Computational Partial Differential Equations
Project 4: Mastery

Tudor Trita Trita
CID 01199397

April 24, 2020

This is my own work unless stated otherwise.

Structure of Coursework:

The coursework .zip file contains the following:

- iter.f90: Numerical scheme written in Fortran 90 to be imported into Python.
- question2.py: Code used to generate figures and output for Question 2.
- question3.py: Code used to generate figures and output for Question 3.
- scheme.py: Main file containing framework for running the grid solution.
- Proj4M_01199397.pdf: (This document) Compiled LaTeX report.

Software Requirements:

The code for the coursework has been written in Python 3.7 and the following third-party packages have been used:

- Numpy (1.18.1): Used for access to fast array operations.
- Matplotlib (3.1.3): Used for plotting capabilities.

To compile the Fortran code for use in Python, run the following command in a terminal `f2py -c -m mod iter.f90` in the zip directory, or alternatively `source setup.sh`. This module can then be imported from inside Python as normal with the line `import mod`.

1 Question 1

1.1 Preliminary Work

We begin by defining the finite difference schemes used throughout the coursework. If we let $\phi_{i,j} := \phi(i,j)$ denote the discretised version of our solution and $\phi^n_{i,j}$ the discretised solution at iteration n.

We use the following second order finite difference approximations to the first, second and mixed derivatives:

$$u = \phi_x = \frac{\phi_{i+1,j} - \phi_{i-1,j}}{2\Delta x}, \quad v = \phi_y = \frac{\phi_{i,j+1} - \phi_{i,j-1}}{2\Delta y}, \tag{1}$$

$$u_x = \phi_{xx} = \frac{\phi_{i+1,j} - 2\phi_{i,j} + \phi_{i-1,j}}{(\Delta x)^2}, \quad v_y = \phi_{yy} = \frac{\phi_{i,j+1} - 2\phi_{i,j} + \phi_{i,j-1}}{(\Delta y)^2}, \tag{2}$$

We now turn to the main equation in question $(\hat{\rho}(1+u))_x + (\hat{\rho}v)_y = 0$. Using the product rule for partial derivatives, we can obtain the equation

$$\hat{\rho}_x(1+u) + \hat{\rho}_y v + \hat{\rho}(u_x + v_y) = 0, \tag{3}$$

As can be seen in the equation above, we now need the discretisations of the partial derivatives of $\hat{\rho}$, which are obtained below

$$\hat{\rho}_x = \frac{\hat{\rho}_{i+1,j} - \hat{\rho}_{i-1,j}}{2\Delta x}, \quad \hat{\rho}_y = \frac{\hat{\rho}_{i,j+1} - \hat{\rho}_{i,j-1}}{2\Delta y}. \tag{4}$$

1.2 Boundary Conditions

To be able to compute the points at the boundary, we will pad our domain with one set of ghost points and use the boundary conditions given to compute them. These conditions are the same as in coursework 2, hence we can use the same formulas as then, namely

$$\phi_{0,j} = \phi_{2,j}, \quad \phi_{N-1,j} = \phi_{N+1,j}, \quad \phi_{i,0} = \phi_{i,2}, \quad \phi_{i,N-1} = \phi_{i,N+1} \tag{5}$$

on all the boundaries excluding on the airfoil itself. On the airfoil boundary, however, we can transfer our inviscid flow tangency condition to $y = 0$ obtaining an equation for the ghost point as follows:

$$\begin{aligned} \frac{\partial \phi}{\partial y} - \left(1 + \frac{\partial \phi}{\partial x}\right) \frac{\mathrm{d}y_b(x)}{\mathrm{d}x} &= 0, \\ \implies \frac{\partial \phi}{\partial y} - \left(1 + \frac{\partial \phi}{\partial x}\right) (2\tau(1-2x)) &= 0. \end{aligned}$$

and we can solve for the fictitious points on the airfoil boundary as follows

$$\begin{aligned} \frac{\phi_{i,j+1} - \phi_{i,j-1}}{2\Delta y} - \left(1 + \frac{\phi_{i+1,j} - \phi_{i-1,j}}{2\Delta x}\right) (2\tau(1-2x_i)) &= 0, \\ \implies \phi_{i,0} = \phi_{i,2} - 2k\left(1 + \frac{\phi_{i+1,j} - \phi_{i-1,j}}{2h}\right) (2\tau(1-2x_i)). \end{aligned}$$

1.3 Numerical Scheme

The following algorithm outlines the scheme used for this coursework. This scheme allows us to compute residuals for both ϕ and ρ at each iteration if we want, thus measuring the convergence of both of these quantities.

Algorithm 1:

```
(STEP 1.1) Given initial guesses $\phi_0,\ \rho_0$ compute $\hat{\rho} = \rho_0 + 1$.
(STEP 1.2) Compute the values for $u, v$ using appropriate finite differences.
for $k = 1, \ldots,$ No. Iterations do
    (STEP 2.1) Compute the values for $\hat{\rho}_x, \hat{\rho}_y$ using appropriate finite differences.
    (STEP 2.2) Do 1 iteration of Gauss Seidel on equation (3) and apply boundary conditions.
    (STEP 2.3) Compute the values for $u, v$ using appropriate finite differences.
    (STEP 2.4) Update $\hat{\rho}$ using equation (8) in the question with new $u, v$ values.
end
(STEP 3) Output $\rho = \hat{\rho} - 1$.
```

This program is implented mostly in Python but uses Fortran90 for the computationally intense parts. We have used `f2py` to compile and wrap Fortran90 programs for use in Python as a regular module whilst maintaining compiled speeds. These are the files `scheme.py, iter.f90`. This is because Python introduces significant overheads when executing loops since each line must be interpreted at every iteration. In Fortran, the compiler is able to optimise loops and a great speedup of approximately two orders of magnitude is achieved, allowing us to run the program using finer grids as well as more iterations.

