3 Boundary Conditions

The first instance of the use of stream flow follows this, with the input of the stream function: $psi_y = (U * Y).'$. In coding, this uses the earlier created y vector with increments of dy, which is then transposed to give a column vector, which can be used for the creation of boundary conditions.

$$psi_y = \begin{bmatrix} 0 \\ 0.5 \\ 1 \\ 1.5 \\ 2 \end{bmatrix}$$

Figure 2: Example psi_y Transpose if $dy = 0.5$

The while loop is preceded by the defining of the starting iteration as zero, and the decided tolerance and error. The tolerance and error determine at what point of convergence is the solution at a point that can be considered as correct. The boundary conditions are created, in this case, inside the while loop, in order to keep them constant throughout iterations. Both walls and the cylinders cannot be penetrated and hence can be considered to have constant psi values within their bodies and upon their surfaces. These are however independent of each other. The bottom wall, with a $y = 0$ value, can be set at a zero value. The top, at a maximum value, defined by the result of $U*Y$ at maximum Y value. In the instance of this problem, the preset values of $Y = 2$ and $U = 1$, resulting in the maximum stream function value of 2. The cylinders are done slightly differently, in order to increase the speed of the code. Rather than approaching each circle individually, for loop is used to determine the mean value of psi values within each circles. This is achieved by

iterating the sequence in figure 3 for each circle, and taking values from the arrays defined earlier. The left and right of the chamber are defined as the same, and to be of increasing at steady intervals. These are created by selecting the desired column of the psi matrix, and inputting the transposed psi_y vector. These stated conditions provide a psi matrix of unchanging values for these parts of the simulation.

if
$i = 1 : NC; cyl = find(sqrt((x_a rray - Xcirc(1, i)).^2 + (y_a rray - Ycirc(1, i)).^2) <= Rcirc(1, i)); psi(cyl) = mean(psi(cyl));$
end

Figure 3: Circle Loop

0.2.4 Iteration

In order to fill the rest of the matrix, and create streamlines, the psi must be plotted throughout and changed accordingly to the effect of the cylinders and boundaries. This is called a relaxation method. Initially, this was achieved using a **for** loop, but in the interest of increasing the speed an efficiency of the code, this section was vectorised[2], for the values from each incremented value in dx and dy. This section again used Laplace's equation in the form.
The following few lines of code reset the psi values to those calculated by Laplace's equations, and through iteration, will become more accurate and close to a correct solution. In order to see this converge and eventually bring the loop to an end, the error value must be updated during each loop. To conclude the loop, the iteration and iteration count must be updated by adding one to each and a vector created for, in order to create a convergence plot with the updated error, which is also vectorised. The loop will only end when the error reaches a value lower than a preset tolerance, in this case 0.01.

0.2.5 Plotting

After the loop, all that remains is the plotting stage. Four plots were chosen to help the user visualise the various characteristics of the code as well as the fluid flow. First the figure is created; a full screen display with 4 subplots. The contour of psi values is

plotted to show streamlines, with 50 streamlines, as appropriate to the size of the sub-plot. The axis were made of equal size, to show a proportional simulation, and visible circles were plotted clearly, to show the location of the cylinders and help emphasize the fluid flowing around them. The steamlines were made black, as opposed to a range of colours, to present the results more clearly.
The second sub-plot is a meshgrid, presenting the psi values at each point on the 2D grid. This is shown in a colour variation to improve the clarity of the results and make it easier to visualise the circles. It also helps highlight the method of Laplaces equation, in the averaging of points across each cylinders cross section.
The third is the convergence plot, comparing the number of iterations to the size of error produced. Initially, a plot with scalar axis was produced. The nature of the rapid convergence made this hard to interpreted, hence providing requirement for a log scale on the X axis.
A quiver is produced to emphasise the results of the streamline plot, showing the direction and size of the flows velocity. Due to the nature of the resolution of the simulation, the arrows are small, thus zooming in may be necessary.

0.2.6 Error Conditions

Finally, some error conditions are included, in case of custom values which would cause a problem in the production of figures. A limit is put on how high the resolution can be, by restricting how small the dx and dy values can be (below 0.0125 is prohibited). It was considered that if the circles were outside the grid, an error message may be useful, however, it would have little effect on the fluid flow.

0.2.7 GUI

As an extension, the code was input into a custom made GUI. This allowed for a more user-friendly interface, allowing the user to press one of 3 buttons to produce the same four graphs. The first, default, inputs the default settings into the code, the second allows the input of custom settings whilst the third closes down the GUI.

