MAE 423 Heat Transfer Final Report

Princeton University

Department of Mechanical and Aerospace Engineering

Presented by: Ali Ekin Gurgen

December 8, 2020

Table of Contents

1. Executive Summary

The goal of this project was to numerically simulate a low-Reynolds-number two-dimensional convective flow around a heated cylindrical shaped body with and without fins. By doing so, we aimed to observe various interesting phenomena related to viscous flows and unsteady convective heat transfer as well as investigating the effects of adding thin fins to this particular geometry. The cylindrical bodies had a 400K surface temperature and the inflow boundary temperature as well as the upper and lower lid temperatures were 300K. In the first part, we implemented a two-dimensional streamfunction-vorticity finite-difference method on MATLAB to numerically simulate the flow and the resulting heat transfer. The results of this part are presented in Section 3.1. In the second part, we used the same simulation code but simply changed the geometry of the object by adding two thin fins in the upstream and downstream directions. The results of this part are presented in Section 3.2. The governing equations of the implemented numerical simulation can be found in Section 2.

We observed that the flow over the cylinder resulted in trailing vortices which slowly grew and became fully developed when the flow reached a dynamic equilibrium. In the second part, we observed that these vortices started further downstream and at a later timestep. Most importantly, we observed that addition of the fins increased the total heat transfer rate from the cylinder. Therefore, we can conclude that adding thin fins to cylindrical geometries can be beneficial for certain applications by increasing the convective heat loss from the body.

2. Method

We utilized the streamfunction-vorticity numerical method to simulate the flow. We also used the relevant energy equations to determine the thermal effects in the domain. The first step of the implementation was to define the relevant parameters and set up the geometry. One of the important details of this part was the choice of grid spacing. The grid spacing h follows from the conditions that need to be met in order to adequately model the diffusion of vorticity:

$$Re_h = \frac{U_\infty * h}{\nu} < 10 \quad (1)$$

$$\frac{U_\infty * h}{\alpha} < 10 \tag{2}$$

Other governing equations used during this step are:

$$U_{max} = 5 * U_{inf} \tag{3}$$

$$\Delta t = \frac{h}{2 * U_{max}} \tag{4}$$

The next step was to solve for the initial streamfunction (psi) distribution. Since we assume that the vorticity is initially zero, we simply need to solve the LaPlace equation in discrete space variables. The initial boundary inflow and outflow boundary conditions for psi also played an important role at this step. Equation given below shows the implementation of the LaPlace equation in discrete variables.

$$\psi_{i,j}^{(k+1)} = \psi_{i,j}^{(k)} + \frac{F}{4}(\psi_{i+1,j}^{(k)} + \psi_{i-1,j}^{(k+1)} + \psi_{i,j+1}^{(k)} + \psi_{i,j-1}^{(k+1)} - 4\psi_{i,j}^{(k)}) \tag{5}$$

After that, we initially set the vorticity at all points within the field to zero and invoked the no-slip condition to generate vorticity. The following wall condition was used:

$$\omega_w = -2\frac{\psi_{w+1} - \psi_w}{h^2} \tag{6}$$

In order to perform the bulk computation of the flow, the velocities were computed as:

$$u_{i,j}^n = \frac{\psi_{i,j+1}^n - \psi_{i,j-1}^n}{2h} \quad \text{and} \quad v_{i,j}^n = \frac{\psi_{i-1,j}^n - \psi_{i+1,j}^n}{2h} \tag{7}$$

And the Laplacian of vorticity was computed as:

$$\nabla^2 \omega_{i,j}^n = \frac{\sum \omega_{neighbors}^n - 4\omega_{i,j}^n}{h^2} \tag{8}$$

We, then, updated the psi by using the governing equation $\nabla^2 \psi = -\omega$, which resulted in:

$$\psi_{i,j}^{(k+1)} = \psi_{i,j}^{(k)} + \frac{F}{4}(\psi_{i+1,j}^{(k)} + \psi_{i-1,j}^{(k+1)} + \psi_{i,j+1}^{(k)} + \psi_{i,j-1}^{(k+1)} + 4h^2\omega_{i,j}^{n+1} - 4\psi_{i,j}^{(k)}) \tag{9}$$