It remains to explain how to do the Gauss-Seidel iteration in (STEP 2.2). Since we already have the values (matrices) for $u, v, \hat{\rho}_x, \hat{\rho}_y$, we can discretise equation (3) using finite differences to obtain

$$\hat{\rho}_x(1+u) + \hat{\rho}_y v + \hat{\rho}\left(\frac{\phi_{i+1,j} - 2\phi_{i,j} + \phi_{i-1,j}}{(\Delta x)^2} + \frac{\phi_{i,j+1} - 2\phi_{i,j} + \phi_{i,j-1}}{(\Delta y)^2}\right) = 0, \tag{6}$$

$$\Longrightarrow 2\hat{\rho}\left(\frac{1}{(\Delta x)^2} + \frac{1}{(\Delta y)^2}\right)\phi_{i,j} = \hat{\rho}\left(\frac{\phi_{i+1,j} + \phi_{i-1,j}}{(\Delta x)^2} + \frac{\phi_{i,j+1} + \phi_{i,j-1}}{(\Delta y)^2)}\right) + \hat{\rho}_x(1+u) + \hat{\rho}_y v. \tag{7}$$

Hence, if we let $\alpha = 2\hat{\rho}\left(1/(\Delta x)^2 + 1/(\Delta y)^2\right)$, then the Gauss-Seidel iteration is

$$\phi_{i,j}^{n+1} = \frac{1}{\alpha}\left[\hat{\rho}\left(\frac{\phi_{i+1,j}^n + \phi_{i-1,j}^n}{(\Delta x)^2} + \frac{\phi_{i,j+1}^n + \phi_{i,j-1}^n}{(\Delta y)^2)}\right) + \hat{\rho}_x(1+u) + \hat{\rho}_y v\right]. \tag{8}$$

With this scheme, we can compute the residuals for both ϕ, ρ after each iteration if we wish.

1.4 Grid Independence

Following investigations similar to those undertaken in CW2, we have established the fact that the results are grid independent since the solutions that we obtain converge to the same values no matter what the coarseness of the grid is, provided it is fine enough to pick up on values at around the points $x = 0, x = 1$. The speed of convergence does depend on the grid size. We found that there is an inverse relationship between number of points in our grid and speed of convergence.

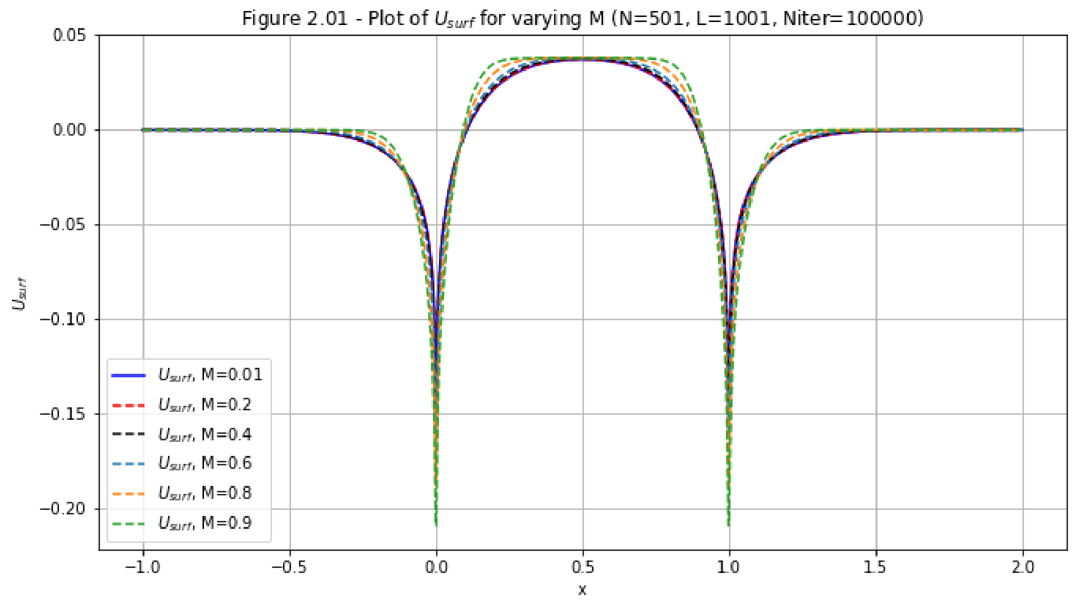
2 Question 2

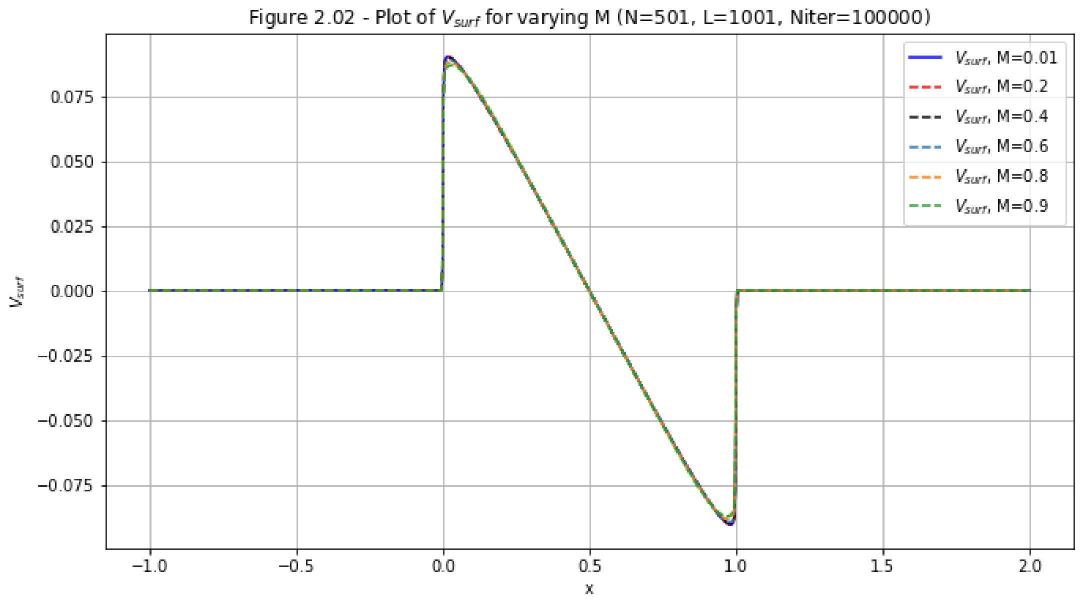
For this question, we will fix the following parameters:

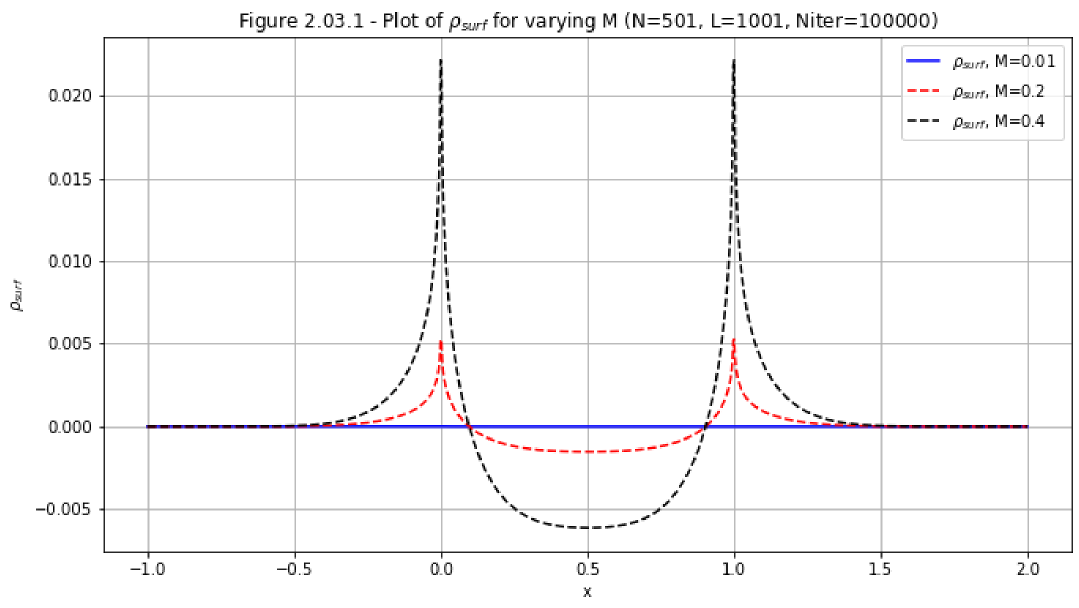
- $N = 501, L = 1001$ giving us a fine grid size.
- $q = 1, s = 2, r = 1, \tau = 0.05, \gamma = 1.4$

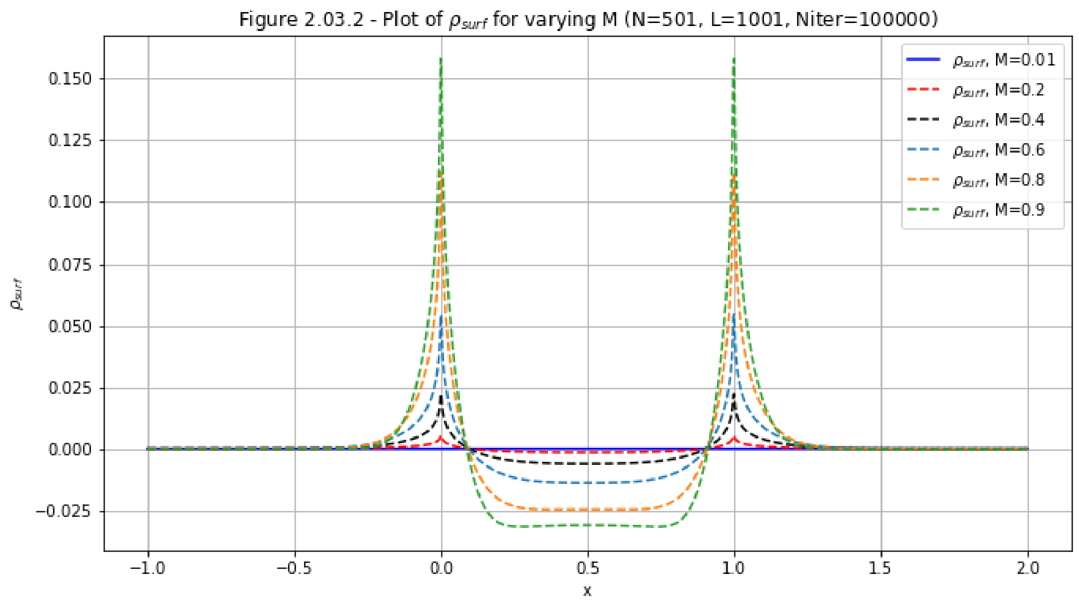
We will therefore focus our analysis on the behaviour of the solution as we vary M.

