Computational Fluid Dynamics: Project 2

Hari Balaji (22110092)

Department of Mechanical Engineering

Prof. Dilip Srinivas Sundaram

1. Problem Statement

Computational Solution of Viscous Flow over a Flat Plate

Governing Equations:
Continuity Equation:

$$\frac{\partial u}{\partial x} + \frac{\partial v}{\partial y} = 0 \tag{1}$$

Momentum Equation:

$$u\frac{\partial u}{\partial x} + v\frac{\partial u}{\partial y} = \nu\frac{\partial^2 u}{\partial y^2} \tag{2}$$

Here, ν is the kinematic viscosity of the fluid.
The boundary layer equations need to be solved for $\mathrm{Re}_L = 10^4$.

$$\mathrm{Re}_L = \frac{U_\infty L}{\nu} \tag{3}$$

Here, L is the plate length and U_∞ is the freestream velocity.
Discretisation method to be used for the discretisation of the PDEs: Finite Difference Method.
The PDEs need to be solved using the following schemes, and so the project has been divided into three parts as shown below:

- **Part 1:** Euler-Explicit Scheme
- **Part 2:** Euler-Implicit Scheme
- **Part 3:** Crank-Nicolson Scheme

Simulation Requirements:

1. Computation of the x-velocity (u) and y-velocity (v) fields and show them as contour plots, along with plotting of the normalised x-velocity (F') as a function of the similarity variable (η) as a line plot and comparison of the results with the Blasius solution.

$$F'(\eta) = \frac{u}{U_\infty} \tag{4}$$

$$\eta = y\sqrt{\frac{U_\infty}{\nu x}} \tag{5}$$

2. Computation of the boundary layer thickness and plotting of the variation of the boundary layer thickness with the x-coordinate and comparison of the results with the Blasius flat plate boundary layer solution below:

$$\frac{\delta}{x} = \frac{4.91}{\sqrt{\mathrm{Re}_x}} \tag{6}$$

2. Mesh Details and Discretisation Approach

2.1. Mesh Details

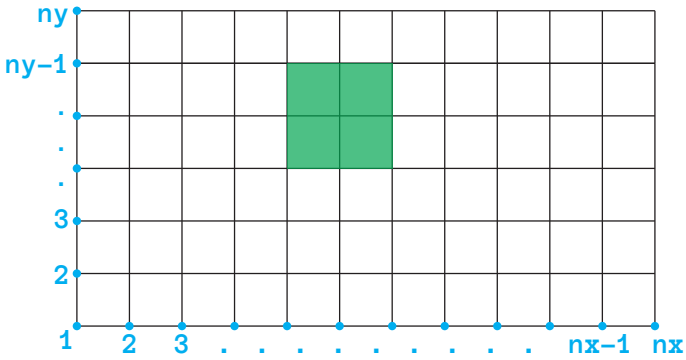

Figure 1. *Mesh of the Computational Domain*

The mesh has been defined in a rectangular domain, where the bottom surface is the surface where the formation of the boundary layer occurs. The left surface is the inlet, where the fluid enters the domain.
The number of grid points in the x and y-directions are nx and ny respectively.
To perform the discretisation, we need to consider an arbitrary point inside the domain. Consider the region in the domain highlighted in *green*:

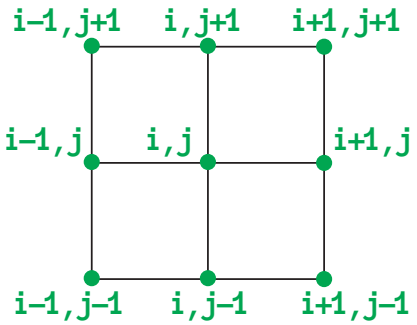

Figure 2. *Nomenclature of Points in the Mesh*

For the point i, j, i refers to the column number and j refers to the row number inside the domain. The various points around it are labeled since they are needed for the calculation of the state of the system based on the discretisation used.
It is important to note that in MATLAB, we use a matrix to obtain the solution, and hence the indices need to be correctly coded to ensure no discrepancies in the solution.

2.2. Boundary Conditions

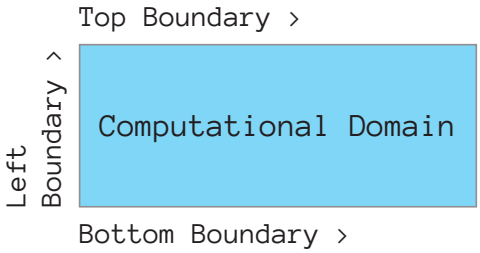

Figure 3. *Computational Domain and Known Boundaries*

As seen from figure 3, there are three known boundaries for the computational domain where we solve the boundary layer equations.

- *The top boundary* will have the boundary condition $u_{i,j} = U_\infty$, since this is the set of points in the domain where the boundary layer has separated completely and the viscous effects have disappeared.
- *The left boundary* will have the boundary condition $u_{i,j} = U_\infty$ and $v_{i,j} = 0$, since this is the freestream velocity with which the fluid enters the computational domain.
- *The bottom boundary* will have the boundary condition $u_{i,j} = 0$, since this is the set of points in the domain which are closest to the wall due to which the viscous effects are maximum, imposing a no-slip condition.

