PRINCETON UNIVERSITY

MAE 423: Heat Transfer

Fall 2019-2020

Final Project Report

January 14, 2020

Submitted By:
Morgan Baker
Jens Clausen
Sam Dale

Submitted To:
Prof. Daniel Nosenchuck
Benjamin Schaffer

Executive Summary

The goals for this project were to first simulate a viscous, low Reynolds number flow around a cylinder (Part 1) and then simulate a similar flow around an arbitrary geometry (Part 2). In Part 2, the group was interested in determining if the heat transfer increased due to the addition of a line of solid material attached to the downstream side of the cylinder at the center line. The first part of this project was used to make sure that there was a working simulation, which could test a more interesting geometry in Part 2. As a result, most of the coding work was spent on Part 1.

For this project, the hypothesis was that the heat transfer would increase due to the reduction in separation of the flow on the downstream side of the cylinder. The main assumptions and restrictions were that the flow is of a low Reynolds number and there is only two-dimensional heat transfer. The method used was a two-dimensional stream-function/vorticity finite difference model of governing equations with a Cartesian basis.

The flow over a heated cylinder (Part 1) produced a flow with trailing vortices with an associated Strouhal number of 0.241. The investigation of adding a line of solid material to the downstream side of the cylinder (Part 2) resulted in a Strouhal number of 0.238. This decrease in the Strouhal number between the two parts indicates that the line has slightly reduced the number of vortices created. In the simulations that were produced, it can be seen when comparing the flows of Part 1 and Part 2 that the vortices are smaller and begin further downstream in Part 2. The bigger change is that of the average heat transferred from Part 1 where the average power was 6.6W and increased to 9W in Part 2 with the addition of the fin.

Contents

1 Method

1.1 Setup and Boundary Conditions

$$Re_h = \frac{U_\infty h}{\nu}, Re_h < 10 \tag{1}$$

$$h = min\left(\frac{(10-1)\nu}{U_{inf}}, \frac{(10-1)\alpha}{U_{inf}}\right) \tag{2}$$

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

$$\Delta t \leq \frac{h}{U_{max}} \tag{4}$$

$$\Delta t = \frac{h}{2U_{max}} \tag{5}$$

$$\psi_{freelid} = \frac{U_{inf} height}{2} \tag{6}$$

$$\psi_{i,j} = jU_{inf} - \psi_{freelid} \tag{7}$$

$$\psi_{solid} = \psi_{freelid} = 0 \tag{8}$$

$$T_{solid} = 400K \tag{9}$$

$$T_{fluid} = 300K \tag{10}$$

1.2 Initial Psi Setup

$$\psi_{i,j}^{k+1} = \psi_{i,j}^{k} + \frac{F}{4}\left(\psi_{i+1,j}^{k} + \psi_{i-1,j}^{k+1} + \psi_{i,j+1}^{k} + \psi_{i,j-1}^{k+1} - 4\psi_{i,j}^{k}\right) \tag{11}$$

1.3 Wall Conditions

$$\omega_w = -2\frac{\psi_{w+1} - \psi_w}{h^2} = \frac{-2(\psi_{i-1,j} + \psi_{i+1,j} + \psi_{i,j-1} + \psi_{i,j+1})}{h^2} \tag{12}$$

1.4 Bulk Fluid

$$u_{i,j}^n = \frac{\psi_{i,j+1}^n - \psi_{i,j-1}^n}{2h} \tag{13}$$

$$v_{i,j}^n = \frac{\psi_{i-1,j}^n - \psi_{i+1,j}^n}{2h} \tag{14}$$

$$\nabla^2 \omega_{i,j}^n = \frac{\sum \omega_{neighbors}^n - 4\omega_{i,j}^n}{h^2} = \frac{\omega_{i-1,j}^n + \omega_{i+1,j}^n + \omega_{i,j-1}^n + \omega_{i,j+1}^n - 4\omega_{i,j}^n}{h^2} \tag{15}$$