2.1 Line Plots of u, v, ρ on Airfoil Surface $y = 0$



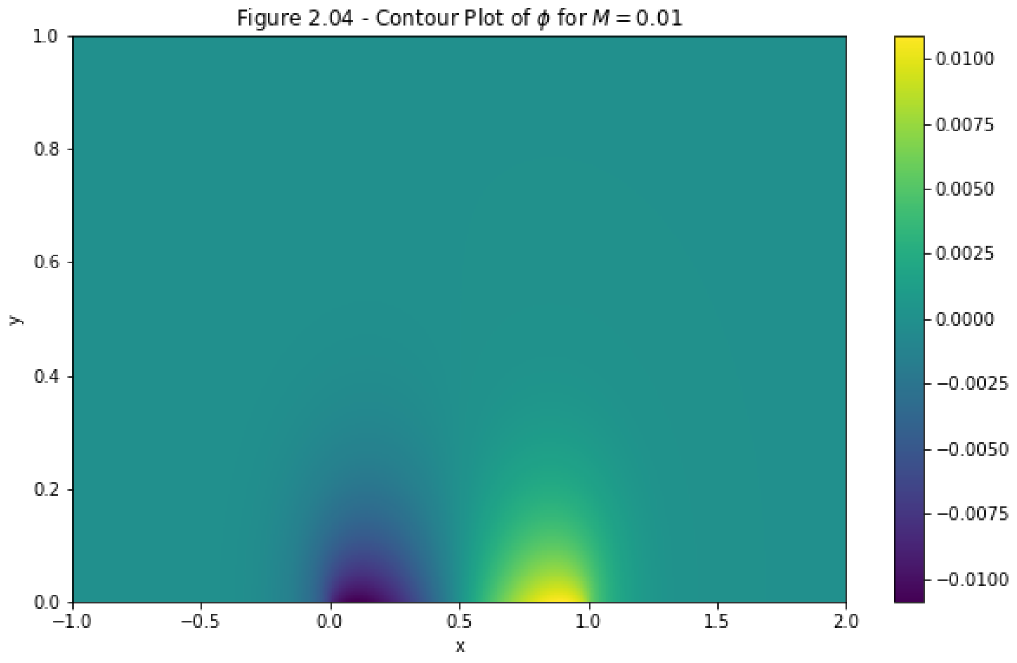


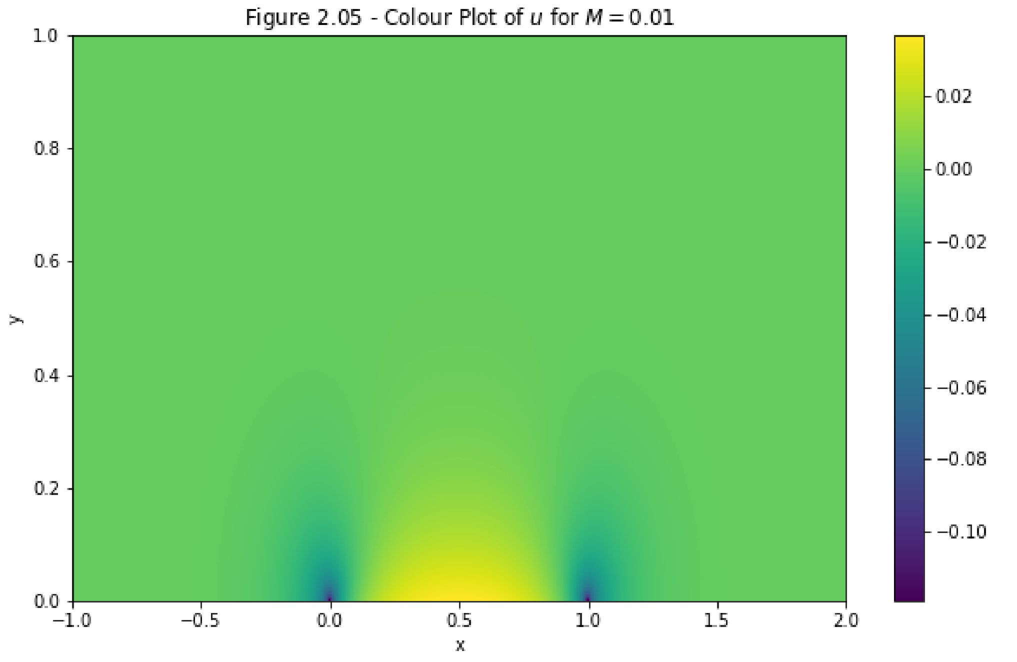


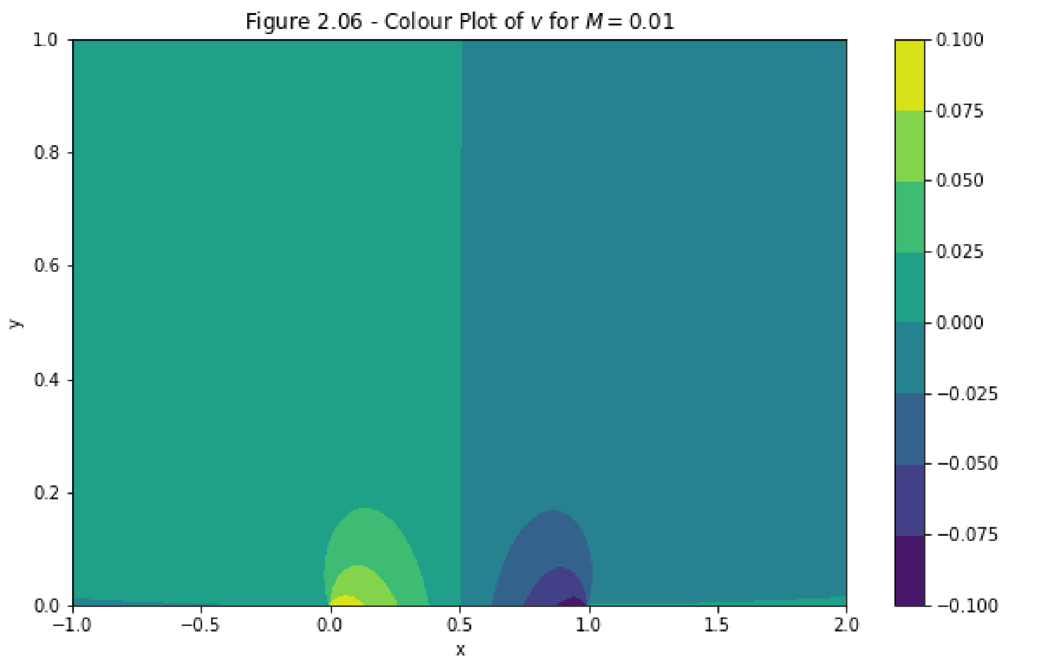


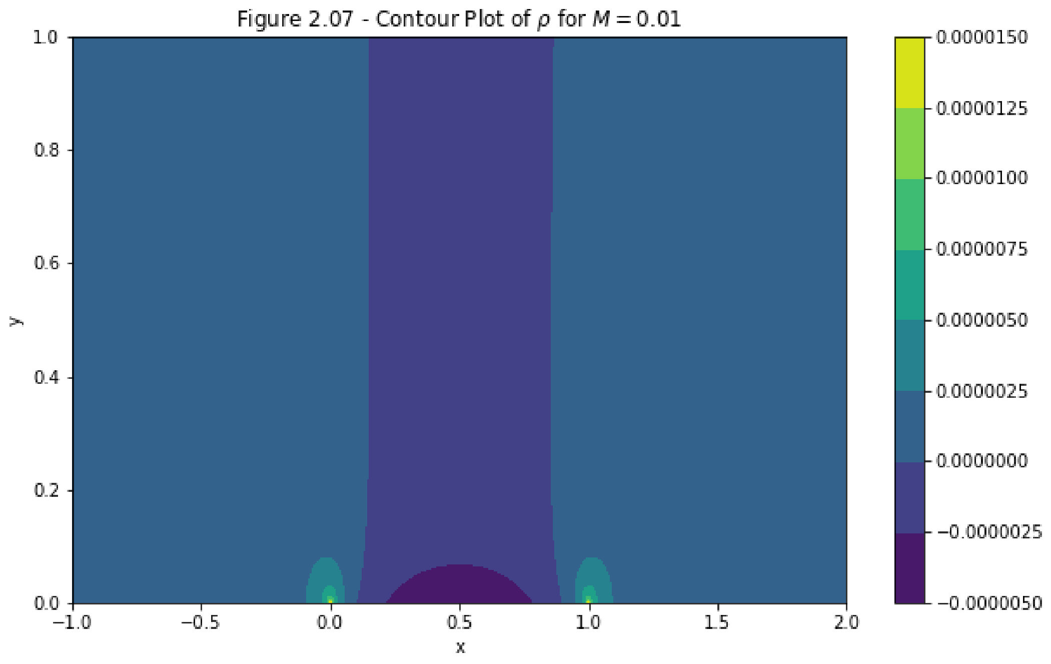
As can be seen from the figures 2.03.1 and 2.03.2, the density ρ is nearly flat when $M = 0.01$. This makes sense, since the ρ term has nearly vanished for these values as they are very small. As we can see, as we increase M, the density ρ becomes more pronounced. At the edges of the airfoil, we see that ρ is highest and contains two peaks at the points $x = 0, x = 1$. We also see that there is a minimum value in ρ at around the point $x = 0.5$. Physically, the minimum is due to the fact that the horizontal airspeed is fastest at this point, and therefore the density will be smallest.