The next step was to compute the outflow boundary conditions by assuming that u and v are constant. After that, extended our code to include the energy equation which governs the heat flow. Solving the energy equation without the dissipation term and assuming two-dimensional heat transfer with constant properties, we arrived at the following equation:

$$T_{i,j}^{n+1} = T_{i,j}^{n} + \Delta t \left[- u_{i,j}^{n} \frac{(\Delta T)_{i,j}^{n}}{h} - v_{i,j}^{n} \frac{(\Delta T)_{i,j}^{n}}{h} + \alpha \nabla^2 T_{i,j}^{n} \right] \quad (10)$$

We initially considered two programming languages for this implementation: Python and MATLAB. Although we were more familiar with Python, since an initial starter code was provided for it we pursued MATLAB. Choosing MATLAB over Python was probably also a good choice due to its efficiency.

One of the most important components of successfully implementing this numerical simulation was checking the validity of each step one at a time and debugging if necessary before moving on to the next step. Luckily, the lecture notes given in MAE 423 explained the entire algorithm very clearly in a step-by-step manner. Even though the code was written in a single *main.m* file, we followed a modular debugging technique throughout the implementation. We followed good software engineering practices throughout the project and debugging did not turn out to be an issue.

Throughout the simulation, plots of omega, psi, temperature, and centerline temperature distribution along x were saved in a local directory. An additional file called *make-video*, written in Python, was used to produce videos from these frames. These videos were uploaded on Google Drive and the URL link is given in Section 3. We validated the results of the simulations by visually inspecting the resulting videos and confirming that we observe expected behaviors. We also compared our results to various resources on the internet that depict flows over cylindrical objects.

One of the main challenges of running the simulations was the fact that they took a very long time and our implementation was prone to computer malfunctions. In fact, several times we failed to complete simulations due to battery shortage and accidental logging off. Ideally, we could make our implementation more robust by saving our data at several checkpoints throughout the simulation. It could also be nice to utilize a cloud computing service such as AWS or GCP to run the simulation.

3. Results

As mentioned earlier, videos that visually present the results of the simulations were created. These simulation videos for both parts can be found using the URL below: https://drive.google.com/drive/folders/1OuIFFyIqWjf8zsjCbUCPfH7l3rJ4tYDr?usp=sharing

For demonstration purposes, we will also present one frame (each taken at timestep=20000) from each video in the table below. Notice that grey-levels, color-values and contour lines were used to visualize the changes in temperature, vorticity and streamlines.

Table 1: Frames Used for Constructing Videos (at timestep = 20000)

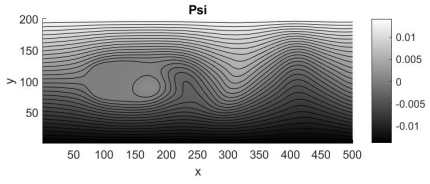
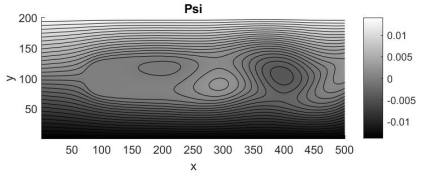
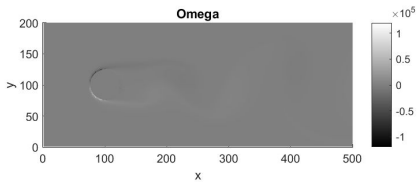
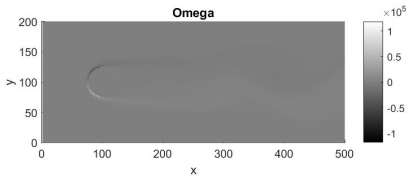
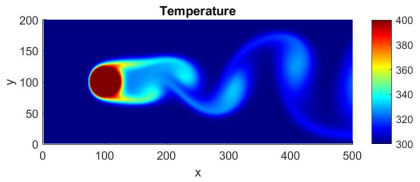
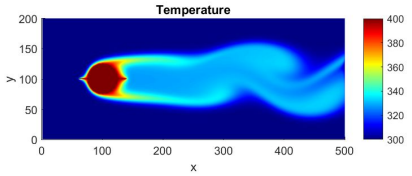
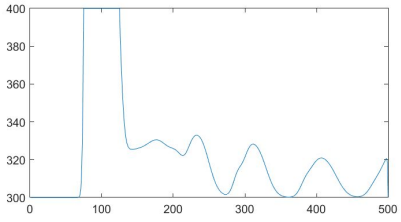
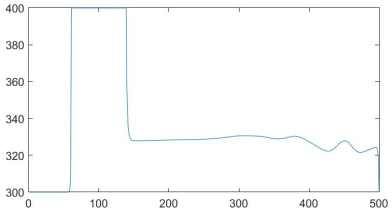
	PART 1	PART 2
psi		
omega		
temperature		
centerline temperature along x		