2.3. Variables and Constants Utilised

Domain Length (L)	1 m
Domain Height (H)	0.2 m
Number of grid points in the x-direction (nx)	10000
Number of grid points in the y-direction (ny)	200
Reynolds Number (Re)	10^4
Freestream Velocity (U_∞)	1 m/s
Coefficient of Viscosity (ν)	10^{-4} Pa-s

Table 1. *Caption*

2.4. Approach for Discretising the Equations

Part 1 of the project requires us to solve the boundary layer equations using the Euler-explicit scheme and *Part 2* requires us to solve the equations using the Euler-implicit scheme.
The explicit scheme calculates the values of the solution at the required grid-point using the values of the solution at the previous grid-point.
The implicit scheme calculates the solution of the current state of the system using the values at the current and next state of the system (whose values are guessed), and then the solution is iterated upon until convergence is achieved. [3]
Part 3 of the project requires us to solve the boundary layer equations using the Crank-Nicolson scheme.
The Crank-Nicolson scheme is a special case of the implicit scheme, and it involves calculating the state of the system between two states. It is an equivalent method to the implicit midpoint method. This method is better than the implicit scheme, since it gives second-order convergence in time. [2]

3. Discretised Equations

Part 1: Euler-Explicit Scheme

Consider equations 1 and 2. Applying the finite difference method for forward marching in the x-direction to obtain the first and second derivatives at i, j,

$$\left(\frac{\partial u}{\partial x}\right)_{i,j} = \frac{u_{i+1,j} - u_{i,j}}{\Delta x}$$

$$\left(\frac{\partial v}{\partial y}\right)_{i,j} = \frac{v_{i,j+1} - v_{i,j-1}}{2\Delta y}$$

$$\left(\frac{\partial u}{\partial y}\right)_{i,j} = \frac{u_{i,j+1} - u_{i,j-1}}{2\Delta y}$$

$$\left(\frac{\partial^2 u}{\partial y^2}\right)_{i,j} = \frac{u_{i,j+1} - 2u_{i,j} + u_{i,j-1}}{\Delta y^2}$$

The forward difference method has been applied in the x-direction and the central difference method has been applied in the y-direction.
Substituting the partial derivatives obtained above into equations 1 and 2, the following equations are obtained:

$$\frac{u_{i+1,j} - u_{i,j}}{\Delta x} + \frac{v_{i,j+1} - v_{i,j-1}}{2\Delta y} = 0 \tag{7}$$

$$u_{i,j}\left(\frac{u_{i+1,j} - u_{i,j}}{\Delta x}\right) + v_{i,j}\left(\frac{u_{i,j+1} - u_{i,j-1}}{2\Delta y}\right) = \nu\left(\frac{u_{i,j+1} - 2u_{i,j} + u_{i,j-1}}{\Delta y^2}\right) \tag{8}$$

Simplifying the above equations, we get:

$$v_{i,j+1} = u_{i,j}\frac{2\Delta y}{\Delta x} - u_{i+1,j}\frac{2\Delta y}{\Delta x} + v_{i,j-1} \tag{9}$$

$$u_{i+1,j} = \nu\left(\frac{u_{i,j+1} - 2u_{i,j} + u_{i,j-1}}{\Delta y^2}\right)\frac{\Delta x}{u_{i,j}} - v_{i,j}\left(\frac{u_{i,j+1} - u_{i,j-1}}{2\Delta y}\right)\frac{\Delta x}{u_{i,j}} + u_{i,j} \tag{10}$$

The value of $u_{i+1,j}$ obtained from equation 10 can be put into equation 9 to obtain the value of $v_{i,j+1}$. These values can consequently be used to calculate the next set of values of u and v, and this can done until the values of u and v are known for all the points.

Part 2: Euler-Implicit Scheme

Consider equations 1 and 2. Applying the finite difference method for forward marching in the x-direction to obtain the first and second derivatives at $i+1, j$,

$$\left(\frac{\partial u}{\partial x}\right)_{i+1,j} = \frac{u_{i+1,j} - u_{i,j}}{\Delta x}$$

$$\left(\frac{\partial v}{\partial y}\right)_{i+1,j} = \frac{v_{i+1,j+1} - v_{i+1,j-1}}{\Delta y}$$

$$\left(\frac{\partial u}{\partial y}\right)_{i+1,j} = \frac{u_{i+1,j+1} - u_{i+1,j-1}}{2\Delta y}$$

$$\left(\frac{\partial^2 u}{\partial y^2}\right)_{i+1,j} = \frac{u_{i+1,j+1} - 2u_{i+1,j} + u_{i+1,j-1}}{\Delta y^2}$$