$$\begin{aligned}\Delta(u\omega)^n &= \begin{cases} (u\omega)_{i+1,j}^n - (u\omega)_{i,j}^n & u_{i,j}^n < 0 \\ 0 & u_{i,j}^n = 0 \\ (u\omega)_{i,j}^n - (u\omega)_{i-1,j}^n & u_{i,j}^n > 0 \end{cases} \\ &= \begin{cases} u_{i+1,j}^n \omega_{i+1,j}^n - u_{i,j}^n \omega_{i,j}^n & u_{i,j}^n < 0 \\ 0 & u_{i,j}^n = 0 \\ u_{i,j}^n \omega_{i,j}^n - u_{i-1,j}^n \omega_{i-1,j}^n & u_{i,j}^n > 0 \end{cases}\end{aligned} \tag{16}$$

$$\begin{aligned}\Delta(v\omega)^n &= \begin{cases} (v\omega)_{i,j}^n - (v\omega)_{i,j}^n & v_{i,j}^n < 0 \\ 0 & v_{i,j}^n = 0 \\ (v\omega)_{i,j}^n - (v\omega)_{i,j}^n & v_{i,j}^n > 0 \end{cases} \\ &= \begin{cases} v_{i,j}^n \omega_{i,j}^n - v_{i,j}^n \omega_{i,j}^n & v_{i,j}^n < 0 \\ 0 & v_{i,j}^n = 0 \\ v_{i,j}^n \omega_{i,j}^n - v_{i-1,j}^n \omega_{i-1,j}^n & v_{i,j}^n > 0 \end{cases}\end{aligned} \tag{17}$$

$$\omega_{i,j}^{n+1} = \omega_{i,j}^n + \Delta t \left(-\frac{\Delta(u\omega)^n}{h} - \frac{\Delta(v\omega)^n}{h} + \nu \nabla^2 \omega_{i,j}^n \right) \tag{18}$$

$$\psi_{i,j}^{k+1} = \psi_{i,j}^k + \frac{F}{4} \left(\psi_{i+1,j}^k + \psi_{i-1,j}^{k+1} + \psi_{i,j+1}^k + \psi_{i,j-1}^{k+1} + 4h^2 \omega_{i,j}^{n+1} - 4\psi_{i,j}^k \right) \tag{19}$$

1.5 Outflow Boundary

$$\omega_{i,j} = \omega_{i-1,j} \tag{20}$$

Upper/Lower Boundaries:

$$\psi = constant = 0 \tag{21}$$

$$\omega = 0 \tag{22}$$

1.6 Temperature Update

$$\nabla^2 T^n_{i,j} = \frac{\sum T^n_{neighbors} - 4T^n_{i,j}}{h^2} = \frac{T^n_{i-1,j} + T^n_{i+1,j} + T^n_{i,j-1} + T^n_{i,j+1} - 4T^n_{i,j}}{h^2} \tag{23}$$

$$u^n_{i,j}(\Delta T)^n_{i,j} = \begin{cases} u^n_{i,j}\left(T^n_{i+1,j} - T^n_{i,j}\right) & u^n_{i,j} < 0 \\ 0 & u^n_{i,j} = 0 \\ u^n_{i,j}\left(T^n_{i,j} - T^n_{i-1,j}\right) & u^n_{i,j} > 0 \end{cases} \tag{24}$$

$$v^n_{i,j}(\Delta T)^n_{i,j} = \begin{cases} v^n_{i,j}\left(T^n_{i,j+1} - T^n_{i,j}\right) & v^n_{i,j} < 0 \\ 0 & v^n_{i,j} = 0 \\ v^n_{i,j}\left(T^n_{i,j} - T^n_{i,j-1}\right) & v^n_{i,j} > 0 \end{cases} \tag{25}$$

$$T^{n+1}_{i,j} = T^n_{i,j} + \Delta t\left(-\frac{u^n_{i,j}(\Delta T)^n_{i,j}}{h} - \frac{v^n_{i,j}(\Delta T)^n_{i,j}}{h} + \alpha\nabla^2 T^n_{i,j}\right) \tag{26}$$

1.7 Calculate Total Heat Transfer from Cylinder

$$T(t) = -k\frac{\partial T}{\partial n} \tag{27}$$

1.8 Method Discussion

In this project, there were a number of issues. The group started as two separate groups (Sam & Morgan, Jens). The code was written separately at first (Sam & Morgan in Python, Jens in Java and then in Python) On January 10^{th}, both groups were having similar issues related to Python's slow run-times on brute force algorithms. The groups attempted to speed up the programs by implementing techniques such as using slices and Cython. These were either ineffective or not friendly to beginners. Sam and Morgan's code was having issues with the x-velocity and y-velocity matrices as the upstream values would diverge. Jens' code was having problems with the vorticity values not spreading into the flow as desired which was made worse by the slow run-times making the code difficult to debug.