2.2 Contour Plots of ϕ, u, v, ρ over entire field for $M = 0.01$



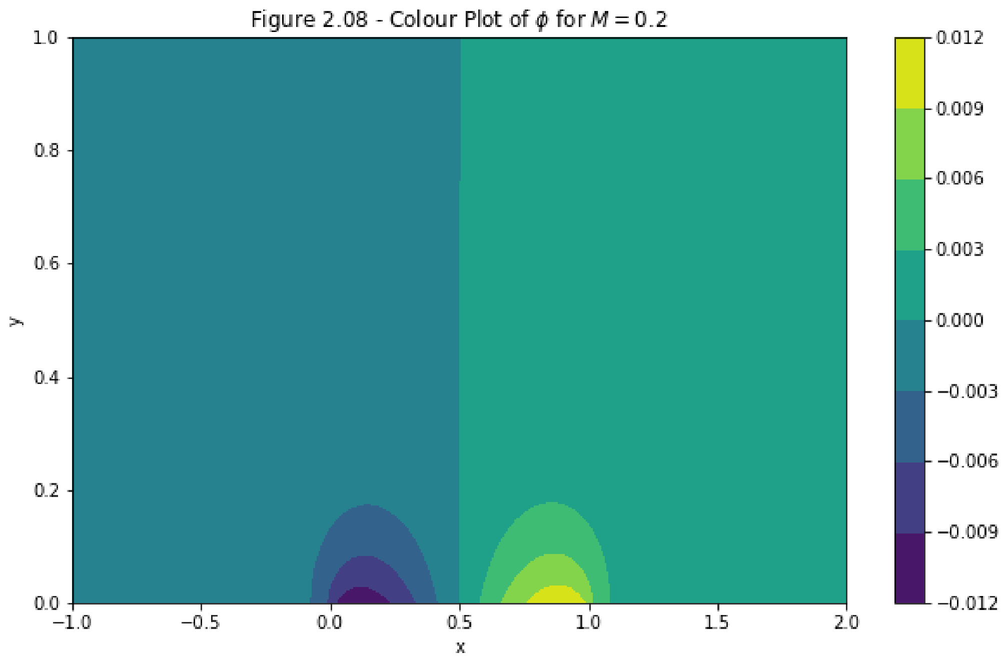


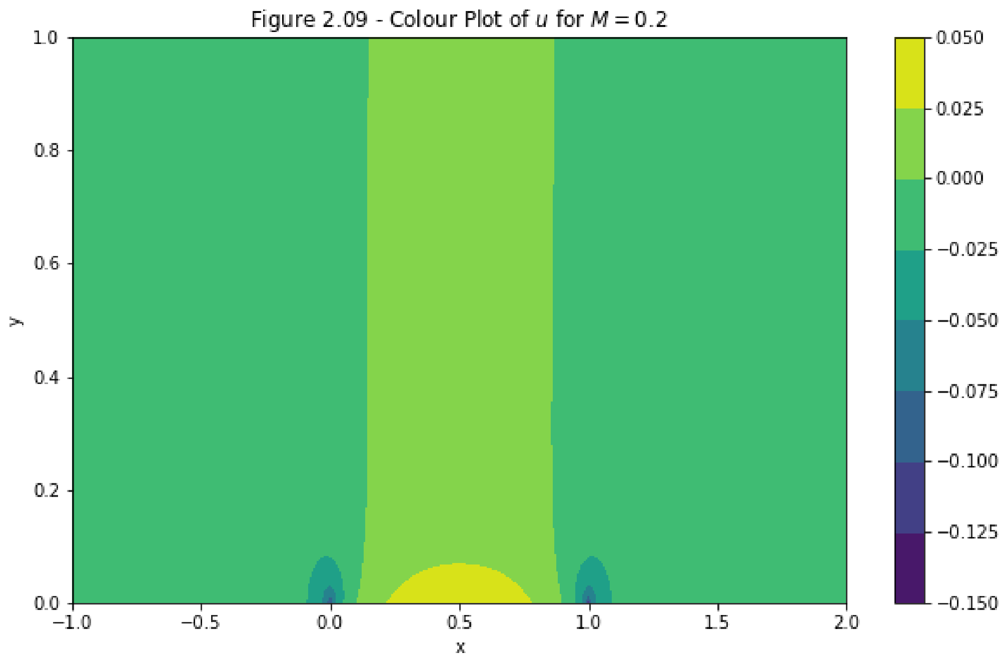


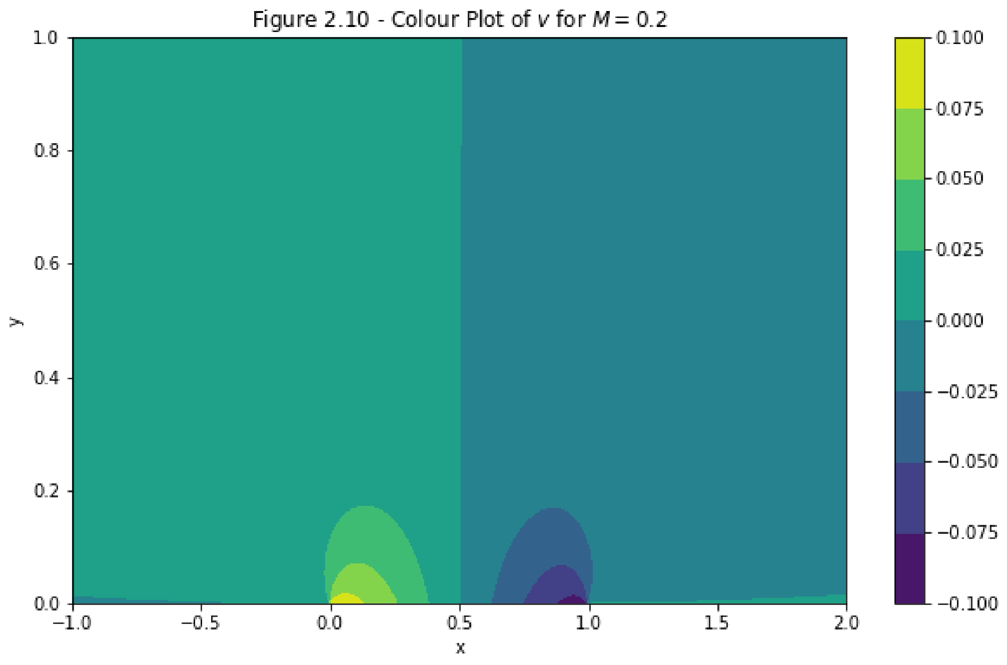


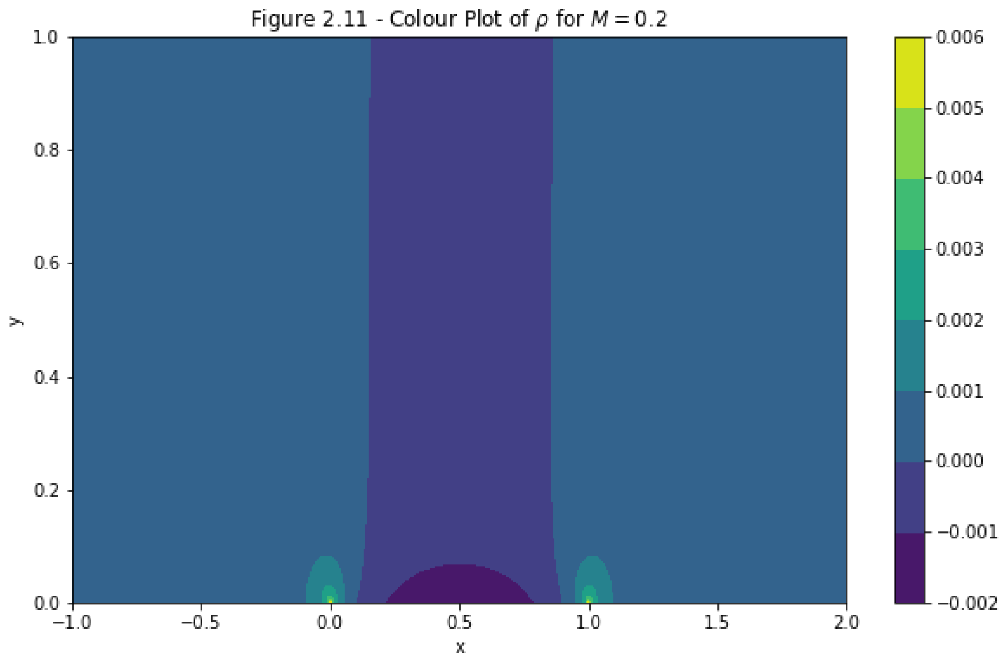