The backward difference method has been applied in the x-direction (for the $i+1$th column in the x-march) and the y-direction.
Substituting the partial derivatives obtained above into equations 1 and 2 and simplifying, the following equations are obtained:

$$v_{i+1,j} = u_{i,j}\frac{\Delta y}{\Delta x} - u_{i+1,j}\frac{\Delta y}{\Delta x} + v_{i,j} \tag{11}$$

$$u_{i+1,j}\left(\frac{u_{i,j}}{\Delta x} + \frac{2\nu}{(\Delta y)^2}\right) = \frac{(u_{i,j})^2}{\Delta x} + u_{i+1,j+1}\left(\frac{\nu}{(\Delta y)^2} - \frac{v_{i,j}}{2\Delta y}\right) + u_{i+1,j-1}\left(\frac{\nu}{(\Delta y)^2} + \frac{v_{i,j}}{2\Delta y}\right) \tag{12}$$

The above two equations can be solved for $u_{i+1,j}$ and $v_{i+1,j}$ iteratively. Due to the non-linearity of the Navier-Stokes equations, we substitute $u_{i,j}$ and $v_{i,j}$ in the momentum and continuity equations to obtain the next set of values for u and v. This scheme uses iteration to obtain the solutions of u and v, i.e. until convergence is achieved.

Part 3: Crank-Nicolson Scheme

This method is an implicit method which is second-order accurate in time since it is a combination of the forward and backward Euler methods. It uses the state of the system at the $n+1$th and the nth points to obtain the current state. It takes the average of the values of the derivatives obtained by the methods to calculate the state at the $n+1/2$th point, making it similar to a central difference approximation. [1]

$$\left(\frac{\partial u}{\partial x}\right)_{i,j} = \frac{u_{i,j+1} - u_{i,j}}{\Delta x} \tag{13}$$

$$\left(\frac{\partial v}{\partial y}\right)_{i,j} = \frac{v_{i,j+1} - v_{i,j}}{\Delta y}$$

$$\left(\frac{\partial u}{\partial y}\right)_{i+1/2,j} = \frac{1}{2}\left[\left(\frac{\partial u}{\partial y}\right)_{i,j} + \left(\frac{\partial u}{\partial y}\right)_{i+1,j}\right]$$

$$\left(\frac{\partial^2 u}{\partial y^2}\right)_{i+1/2,j} = \frac{1}{2}\left[\left(\frac{\partial^2 u}{\partial^2 y}\right)_{i,j} + \left(\frac{\partial^2 u}{\partial^2 y}\right)_{i+1,j}\right]$$

Replacing the derivatives in the above equations with the difference expressions, we get:

$$\left(\frac{\partial u}{\partial y}\right)_{i+1/2,j} = \frac{u_{i+1,j+1} - u_{i+1,j-1} + u_{i,j+1} - u_{i,j-1}}{4\Delta y} \tag{14}$$

$$\left(\frac{\partial^2 u}{\partial y^2}\right)_{i+1/2,j} = \frac{u_{i+1,j+1} - 2u_{i+1,j} + u_{i+1,j-1} + u_{i,j+1} - 2u_{i,j} + u_{i,j-1}}{4\Delta y} \tag{15}$$

Now, substituting equations 13, 14 and 15 into the momentum equation (equation 2), we get the discretised momentum equation as:

$$P_{i+1/2,j}u_{i+1,j+1} + Q_{i+1/2,j}u_{i+1,j} + R_{i+1/2,j}u_{i+1,j-1} + S_{i+1/2} = 0 \tag{16}$$

Where:

$$P_{i+1/2,j} = \frac{1/\Delta y - v_{i,j}/2}{2\Delta y} \qquad Q_{i+1/2,j} = -[1/(\Delta)^2 + u_{i,j}/\Delta' x]$$

,

$$R_{i+1/2,j} = \frac{1/\Delta y - v_{i,j}/2}{2\Delta y} \qquad \bar{Q}_{i+1/2,j} = -[1/(\Delta y)^2 + u_{i,j}/\Delta x]$$

$$S_{i+1/2,j} = P_{i+1/2,j}u_{i,j+1} + \bar{Q}_{i+1/2,j}u_{i,j} + R_{i+1/2,j}u_{i,j-1}$$

We get the discretised continuity equation as:

$$v_{i+1,j} = u_{i,j}\frac{\Delta y}{\Delta x} - u_{i+1,j}\frac{\Delta y}{\Delta x} + v_{i,j} \tag{17}$$

4. Solution Methodology

Part 1: Euler-Explicit Scheme

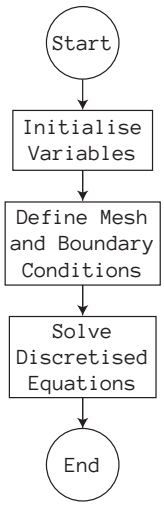

Figure 4. *Euler-Explicit Scheme*

```
for j = 2:nx
    for i = 2:ny-1
        % Calculate u(i, j)
        u(i, j) = u(i, j-1) + nu * (u(i+1, j-1) - 2*u(i, j-1) + u(i-1, j-1)) / u(i, j-1) * dx / (dy^2) - (u(i+1, j-1) - u(i-1, j-1)) / 2 * v(i, j-1) / u(i, j-1) * dx / dy;
        v(i, j) = v(i-1, j) - 0.5 * (u(i, j) - u(i, j-1) + u(i-1, j) - u(i-1, j-1)) * dy / dx;
    end
end
```

Code 1. *Euler-Explicit Scheme Logic Loop*

1. Discretise the given set of boundary layer equations (equations 1 and 2) as explained in *section 3* to obtain equations 9 and 10.
2. Define the grid, and initialise the various parameters required to solve the discretised equations.