Figure 1 and 2, given below, show the total convective heat transfer between the fluid and the cylinder.

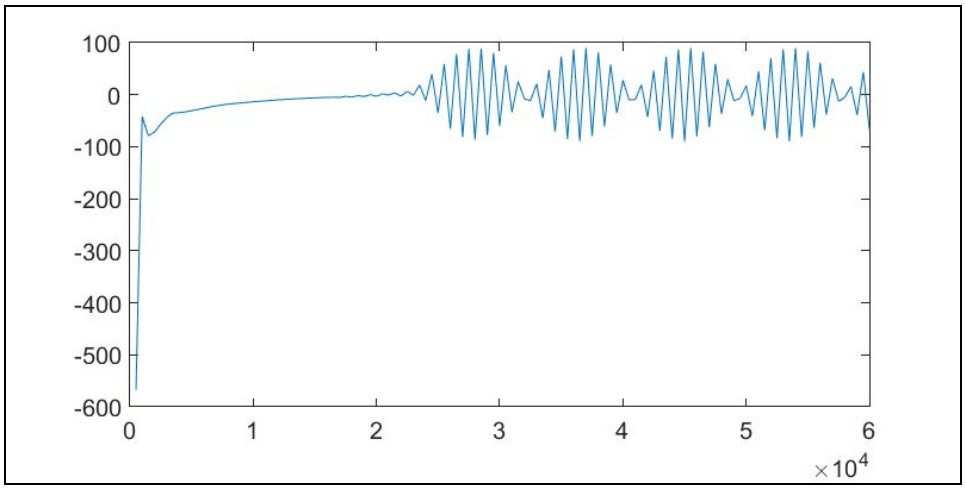


Figure 1: Total heat transfer versus number of timesteps in Part 1

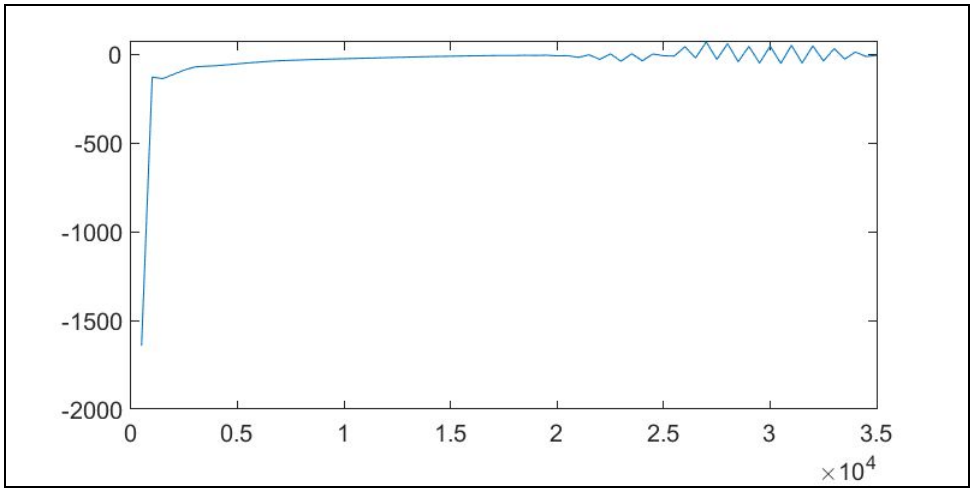


Figure 2: Total heat transfer versus number of timesteps in Part 2

Finally, we plotted the temperature distribution along the x-direction on the horizontal centerline of the cylinder (y = 100 line) at several timesteps throughout the simulation. In addition to presenting these plots in video formats (videos titled *temps* in both folders), for demonstration purposes, we will also give some examples from both simulations in the following pages.