The contour plots above provide information about the solution for $M = 0.01$.

2.3 Contour Plots of ϕ, u, v, ρ over entire field for $M = 0.2$



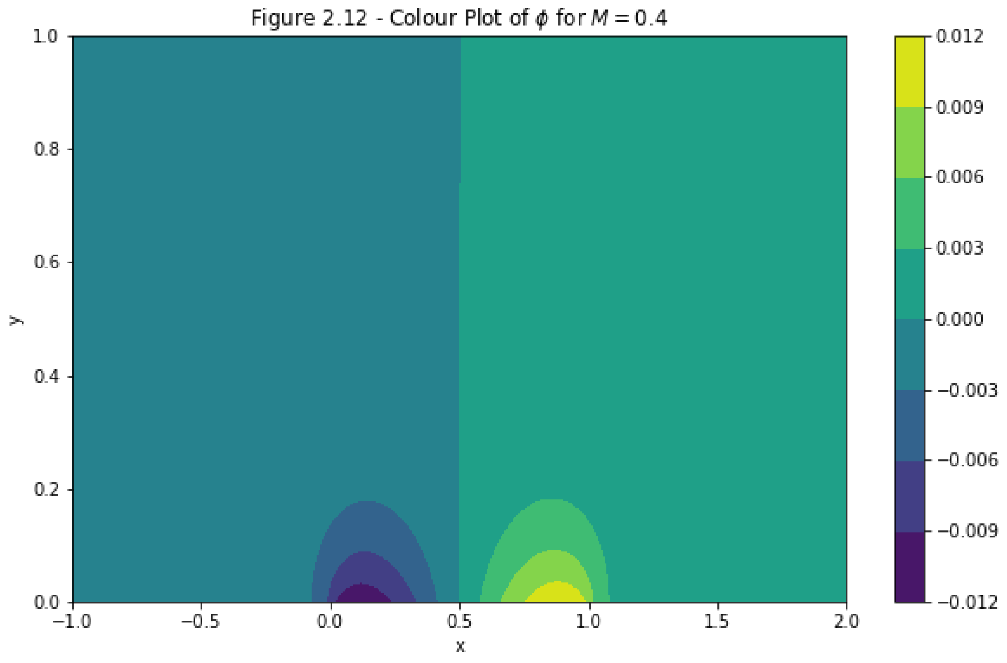






The contour plots above provide information about the solution for $M = 0.2$.