3. Initialise the solution matrices for u and v, and set the boundary conditions for them.

4. Initialise the loops to solve the discretised equations. There will be a nested loop as in seen in code 1. The outer loop traverses through the columns and the inner loop traverses through the rows.

5. After obtaining the solution, plot the x-velocity (u) and y-velocity (v) fields as contour plots, and also plot the other required parameters such as the normalised velocity (F') as a function of the similarity variable (η) and compare it with the theoretical Blasius solution.

6. Plot the variation of the thickness of the boundary layer and compare it with the solution of the Blasius flat-plate boundary layer equation (equation 6).

Part 2: Euler-Implicit Scheme

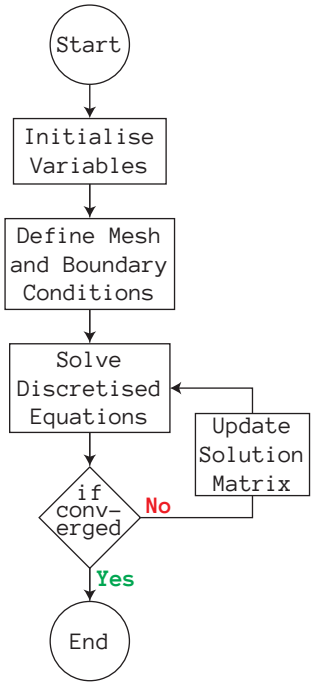

Figure 5. *Euler-Implicit Scheme*

```
for j = 2:nx
    u(:,j) = u(:,j-1);
    p = 0;
    while (true)
        u_old = u(:,j);
        for i = 2:ny-1
            % Calculate u(i, j)
            u(i,j) = ((u(i,j-1)^2 / dx) - u(i+1,
    j) * (v(i,j-1) / (2 * dy) -nu / dy^2) + u(i
    -1,j) * (v(i,j-1) / (2 * dy) + nu / dy^2)) /
     (u(i,j-1) / dx + 2 * nu / dy^2);
        end
        residual = sqrt(sum((u(:,j) - u_old).^2)
    );
        p = p + 1;
        if(residual < tolerance || p >= max_iter
    )
            break
        end
    end
    v(:,j) = v(:,j-1);
```

```
    p = 0;
    while (true)
        v_old = v(:,j);
        for i = 2:ny-1
            % Calculate v(i, j)
            v(i,j) = v(i-1,j) + (u(i,j) - u(i,j
    -1)) * dy / dx;
        end
        residual = sqrt(sum((v(:,j) - v_old).^2)
    );
        p = p + 1;
        if(residual < tolerance || p >= max_iter
    )
            break;
        end
    end
end
```

Code 2. *Euler-Implicit Scheme Logic Loop*

1. Discretise the given set of boundary layer equations (equations 1 and 2) as explained in *section 3* to obtain equations 11 and 12.

2. Define the grid, and initialise the various parameters required to solve the discretised equations.

3. Initialise the solution matrices for u and v, and set the boundary conditions for them. These are the initial guesses for the solution.

4. Initialise the loops to solve the discretised equations. There will be two nested loops inside another loop.

   - The outermost loop traverses through the columns.
   - The second loop checks for convergence in that column. If the solution for that column has converged, the program control breaks out of this loop.
   - The third loop traverses through the rows.

   The idea is that the leftmost column is solved until convergence, and then the next column, then the column after that and so on.

5. After obtaining the solution, plot the x-velocity (u) and y-velocity (v) fields as contour plots, and also plot the other required parameters such as the normalised velocity (F') as a function of the similarity variable (η) and compare it with the theoretical Blasius solution.

6. Plot the variation of the thickness of the boundary layer and compare it with the solution of the Blasius flat-plate boundary layer equation (equation 6).

Part 3: Crank-Nicolson Scheme

The solution methodology for this scheme is the same as the methodology for the Euler-Implicit Scheme. Figure 5 shows the flow of control for the Crank-Nicolson Scheme.
The only difference between the solution methodology for the Crank-Nicolson Scheme and the Euler-Implicit Scheme is the discretised equation implemented in the code. The Crank-Nicolson Scheme will use the discretised momentum equation equation 16 (but the continuity equation will remain the same).
The code 3 will replace lines 6 to 9 in code 2 which will give the required solution.

```
for i = 2:ny-1
    u(i,j) = (u(i, j-1)^2/dx - v(i,j-1)*(u(i+1,
    j) - u(i-1,j) + u(i+1, j-1) - u(i-1, j-1))
    /(4*dy) + (u(i+1, j) + u(i-1,j) + u(i+1, j
    -1) - 2*u(i, j-1) + u(i-1, j-1))*nu/(2*dy^2)
    )/(u(i, j-1)/dx+ nu/dy^2);
end
```