It was at this point that the two groups decided to join into one and tackle the problems together. To tackle the slow run-times, the group decided to completely re-write all the code in MATLAB. This programming language was found to be more intuitive and relatively speedy due to the fact that a number of hours had already been spent working on the project, so it was a case of putting the group's heads together to make sure no careless errors were made while writing the code again. After writing the code in MATLAB, there were some issues since the values for vorticity were getting very large after just a few iterations of the time step. After some debugging, the problems in the code were found and fixed.

Code debugging was a major issue throughout this project. Hours were spent attempting to find solutions to the individual problems, and both Professor Nosenchuck and Ben were contacted multiple times for assistance. However, the separate efforts of the initial two groups were not enough to produce properly functioning code. Therefore, the decision to join together was invaluable to the success of this assignment. Rewriting the code as a collective effort in MATLAB allowed for problems to be found quickly, and time was used much more efficiently.

The results were validated by first comparing the videos that were produced to ones found for similar flows online and also to the one that was shown by Professor Nosenchuck in lecture. Finding lots of similarities would give us a level of confidence that our simulation is valid. We also checked the energy conservation of the flow once it had reached steady state. This means checking the energy flowing out from the cylinder is equal to the energy flowing through the outflow boundary.

Elements that were significant to this project that the group did not produce was the filename sorting algorithm used by the video generation Python file. This algorithm was found on Stack Overflow. A number of MATLAB and Python functions were also used with regards to plotting and formatting that the group did not write.

2 Results

The results of the simulations were made into videos. Plots were made showing the temperature field, how the stream function changes over time, and the vorticity values. These give insight into why the temperature field changes in the way that it does. The simulation videos can be found below:

Video of Part 1 can be found here: `https://youtu.be/Va4I72YeALE`

Video of Part 2 can be found here: `https://youtu.be/b8tadgjrUlY`

(We hope you enjoy the music.)

On the next few pages, there are screenshots from the two simulations that were run. The screenshots are from the flow at steady state. The plot at the top of each screenshot is the stream function. The middle plot is the vorticity, and the bottom plot is the temperature in Kelvin. Along with these, there are plots of Temperature and x Velocity of the flow along the centerline from when the flow has reached steady state.