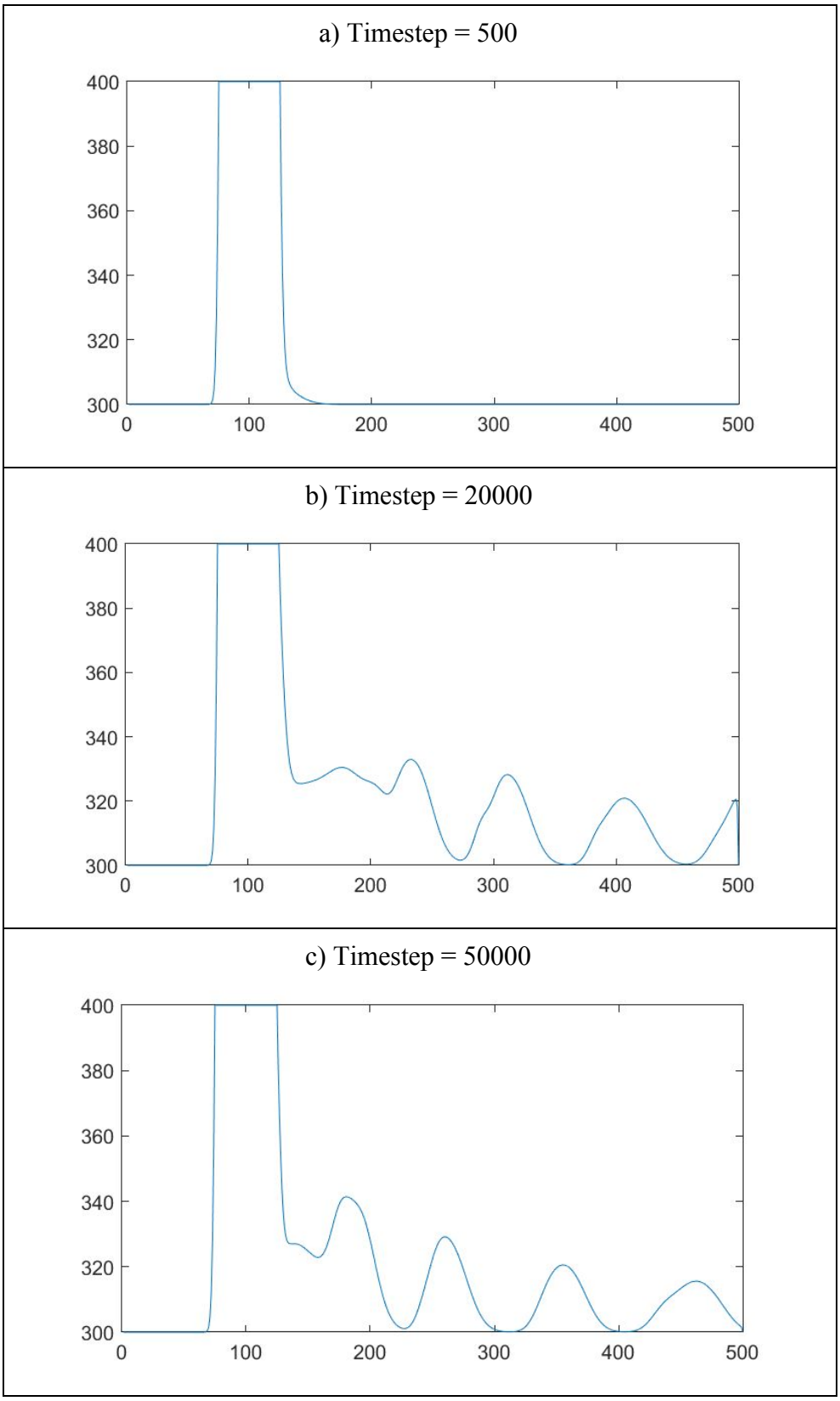


Figure 3: Centerline temperature distr. (in Kelvin) along x at different timesteps (Part 1)