Code 3. *Crank-Nicolson Scheme Logic Loop*

5. Results and Discussions

Part 1: Euler-Explicit Scheme

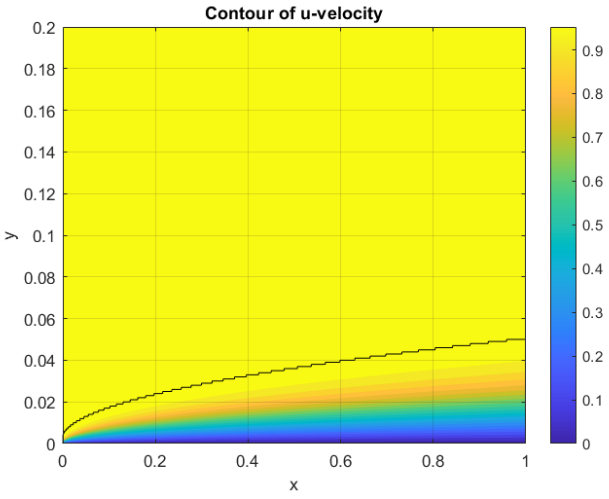

Figure 6. *u-velocity contours*

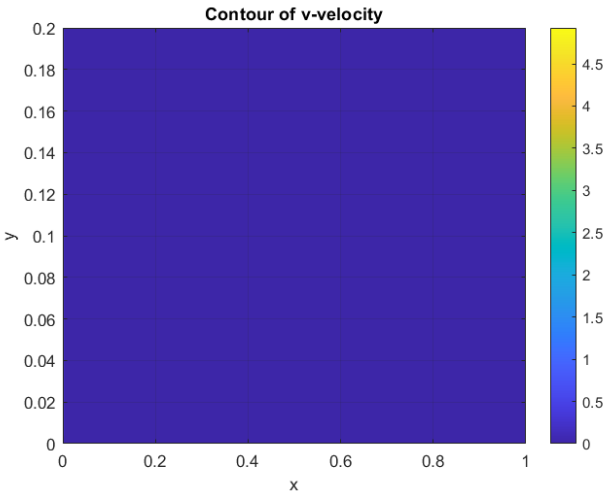

Figure 7. *v-velocity contours*

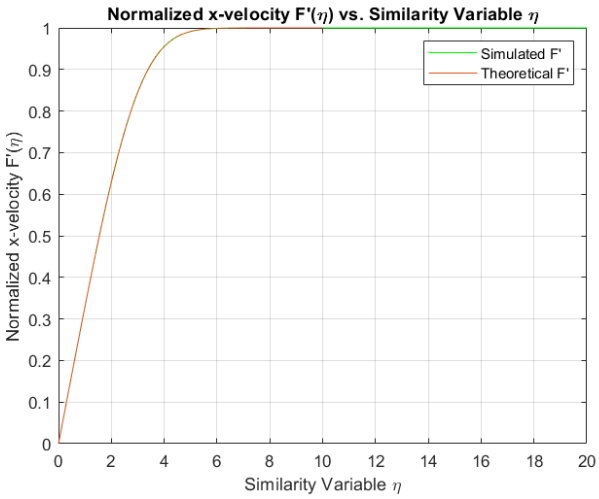

Figure 8. *Normalised u-velocity vs. Similarity Variable*

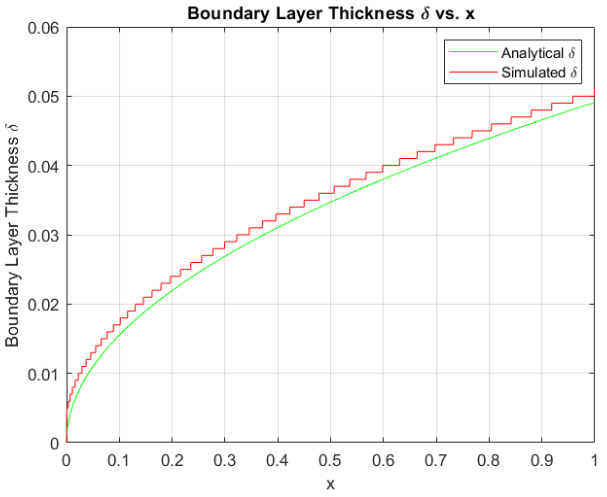

Figure 9. *Variation of the boundary layer thickness with the x-coordinate*

1. *Figure 6* represents the u-velocity contours of the boundary layer formed in the defined computational domain. By reading the colour bar we can see that the velocity at the bottom surface is zero, and it slowly increases until the boundary layer completely separates at about $y = 0.05$ (indicated by the black line).
   The fluid flowing in the region above the black line have the freestream velocity equal to U_∞ and are negligibly affected by the viscous forces.

2. *Figure 7* represents the v-velocity contours of the fluid inside the computational domain. We can see that the v-velocity is almost zero, and this is because we are only considering viscous effects in the x-direction.

3. *Figure 8* represents the normalised u-velocity (F') as a function of the similarity variable (η). We can see from the graph that the theoretical and simulated values of the normalised u-velocity are very similar since the explicit method performs single point calculations for each point and the resulting solution is close to the theoretical values since the method is stable beyond a particular value of the domain's height.