2.4 Contour Plots of ϕ, u, v, ρ over entire field for $M = 0.4$



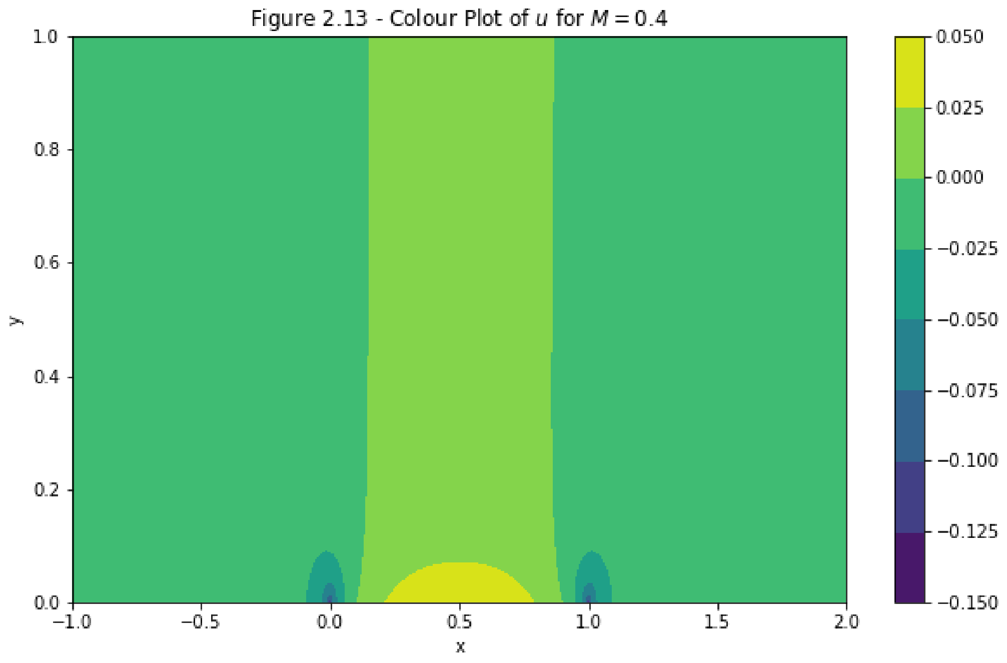


Figure 2.14 - Colour Plot of v for $M = 0.4$

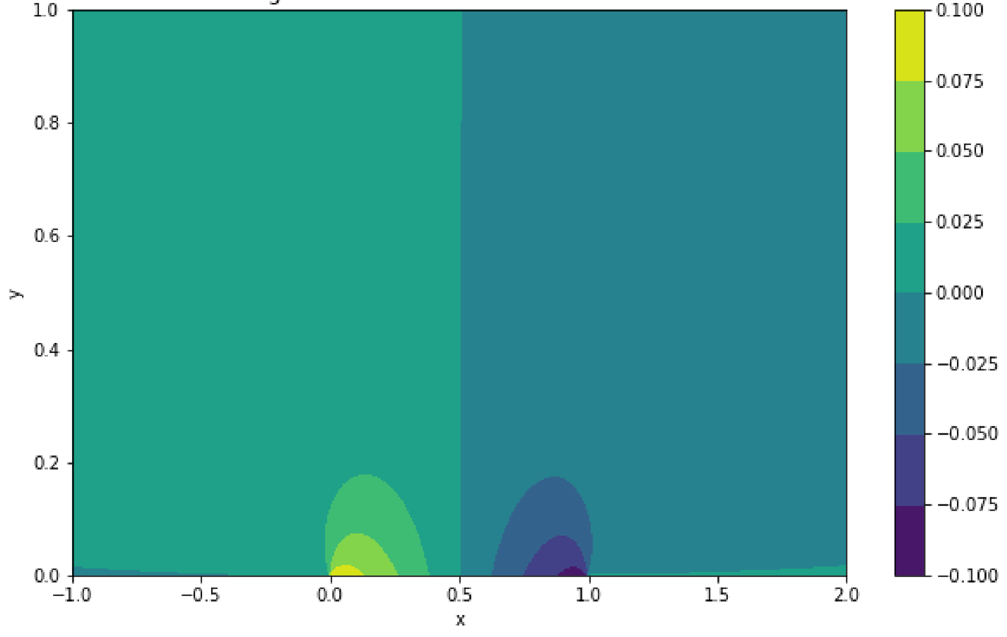
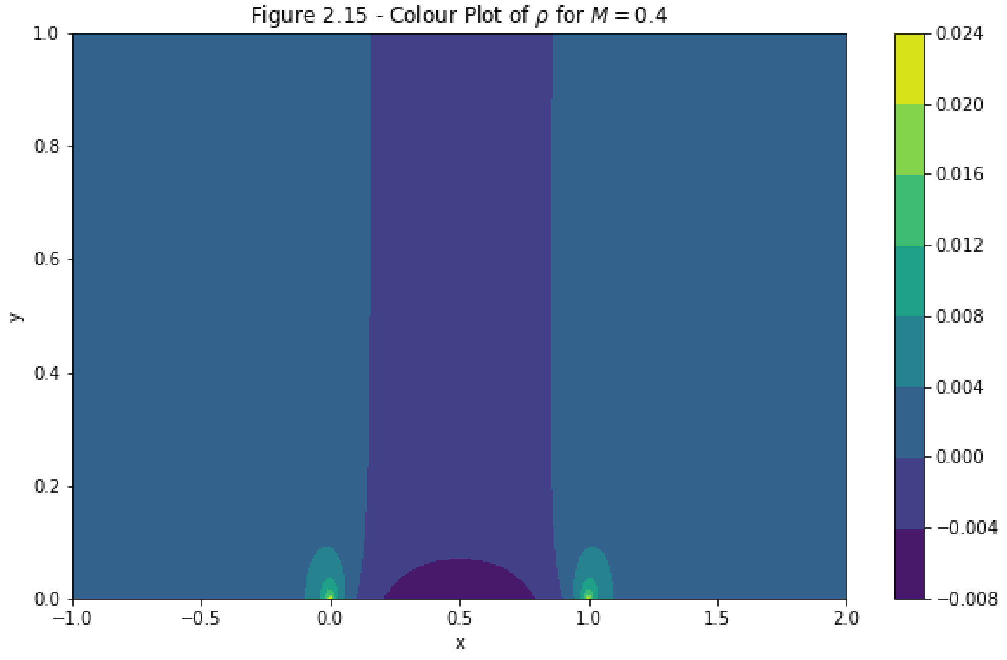


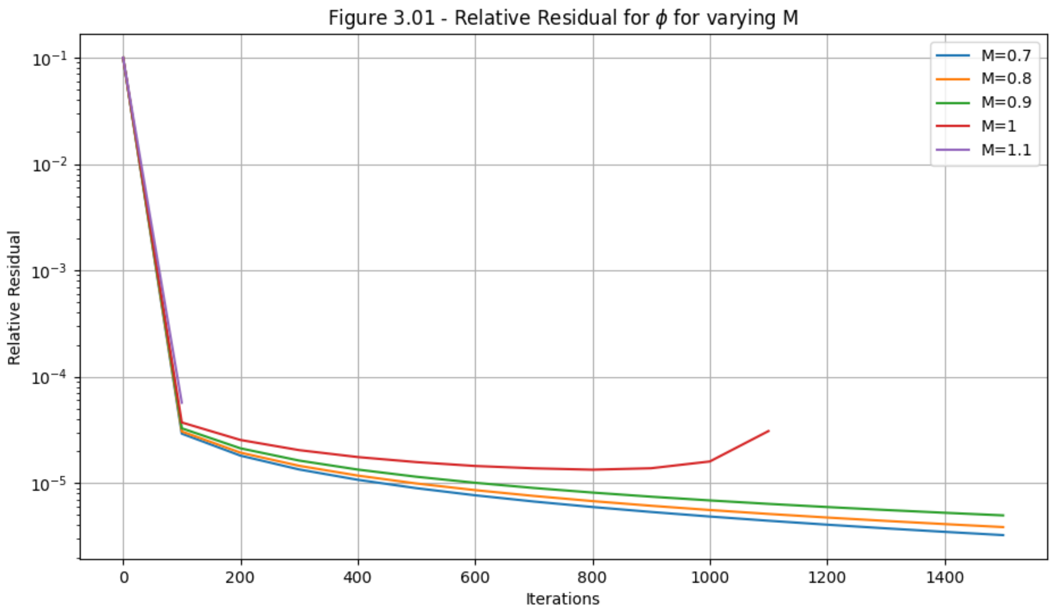
Figure 2.15 - Colour Plot of ρ for $M = 0.4$