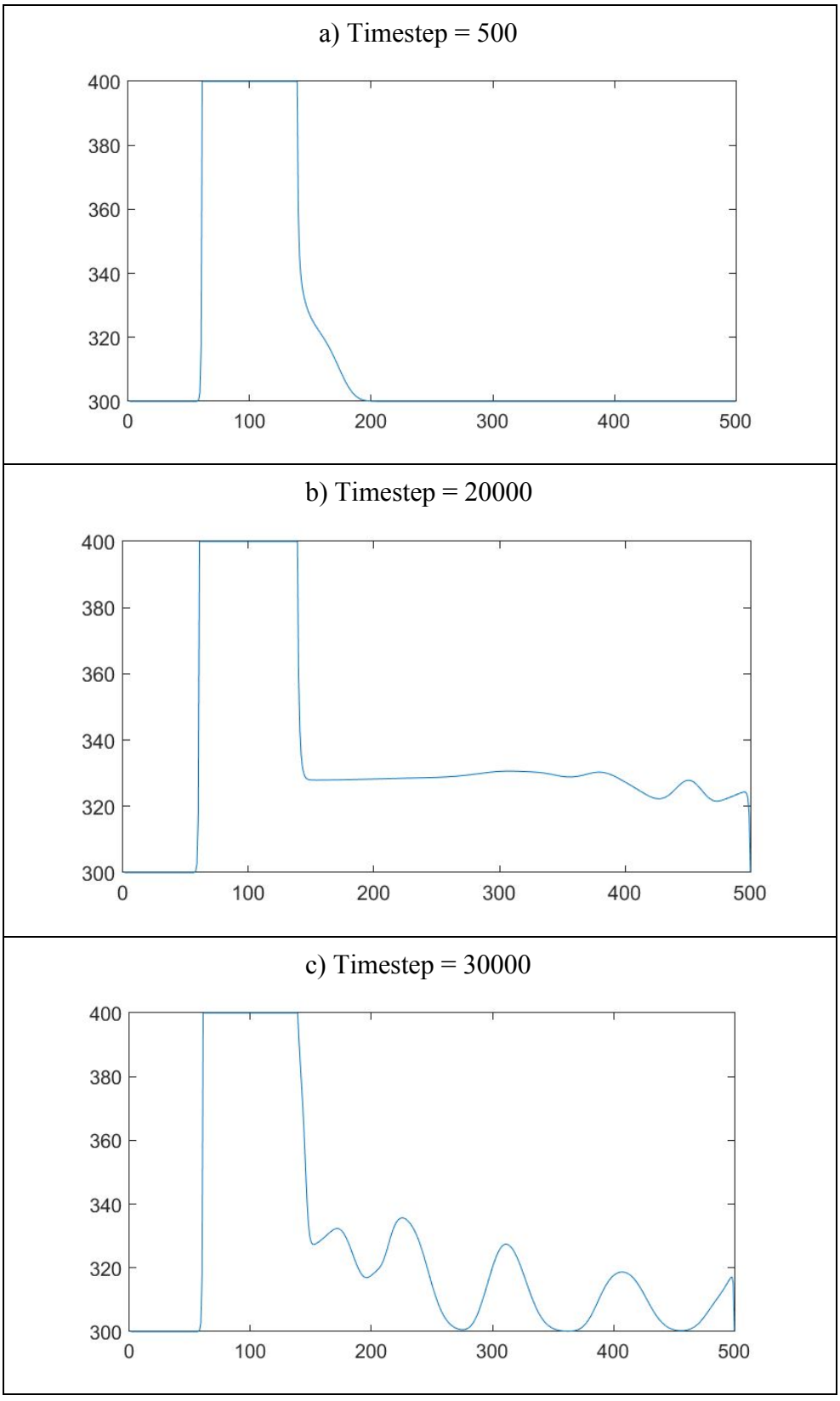


Figure 4: Centerline temperature distr. (in Kelvin) along x at different timesteps (Part 2)

4. Discussion and Conclusions

This project was very beneficial for acquiring a strong understanding of the fundamental concepts of flow mechanics and heat transfer. Most importantly, it was a great opportunity to develop an extensive numerical simulation that works for arbitrary shapes. It would definitely be interesting to learn about some of the real simulation tools that are actively being used in the industry and academia to understand how these numerical methods scale up.