4. *Figure 9* is the plot of the boundary layer thickness (δ) with the x-coordinate for the analytical and the simulated solutions. We can see the space steps in the computational domain from the discontinuous nature of the graph of the simulated solution. It is also slightly above the graph of the analytical solution. This is probably due to the truncation error which occurs during discretisation.

The Euler-Explicit Scheme is a conditionally stable discretisation scheme. It is stable only when the *Courant-Friedrichs-Lewy or CFL condition* is satisfied. For this problem, the condition is $\Delta x \leq \frac{(\Delta y)^2 u_{i,j}}{2\nu}$. When this condition is not satisfied, the explicit scheme will fail to provide an accurate solution. We can observe this in the image below, where the plot drops off at about $x = 0.6$:

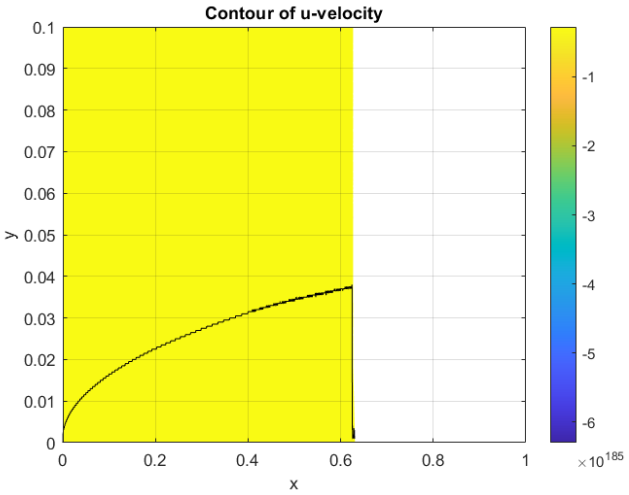

Figure 10. *Instability of the explicit scheme*

Part 2: Euler-Implicit Scheme

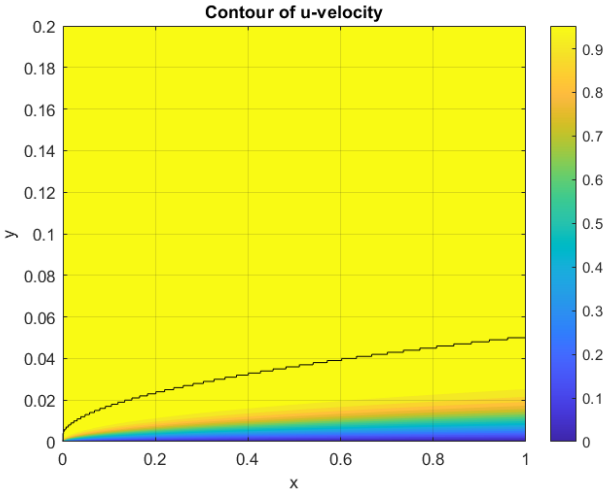

Figure 11. *u-velocity contours*

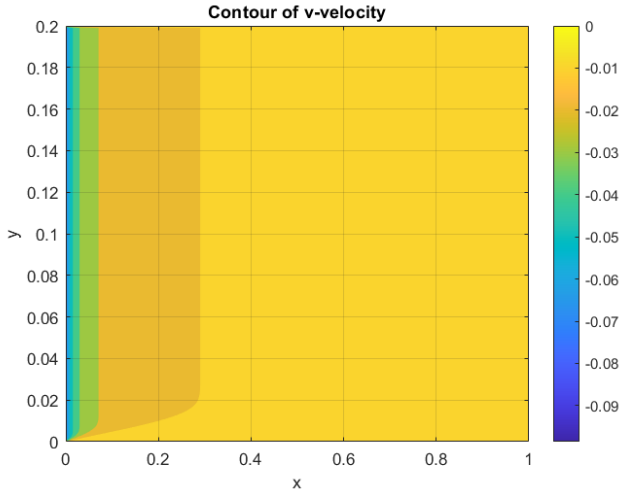

Figure 12. *v-velocity contours*

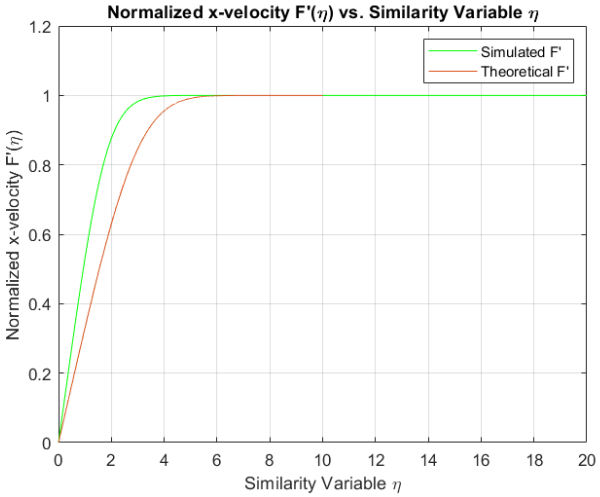

Figure 13. *Normalised u-velocity vs. Similarity Variable*

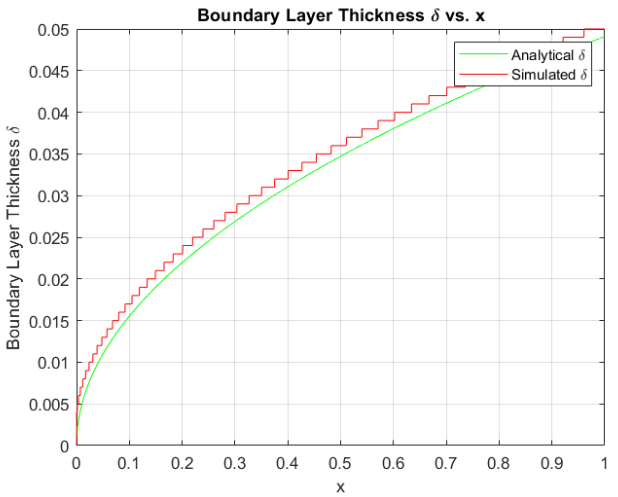

Figure 14. *Variation of the boundary layer thickness with the x-coordinate*

1. *Figure 11* represents the u-velocity contours of the boundary layer formed in the defined computational domain. This is similar to the plot obtained for the

explicit scheme, and we can clearly see the formation and separation of the boundary layer.

2. *Figure 12* represents the v-velocity contours of the boundary layer formed in the defined computational domain. The velocity field is almost zero, and according to the plot it is in the negative y-direction. The cause of this may be the iterative process of the implicit method leading to the generation of this set of values of the v-velocity.

3. *Figure 13* represents the normalised u-velocity (F') as a function of the similarity variable (η). We can see from the graph that the plots of the theoretical and simulated values are similar but they do not overlap. This is probably due to the iterative nature of the implicit scheme.

4. *Figure 14* is the plot of the boundary layer thickness (δ) with the x-coordinate for the analytical and the simulated solutions. This graph is similar to the one obtained for the explicit scheme.

The Euler-Implicit Scheme is an unconditionally stable discretisation scheme.