0.3 Results and Discussion

0.3.1 Contour Streamline Plot

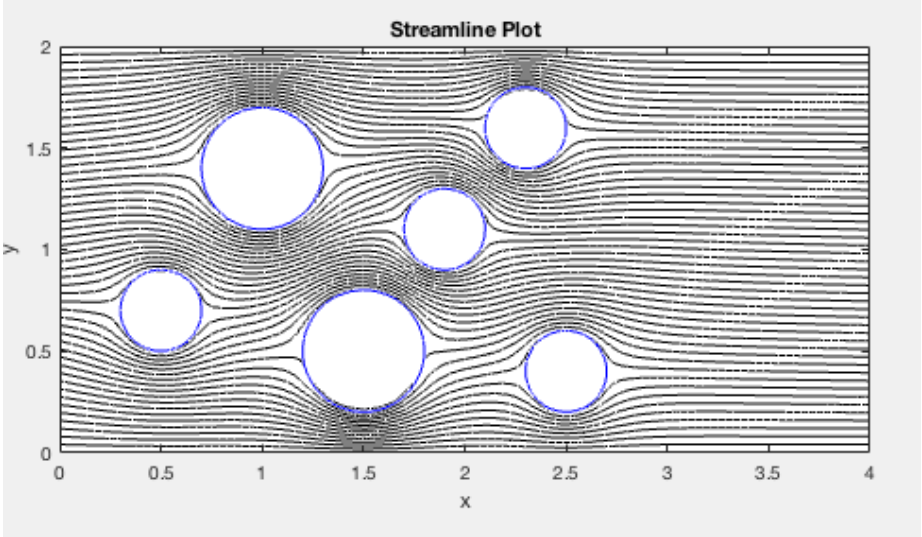

Figure 4: Contour Plot of Streamlines in Chamber

The above plot shows simulated streamlines around the cylinders obstructing the flow. Due to the boundary conditions, the input flow is identical to the output flow, and the flow does not cross any boundaries. This potentially shows an accurate approximation of laminar flow in this scenario. This flow is always fully attached to the cylinder surface and there is no boundary layer separation. As this occurs independent of speed, the assumption in this case is that the fluid is infinitely viscous[3], which is of course impossible. Due to this fact, and the likelihood of some turbulence due to skin friction drag with the boundaries, this model is unlikely to be effective in simulating any real life problems with flow at any significant velocity.

0.3.2 Mesh Grid

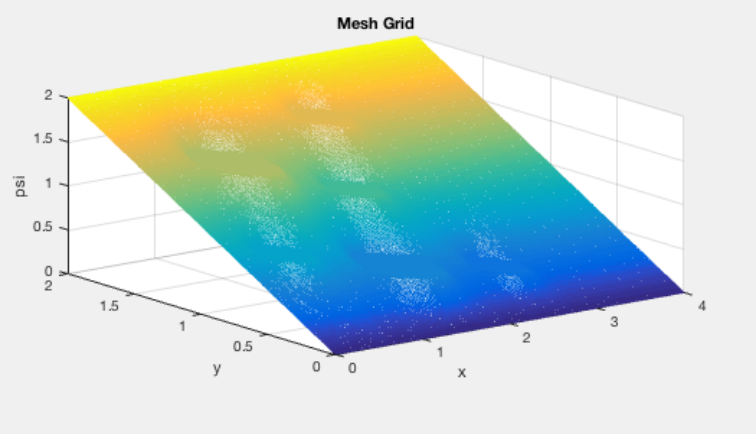

Figure 5: Meshgrid of psi(streamfunction values) in the Chamber

The meshgrid pot, whilst less useful than the steamline plot, does help the user to visualise the effect of Laplace's equation in the code. In the results from the run coding, the ability to rotate the grid enables the visualisation of the averaged-out results of the circles, and the varying psi values at each point in the system.

0.3.3 Iteration vs Error

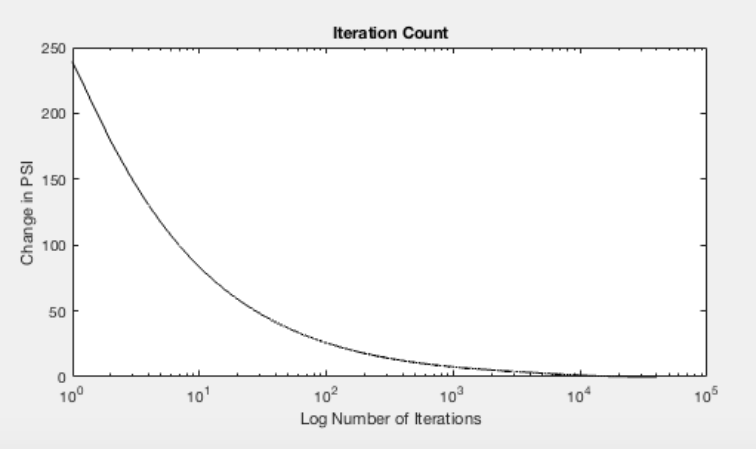

Figure 6: Log of Iteration Count against Error Size

The convergence of an iteration method like this is important in determining the accuracy of the approximation when the relaxation method is implemented. The results above show a very steep curve, even when a log scale is introduced, suggesting a very rapid convergence. However an incredibly large amount of iterations are required to reduce the error to below the set tolerance level. This shows the used method to be incredibly effective at gaining rough estimations of fluid flow, yet it seems to head asymptotically towards a correct value, hence there must be a point where the user defines the approximation to be ample in describing the flow.