We ran both of our simulations on a laptop with a 6-core processor and 16GB memory running on Windows 10. Part 1's simulation took approximately 15 hours to run 60000 timesteps, while Part 2's simulation took approximately 8 hours to complete 35000 timesteps. As mentioned earlier, given that the results demonstrate the expected behavior, we believe that the simulations were reasonably valid.

For Part 2 of the project, we made a simple but meaningful modification to the object by adding two thin (1-unit thick) fins on the upstream and downstream sides of the cylinder. Both of these fins were 15-units long. Instead of simulating for two completely unrelated geometries, we thought that comparing the results of these two similar simulations could provide us useful insights about the effects of the fins on convective flows and heat transfer. As you can see in Figures 1 and 2, adding the thin fins significantly increased the total heat transfer. We have also observed that the shedding vortices formed much later in the simulation of Part 2 due to the addition of the downstream fin, which can also be seen by comparing Figures 3b and 4b. A side-by-side comparison of the video form of these plots also reveals how the total heat transfer rate in the Part 2 simulation is much greater than that of Part 1. We could also see that the upstream fin did not affect the total heat transfer significantly in its local area. An interesting next step for this project could be investigating the effects of changing certain fin parameters such as the thickness, length, number of fins, or the locations of the fins. Similarly, it would be interesting to run these simulations with different flows as well as completely different arbitrary geometries.

In conclusion, this project was very beneficial and interesting. It helped us understand the governing mechanisms of fluids and heat transfer and also gave us a great opportunity to learn how to develop an extensive numerical simulation that has many practical applications. Our simulations demonstrated the effectiveness of thin fins in increasing the convective heat transfer rate as well as showing us its several other interesting effects on the viscous flow.