Part 3: Crank-Nicolson Scheme

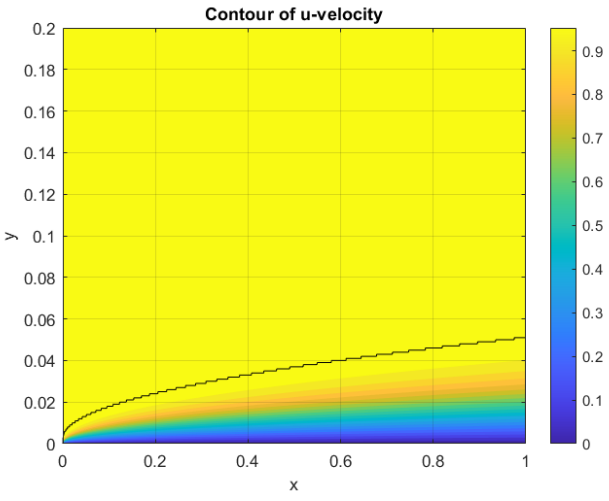

Figure 15. *u-velocity contours*

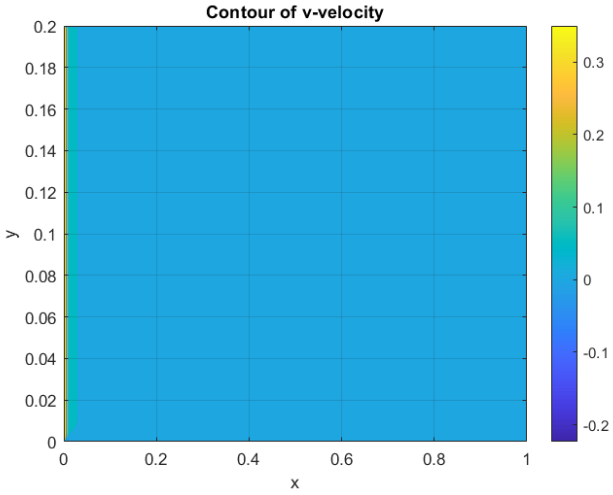

Figure 16. *v-velocity contours*

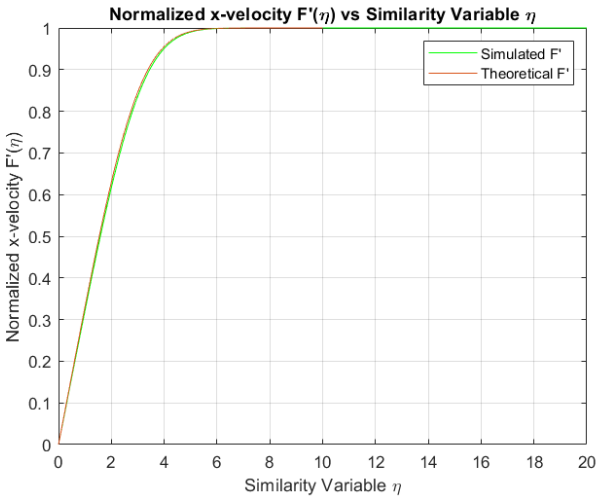

Figure 17. *Normalised u-velocity vs. Similarity Variable*

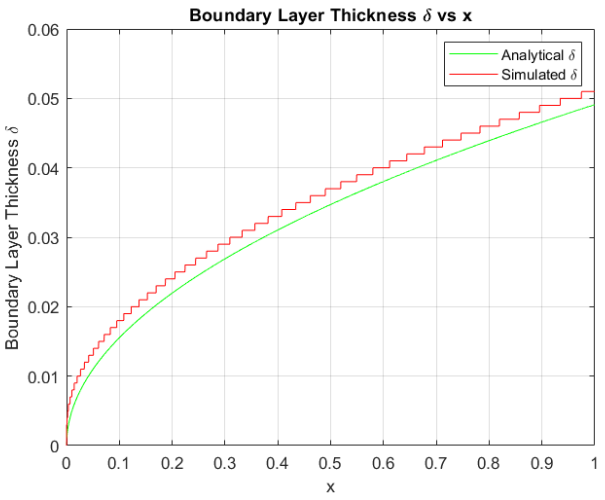

Figure 18. *Variation of the boundary layer thickness with the x-coordinate*

1. *Figure 15* represents the u-velocity contours of the boundary layer formed in the defined computational domain. This is similar to the plot obtained for the

explicit and the implicit scheme. The formation and separation of the the boundary layer can be clearly seen in the image.

2. *Figure 16* represents the v-velocity contours of the boundary layer formed in the defined computational domain. The v-velocity field is zero throughout the domain except for a very thin vertical element at the beginning of the domain. This is probably due to the iterative nature of the Crank-Nicolson Scheme's solution, and the values of the v-velocity may have converged early in that region.

3. *Figure 17* represents the normalised u-velocity (F') as a function of the similarity variable (η). We can see from the graph that the plots of the theoretical and simulated values overlap.

4. *Figure 18* is the plot of the boundary layer thickness (δ) with the x-coordinate for the analytical and the simulated solutions. This graph is similar to the ones obtained for the explicit and the implicit schemes.

The Crank-Nicholson Scheme is an unconditionally stable discretisation scheme.

Comparison between all three discretisation schemes

Discretisation Scheme	CPU Runtime (s)
Euler-Explicit Scheme	0.2379
Euler-Implicit Scheme	0.3101
Crank-Nicolson Scheme	0.3524

Table 2. *CPU runtime of each discretisation scheme*

- **Runtime of the simulation:** From table 2, we can see that the explicit scheme takes the least amount of time to find the solution of the given boundary layer equations, followed by the implicit and the Crank-Nicolson schemes.
- **Stability:** The Euler-Implicit and Crank-Nicolson schemes are unconditionally stable, but the Euler-Explicit scheme is only stable when the CFL condition is satisfied.
- **Accuracy:** The Crank-Nicolson method is the most accurate and provides a solution which is the closest to theoretical values. The explicit scheme is only accurate when it is stable, and the implicit solution provides a solution with slight deviations.