2.1 Part 1

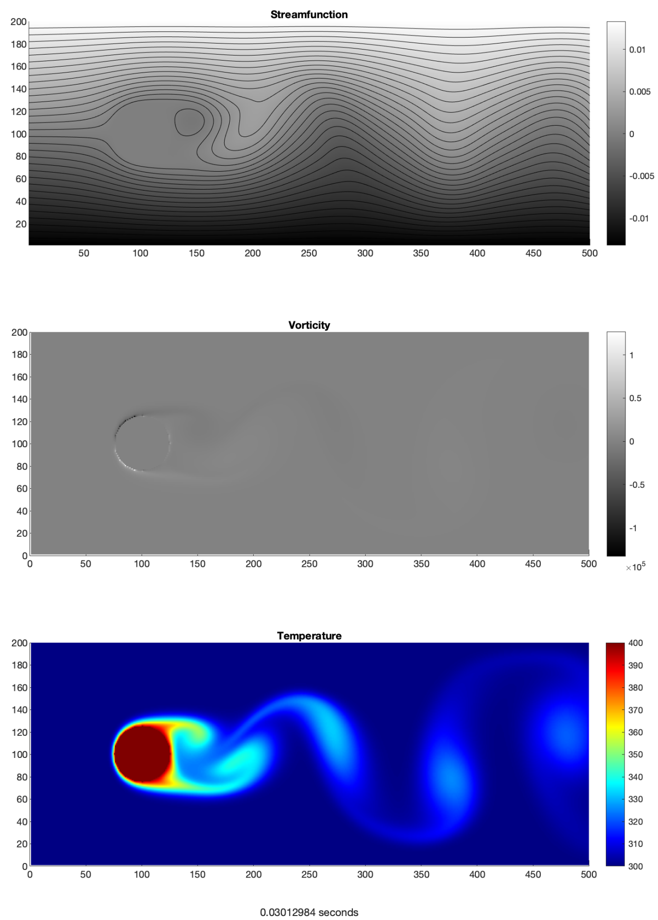

Figure 1: Part 1 Simulation after 56,550 Time Steps

In the simulation for Part 1, the vorticies only start to form after around 10,000 time steps. Initially, the flow is symmetrical about the line y = 100. The vorticies start to form at the t = 0.0084s mark. After the vorticies form, they grow until the flow reaches dynamic equilibrium. From this point on, the flow does not change its behaviour. There is a small area of hot fluid around the cylinder.

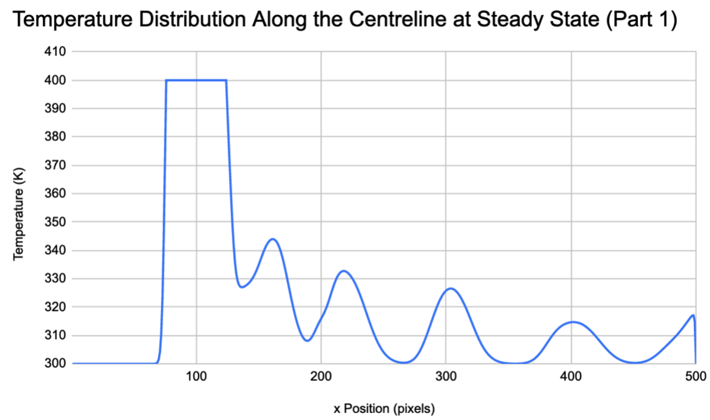

Figure 2: Temperature along Centerline at Steady State (Part 1)

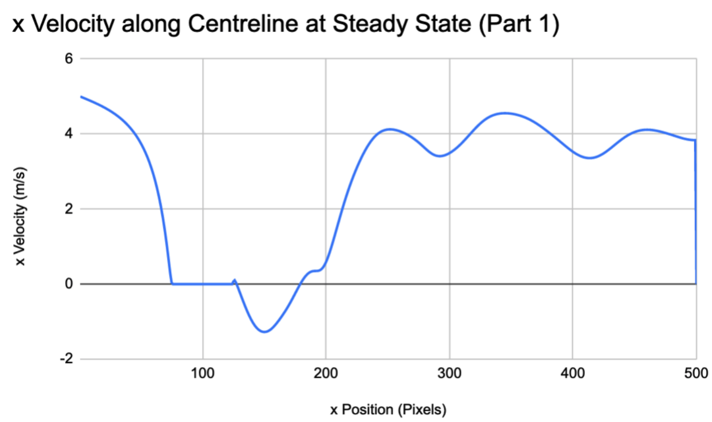

Figure 3: x Velocity along Centerline at Steady State (Part 1)