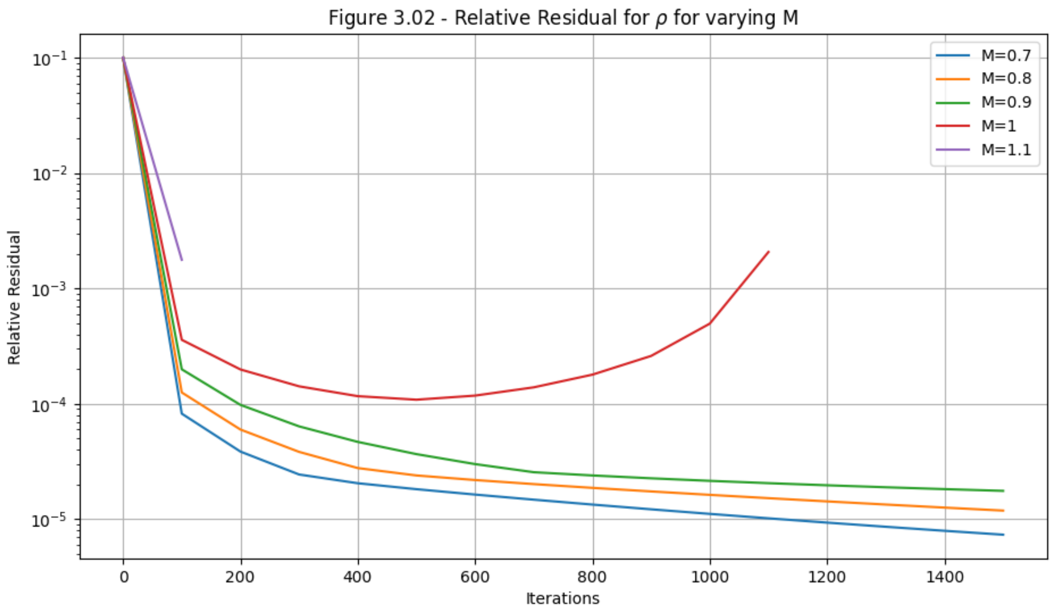


The contour plots above provide information about the solution for $M = 0.4$.

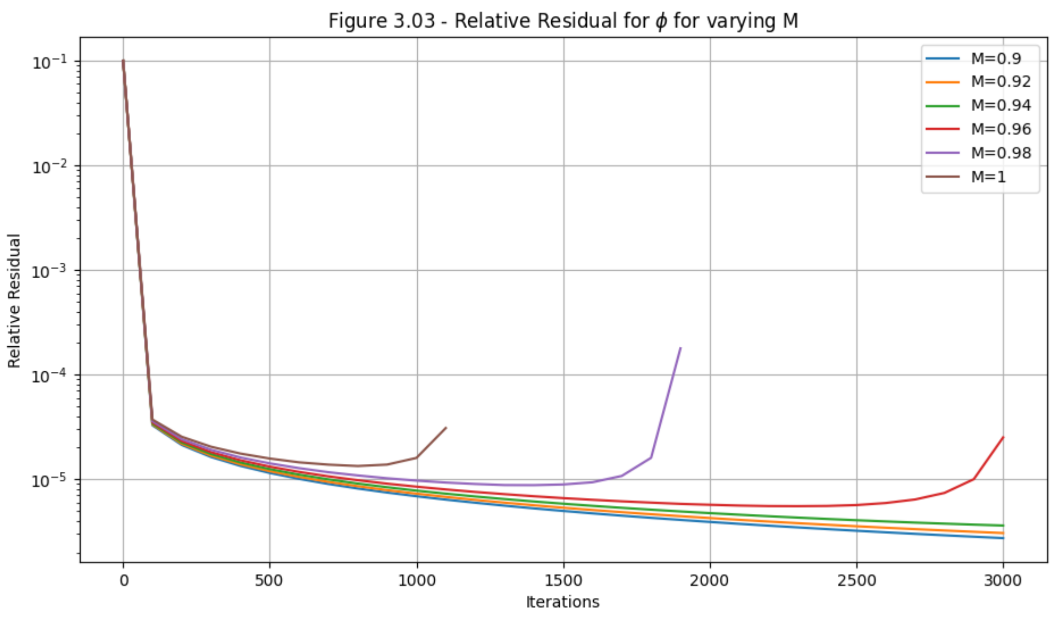
3 Question 3

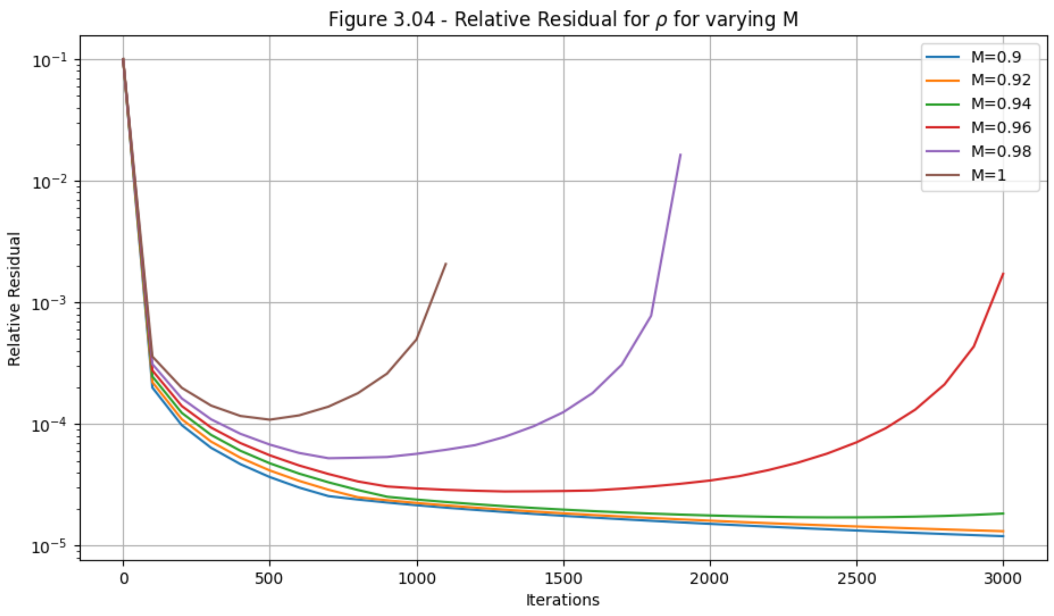
The following two plots show the convergence properties for ϕ, ρ for varying M:





From these plots, we have empirically shown that the scheme does not converge for $M \geq 1$ since the relative errors stop converging; they do not appear on the plot since they become too great. We now investigate at what value of $0.9 \leq M \leq 1$ this happens:





From these figures, it is clear that the largest value of M that the scheme will converge for is approximately equal to $M = 0.9$. We know that the Mach number is the ratio of the speed of the fluid flowing past the airfoil divided by the speed of sound in that fluid. From a physical point, we know that we encounter subsonic flow for $M \leq 1$, sonic flow at $M = 1$ and supersonic flow for $M \geq 1$. Theoretically, the value of M governs what type of PDE equation (6) in the notes is. For $M \leq 0.9$, we see that the PDE is elliptic, and therefore there are no characteristic curves, hence we can iterate using Gauss-Seidel. For $M \geq 0.9$, our equation changes type and becomes hyperbolic, and so we would need to solve in time, otherwise the iteration is unstable, which is exactly what we find empirically. Moreover, we find that for $M \geq 1$, equation (8) in the question becomes complex at some points in our grid. To overcome this, we modify this equation to deal with supersonic flow.