- **Computational Cost:** The Crank-Nicolson and the implicit schemes are the most computationally intensive due to their iterative nature. The explicit scheme is less intensive since it obtains the solution point-by-point.
- **Complexity of Implementation:** The explicit scheme is the least complex to implement. The implicit and Crank-Nicolson schemes are both more complex, but the Crank-Nicolson scheme is harder to implement in code due to the unintuitive nature of its discretisation.

6. Conclusion

This project oversaw the implementation of three numerical discretisation schemes: The Euler-Explicit Scheme, The Euler-Implicit scheme and the Crank-Nicolson Scheme, to solve the boundary layer equations for the viscous flow of a fluid over a flat plate.
The comparison of the three numerical schemes highlighted each scheme's strength, accuracy and computational efficiency. The results of the simulations are consistent with the theoretical predictions. Even the most inaccurate method among the three enabled a clear visualisation of how the boundary layer is formed. Improvements such as optimising these algorithms for larger mesh sizes and Reynolds numbers can be made to increase their robustness and extend their application.

1.

7. Acknowledgements

I would like to thank to Prof. Dilip S. Sundaram for teaching me the principles to enable me to successfully complete this project. His expertise and support greatly helped us.

I would also like to thank the TAs Malay Vyas and Pardha Sai for helping me understand the nuances of the code in this project. Their support was invaluable to me.

I would also like to thank IIT Gandhinagar for providing me the platform and support necessary to complete this project.

Lastly, I would like to thank my peers, whose continued support was instrumental in completing this experiment successfully.

References

[1] H. Schlichting and K. Gersten, *Boundary Layer Theory*, 8th. Berlin, Heidelberg: Springer, 2000, ISBN: 978-3-540-66270-9.

[2] Wikipedia contributors, *Crank–Nicolson method — Wikipedia, The Free Encyclopedia*, [Online; accessed 23-September-2024], 2024. [Online]. Available: https://en.wikipedia.org/wiki/Crank%E2%80%93Nicolson_method.

[3] Wikipedia contributors, *Explicit and implicit methods — Wikipedia, The Free Encyclopedia*, [Online; accessed 23-September-2024], 2024. [Online]. Available: https://en.wikipedia.org/wiki/Explicit_and_implicit_methods.