2.2 Part 2

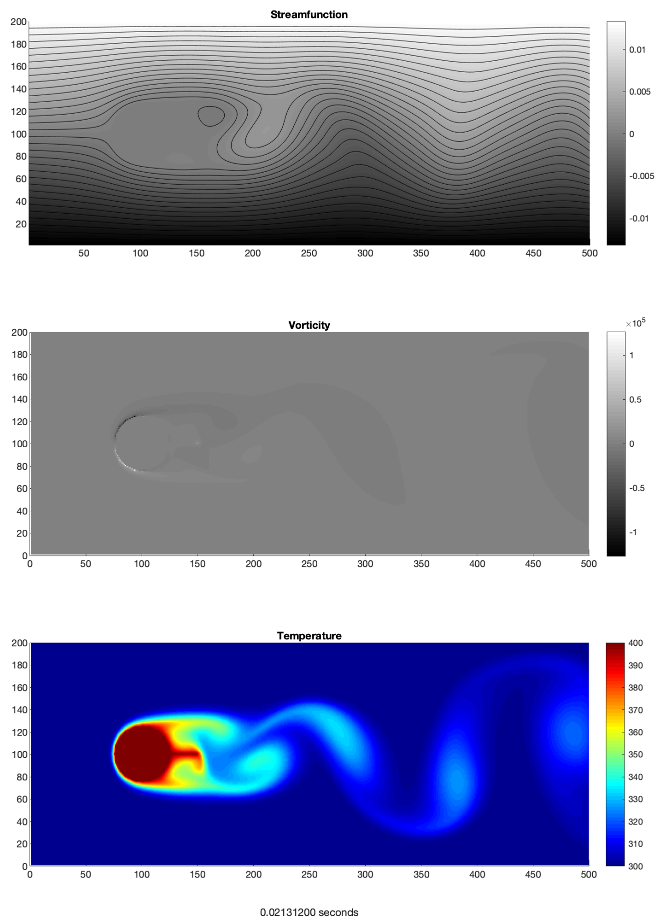

Figure 4: Part 2 Simulation after 40,000 Time Steps

In the part 2 simulation, the behavior is very similar to that of Part 1. Initially, the flow is symmetrical about the line y = 100. Vorticies start to form from the t = 0.0106s point. This is 0.0022s later than they form in Part 1. From here, they grow until they are fully developed where the flow reaches its dynamic equilibrium and remains constant in time from that point on. The main difference between this flow and Part 1 is that the vorticies are slightly smaller and they begin further downstream than they do in Part 1. From the vorticity plots, it can be seen that the vorticity in Part 2's flow fields is much more concentrated in the center of the shedding flow. This is seen in the contrast of the clockwise and counter-clockwise vorticities' colors (In Part 2: more white and black, in Part 1: more gray). The fin on the back of the cylinder in Part 2 has caused there to be a much more noticeably large area of hot fluid on the downstream side of the cylinder.

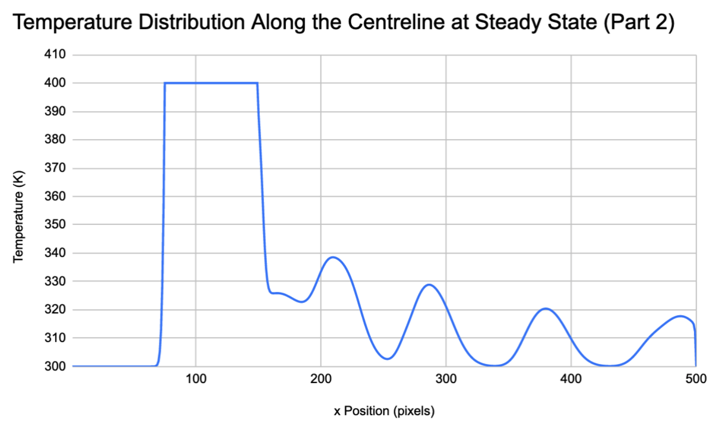

Figure 5: Temperature along Centerline at Steady State (Part 2)

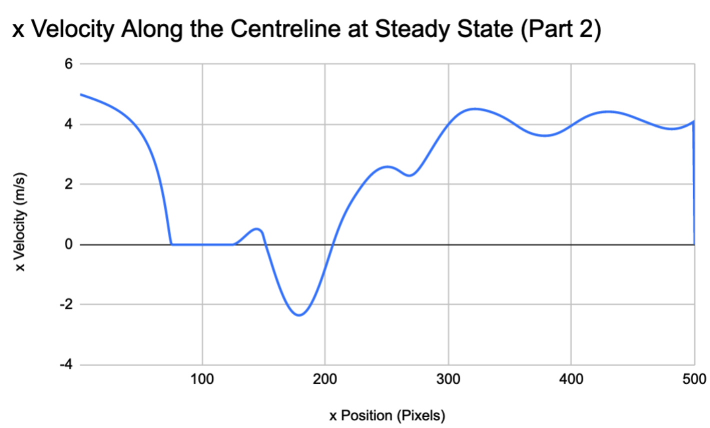

Figure 6: x Velocity along Centerline at Steady State (Part 2)

3 Discussion and Conclusions

A lot was learned from this project. Unfortunately for the group, a lot of this learning was about Python's inefficiencies (Ha Ha..). Looking past this, the group learned a great deal about the basics of simulating flows as no one had ever written a simulation of a fluid. This project gave invaluable insight into one way of going about doing this. While there are undoubtedly many more complex pieces of software that can handle simulations that are of a much higher level, it is useful to understand how the governing equations can be manipulated into equations that can be used without a PDE solver.