5. Appendix

Main Simulation Code (main.m)

```
% MAE 423 Heat Transfer Final Project
% @author: Ali Ekin Gurgen


close all;
global is_second_part;
is_second_part = true;


figure();
set(gcf, 'Position',  [0, 0, 512, 256]);


%**************************************************************************
% Define geometry and constants
%**************************************************************************
height = 200;
width = 500;


mesh = zeros(width, height);
mesh_neighbors = zeros(width, height);
omega = zeros(width, height);
psi = zeros(width, height);
temp = zeros(width, height);


v = zeros(width, height);
u = zeros(width, height);
```

```
cyl_x = 100;

cyl_y = 100;

cyl_d = 50;


num_time_steps = 35000;

frames_per_img = 100;

tolerance = 0.01;


T_surf = 400;

T_flow = 300;


% Properties of air at 300K (from engineeringtoolbox.com)

alpha = 22.18 * 10^(-6);

visc = 1.568 * 10^(-5);

k = 2.624 * 10^(-2);

F = 1.5;

U_inf = 5;

const = U_inf * (height / 2);


h = min((9*visc/U_inf), (9*alpha/U_inf));

dt = (h / (5 * U_inf)) / 2;

u(1,:) = U_inf;


steps = [];

total_heat_transfer = [];
```

```
%*************************************************************************
% Cylinder Setup
%*************************************************************************
for i = 1:width
        for j = 1:height
        dist = sqrt((i - cyl_x)^2 + (j - cyl_y)^2);
        if dist <= cyl_d/2 || (is_second_part && ((j == 100) && (i > 60) && (i < 140)))
                mesh(i, j) = 1;
                temp(i, j) = T_surf;
        else
                temp(i, j) = T_flow;
                psi(i, j) = (U_inf * j - const) * h;
        end
        end
end


for i = 2:(width - 1)
        for j = 2:(height - 1)
        if (mesh(i, j) == 0)
                if (mesh(i - 1, j) || mesh(i + 1, j) || mesh(i, j - 1) || mesh(i, j + 1))
                  mesh_neighbors(i, j) = 1;
                end
        end
        end
end
```

```
%**************************************************************************

% Solve for psi at t=0

%**************************************************************************

running = true;

while running

        psi_prev = psi;

        for i = 2:(width - 1)

        for j = 2:(height - 1)

                if (mesh(i,j) == 0)

                psi(i, j) = psi(i, j) + (F / 4) * (psi(i - 1, j) + psi(i + 1, j) + psi(i, j -
1) + psi(i, j + 1) - 4 * psi(i, j));

                end

        end

        end

        differences = abs(psi - psi_prev) ./ psi_prev;

        if (max(differences) <= tolerance)

        running = false;

        end

end



%**************************************************************************

% Run Iterative Method

%**************************************************************************


for time_step = 1:num_time_steps

        omega_prev = omega;

        temp_prev = temp;



        for i = 2:(width - 1)
```

```
        for j = 2:(height - 1)
                if mesh(i, j)
                u(i, j) = (psi(i, j + 1) - psi(i, j - 1)) / (2 * h);
                v(i, j) = (psi(i - 1, j) - psi(i + 1, j)) / (2 * h);
                omega(i, j) = (-2 / (h * h)) * (psi(i - 1, j) + psi(i + 1, j) + psi(i, j - 1)
+ psi(i, j + 1));
                end
        end
        end
        u(:,1)=U_inf;
        u(:,height)=U_inf;

        for i = 2:(width - 1)
        for j = 2:(height - 1)
                if (mesh(i, j) == 0)
                v_omega = 0;
                u_omega = 0;

                if (v(i, j) < 0)
                        v_omega = v(i, j + 1) * omega_prev(i, j + 1) - v(i, j) * omega_prev(i,
j);
                else
                        v_omega = v(i, j) * omega_prev(i, j) - v(i, j - 1) * omega_prev(i, j -
1);
                end

                 if (u(i, j) < 0)
                        u_omega = u(i + 1, j) * omega_prev(i + 1, j) - u(i, j) * omega_prev(i,
j);
                else
```

```
                        u_omega = u(i, j) * omega_prev(i, j) - u(i - 1, j) * omega_prev(i - 1,
j);

                end



                vortL = (omega_prev(i - 1, j) + omega_prev(i + 1, j) + omega_prev(i, j - 1) +
omega_prev(i, j + 1) - 4 * omega_prev(i, j)) / (h * h);

                omega(i, j) = omega_prev(i, j) + dt * (-u_omega / h - v_omega / h + visc *
vortL);

                end

        end

        end

        omega(width, :) = omega(width - 1, :);

        psi(width, :) = 2 * psi(width - 1, :) - psi(width - 2, :);



        running = true;

        while running

        psi_prev = psi;

        for i = 2:(width - 1)

                for j = 2:(height - 1)

                if (mesh(i,j) == 0)

                        psi(i, j) = psi_prev(i, j) + (psi(i - 1, j) + psi(i + 1, j) + psi(i, j
- 1) + psi(i, j + 1) + 4 * h * h * omega(i, j) - 4 * psi(i, j)) * (F / 4);

                end

                end

        end

        differences = abs(psi - psi_prev) ./ psi_prev;

        if (max(differences) <= tolerance)

                running = false;

        end

        end
```

```
        for i = 2:(width -1)
        for j = 2:(height - 1)
                if (mesh(i, j) == 0)
                v_deltaT = 0;
                u_deltaT = 0;

                if v(i, j) < 0
                        v_deltaT = v(i, j) * (temp_prev(i, j + 1) - temp_prev(i, j));
                else
                        v_deltaT = v(i, j) * (temp_prev(i, j) - temp_prev(i, j - 1));
                end

                if u(i, j) < 0
                        u_deltaT = u(i, j) * (temp_prev(i + 1, j) - temp_prev(i, j));
                else
                        u_deltaT = u(i, j) * (temp_prev(i, j) - temp_prev(i - 1, j));
                end

                tempL = (temp_prev(i - 1, j) + temp_prev(i + 1, j) + temp_prev(i, j - 1) +
temp_prev(i, j + 1) - 4 * temp_prev(i, j)) / (h * h);
                temp(i, j) = temp_prev(i, j) + dt * (-u_deltaT / h - v_deltaT / h + alpha *
tempL);
                end
        end
        end

        if mod(time_step, frames_per_img) == 0
        saveFigures(psi, omega, temp, time_step);
        disp("Current Step: " + time_step)
        end
```

```
        if mod(time_step, 5*frames_per_img) == 0
        if is_second_part
                save_dir = './images-2/';
            else
                save_dir = './images/';
        end


        steps(end+1) = time_step;
        sum_heat_transfer = 0;
        for i = 1:width
                for j = 1:height
                if mesh_neighbors(i, j)
                      sum_heat_transfer = sum_heat_transfer + (temp(i, j) - temp_prev(i, j)) /
h;
                end
                end
        end
         total_heat_transfer(end+1) = -k * sum_heat_transfer;
        plot(steps, total_heat_transfer);
        saveas(gcf, 'total_transfer', 'png');
        clf;


        temp_dist_along_x = temp(:,100);
        plot((1:500), temp_dist_along_x);
        saveas(gcf, sprintf(strcat(save_dir, 'temps/temps-%d.png'), time_step));
        clf;
        end
end
```

```
%**************************************************************************
% Function to save figures
%**************************************************************************
function saveFigures(psi, omega, temp, time_step)
        global is_second_part;
        if is_second_part
        save_dir = './images-2/';
        else
        save_dir = './images/';
        end


        s = pcolor(flipud(rot90(psi)));
        colormap(gcf, gray);
        colorbar;
    contour(flipud(rot90(psi)), 30, 'black');
        title("Psi");
        xlabel('x')
        ylabel('y')
        saveas(gcf, sprintf(strcat(save_dir, 'psi/psi-%d.png'), time_step));
        clf;


        s = pcolor(flipud(rot90(omega)));
        colormap(gcf, gray);
        colorbar;
        title("Omega");
        xlabel('x')
        ylabel('y')
        saveas(gcf, sprintf(strcat(save_dir, 'omega/omega-%d.png'), time_step));
```

```
        clf;


        s = pcolor(flipud(rot90(temp)));

        colormap(gcf, jet);

        colorbar;

        title("Temperature");

        xlabel('x')

        ylabel('y')

        saveas(gcf, sprintf(strcat(save_dir, 'temp/temp-%d.png'), time_step));

        clf;

end
```