0.3.4 Quiver Plot

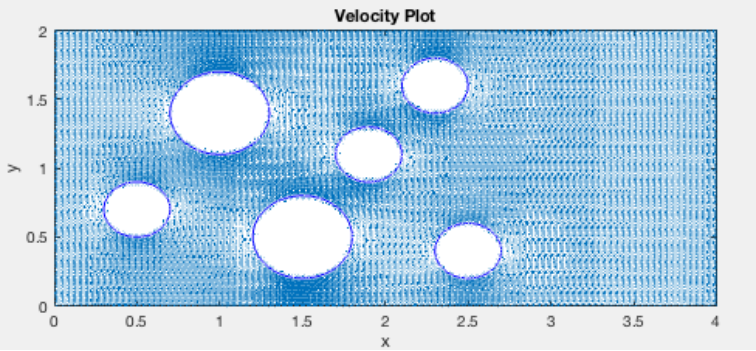

Figure 7: Log of Iteration Count against Error Size

The quiver plot is not dissimilar to the contour plot earlier analysed. This plot however does define not only the direction of the flow, but its velocity at each point, by differentiating the position against iteration number. This would be effective in presenting this, but due to the high resolution in the coding, the arrows appear very small and are not easy for the user to interpret. Although if zoomed in, they do effectively support the results found by the contour plot.

0.3.5 GUI

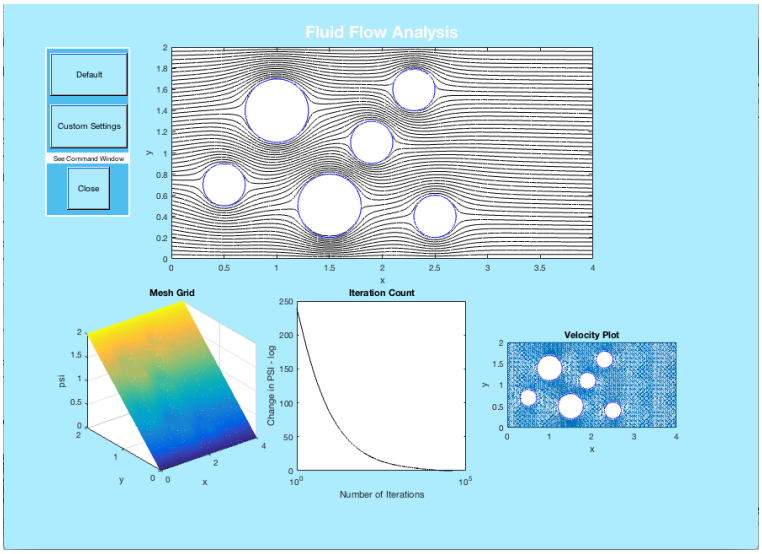

Figure 8: Log of Iteration Count against Error Size

The GUI is an effective way of making the interface for the user more intuitive and as seen above, offers the user an option to use, or not use the default settings. It also provides an easy option to close the program. These options are presented as push buttons which once pushed, run the appropriate code to their action. The GUI effectively presents the results in an attractive layout, allowing the user to analyse the effectiveness and accuracy of the relaxation method on the problem at hand.

0.4 Coding Problems and Potential Modifications

The main issue with the coding, is the time in which it takes to run. This was addressed by the addition of vectorising certain loops to prevent, iterations occuring at an unnecessary frequency. In terms of further improvements, an interpolation method would decrease the error from an earlier stage, and decrease the need for further iteration. Methods such as the Successive over-relaxation method[4]. There are also potentially more built-in functions in matlab which may help decrease the need for iteration[5]. In terms of improving the experience for the user, potentially a pop up box, with input boxes, could be added to eliminate the obscure use of the command window. This could be initially filled with the preset values, to indicate to the user the suggested dimensions and the correct method to input values.

0.5 Conclusion

- The GUI is a useful way of presenting all the information to the user, but would be improved by the addition of an option pop up box.
- The contour plot provides an effective simulation of how fluid would flow, if at very low velocity or high viscosity, but would not provide accurate solutions to real world problems where flow is not entirely ideal and laminar.
- The accuracy of the plot converges rapidly but is unlikely to ever reach an exact value.
- The speed of the program is increased as more elements are vectorised, and would be further increased if interpolation was introduced.
- The program can be adapted by the user to fit custom values for speed, dimensionsm, resolution and obstructions, making it versatile and able to be applied to different problems.
- For the model to be entirely accurate, other aspects, such as frictional forces and irregularities must be considered.

Bibliography

[1] Dr. A.G.W. Lawrie. Computer-based modelling 1.

[2] MathWorks. Vectorisation.

[3] NASA. Flow around cylinders.

[4] Wolfram Mathwolrd. Successive relaxation method.

[5] University of Oxford. Speeding up code.