The simulation of Part 1 took 11 hours to run 60,000 time steps. Part 2 took a little bit more than 7 hours to run 40,000 time steps. This was about 30 times better than if Python was used without the non-trivial Python optimization methods. This is a big reason as to why we would strongly recommend that future students do this project in MATLAB as opposed to Python. The run time was able to be cut down by a factor of 3 by only saving a picture of the simulation every 50 time steps and saving all the data for every time-step that is a multiple of 1,000. A feature was added to the code that allowed for the simulation to start again from any multiple of 1,000 time steps thanks to this method of saving the data. This was useful for moments where our computers let us down in areas such as battery life and logging off due to being idle for too long.

The simulation for this project was reasonably valid. In each part, once the flow had reached equilibrium, the heat flux through a surface around the cylinder was tested and compared to the heat flux through a vertical surface near the outflow boundary. Over time, the difference between these values was negligible. This gave the group confidence that the simulation was valid because energy was conserved from the point where it entered the flow to when it exited the flow at the rightmost boundary. This simulation has it's limitations in terms of components that would be introduced in an equivalent 3 dimensional simulation.

One aspect that the group was surprised about was how fast the flow became unsteady and started to have shedding vorticies. The Part 1 simulation only simulates 0.03 seconds of the flow. This is surprising because of how little time it takes for the vorticies to appear and reach a dynamic equilibrium in a flow with a low Reynolds number. The group was also surprised by how relatively cool the flow remained directly behind the cylinder in Part 1. Even at the end of the simulation, the temperature of the fluid at a distance of one third of the radius is still at 300K. This is surprising because of how low the Reynolds number is. From previous work in fluids, the group knew that the flow had not become turbulent and expected it to behave in a way that resembled inviscid flow, since the velocity gradients are low. However, it was seen that this is not the case. Even at low Reynolds numbers the flow will start to create vorticies.

For the second part of the project, the group chose to run a simulation of a cylinder with a straight, one pixel wide fin on the center line, placed on the downstream side. This line of extra solid material is 25 pixels long (equal to the radius of the circle). This simulation was chosen in order to see the effects it would have on the total heat transfer as well as the St number. This is applicable to lots of situations, which is part of the reason the group found it interesting. If it increases the total heat transfer, it would be a quick way of improving a simple heat exchanger in order to make it more effective. The group found that the average wattage increased from 6.6W in Part 1 to 9.0W in Part 2. This was calculated using the temperature values from the simulation and equation (27). This is a very significant change that has been caused by the fin and is useful in a number of practical applications. Another surprising feature of the flow in Part 2 is that it started to form shedding vorticies at t = 0.0106s, this is 0.0022s later than it took in Part 1. This could explain part of why the average wattage was so different. It seems like the fin that was added has prevented the flow from forming vorticies easily. The group believes this is due to the fact that the solid barrier stops the flow from crossing the center line until further downstream. This is relevant for situations where shedding is an issue. For example, very small heat exchangers, where even small forces can affect the structural integrity of the exchanger. Vortex shedding creates shear forces perpendicular to the inflow direction which can result in failure if not accounted for. The fin has also served the purpose of keeping the flow's velocity high on the downstream side of the cylinder. This can be seen by simply comparing Figure 1 and Figure 2.

We can see from comparing Figure 2 and Figure 5 that the temperature fluctuates at the same frequency in the two simulations but in Figure 5 the temperature peaks are higher at locations downstream than their counterparts from Figure 2. This reflects the higher level of heat transfer taking place in the 2nd simulation that we ran. By comparing Figures 3 and 6 we can see that they are very similar. This tells us that the fin has little effect on the flow's velocity overall, which makes sense from the perspective of mass conservation. The fin does however speed up the flow on the downstream edge of the cylinder. This 'tactical' flow acceleration makes a big difference in the overall heat transfer value.