Python Code for Creating Videos from Plots (make-video.py)

```
import sys
import os
import cv2

def make_video(input_dir, output, arg):

    images = [img for img in os.listdir(input_dir) if img.endswith(".png")]
    images.sort(key=lambda x: int(x[len(arg)+1:len(x)-4]))
    frame = cv2.imread(os.path.join(input_dir, images[0]))
    height, width, layers = frame.shape

    video = cv2.VideoWriter(output, 0, 10, (width,height))

    for image in images:
        video.write(cv2.imread(os.path.join(input_dir, image)))

    cv2.destroyAllWindows()
    video.release()

    print("Saved video for " + arg)

if __name__ == "__main__":

    is_second_part = True

    if not is_second_part:
        input_dir = "C:\\Users\\aliek\\Desktop\\MAE-423\\final-project\\images\\" +
```

```
sys.argv[1]
        output = "C:\\Users\\aliek\\Desktop\\MAE-423\\final-project\\videos\\" + sys.argv[1]
+ ".mp4"
    else:
        input_dir =
"C:\\Users\\aliek\\Desktop\\MAE-423\\final-project\\images-2\\"+sys.argv[1]
        output = "C:\\Users\\aliek\\Desktop\\MAE-423\\final-project\\videos-2\\" +
sys.argv[1] + ".mp4"

    make_video(input_dir, output, sys.argv[1])
```