A possible next step for this simulation is to simulate in 3D or to change the simulation to incorporate compressible and high Reynolds number flows. We have made a number of simplifications in order to write and execute this code in a manageable time frame but this takes away from the versatility of the simulation.

A Code Appendix

A.1 Main Code (Matlab)

```matlab
close all

tic;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
part_two = true;
skip_to_time_steps = false;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

if skip_to_time_steps
    if part_two
        load("./part-2-data/part-2-workspace-time-step-20000.mat");
    else
        load("./data/workspace-time-step-20000.mat");
    end

    print_time_step_frequently = false;
    security_number = 1000;

    old_num_time_steps = num_time_steps;

    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    num_time_steps = 20200;
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    starting_time_step = time_step + 1;

    if num_time_steps > old_num_time_steps
        diff_time_steps = num_time_steps - old_num_time_steps;
        total_transfer = padarray(total_transfer, diff_time_steps, 0, 'post');
    end

    figure(1)
    set(gcf, 'visible', 'off')
    set(gcf, 'Position',  [0, 0, 1080, 1080])
    clf;
```

```
else
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    num_time_steps = 20;
    frame_multiple = 5;
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

    starting_time_step = 1;

    height = 200;
    width = 500;

    print_time_step_frequently = false;
    security_number = 1000;

    total_transfer = zeros(num_time_steps, 2);

    cylinder_diameter = 50;
    cylinder_radius = cylinder_diameter / 2;
    cylinder_center_x = height / 2;
    cylinder_center_y = height / 2;

    error_limit = 0.01;                   % 1% maximum change for convergence

    U_inf = 5;                            % m/s      uniform inflow
    alpha = 22.07 * 10^(-6);              % m^2/s    Thermal Diffusivity at 300K
    k = 0.02624;                          % W/(m*K)  Thermal Conductivity at 300K
    nu = 1.48 * 10^(-5);                  % m^2/s    Kinematic Viscosity at 300K
    F = 1.8;                              %          over-relaxation factor
    free_lid = U_inf * (height / 2);      %          free-lid streamfunction constant

    Re_D = 200;                           % Given Reynolds number

    T_surface = 400;                      % K
    T_boundary = 300;                     % K
    T_init = min(T_surface, T_boundary); % Bulk fluid initial temp
```

```
    h_1 = (10 - 1) * nu / U_inf;
    h_2 = (10 - 1) * alpha / U_inf;
    h = min(h_1, h_2);                      % grid spacing

    U_max = 5 * U_inf;

    dt = (h / U_max) / 2;


    disp("h  = " + h);
    disp("dt = " + dt);


    omega = zeros(width, height);
    psi = zeros(width, height);
    temps = zeros(width, height);

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


    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    % Setup
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

    solid_points = zeros(width, height);
    for i = 1:width
        for j = 1:height
            dist = sqrt((i - cylinder_center_x)^2 + (j - cylinder_center_y)^2);
            if dist <= cylinder_radius
                solid_points(i, j) = 1;
                temps(i, j) = T_surface;
            else
                temps(i, j) = T_boundary;
    %             psi = U_inf * j - free_lid;
                psi(i, j) = (U_inf * j - free_lid) * h;
            end

            if part_two
                if ((j == (height / 2)) && (i < 150) && (i > 100))
                    solid_points(i, j) = 1;
                    temps(i, j) = T_surface;
```

```
                end
            end


        end
    end


    solid_adj_points = zeros(width, height);
    for i = 2:(width - 1)
        for j = 2:(height - 1)
            if ~solid_points(i, j)
                if (solid_points(i - 1, j) || solid_points(i + 1, j) || solid_points(i, j - 1) || solid_points(i, j + 1))
                    solid_adj_points(i, j) = 1;
                end
            end
        end
    end






    u(1,:) = U_inf;







    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    % Gauss-Seidel relaxation of Psi at t = 0
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

    error_flag = true;
    while error_flag
        psi_old = psi;

        for i = 2:(width - 1)
            for j = 2:(height - 1)
                if ~solid_points(i,j)
                    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

  %      psi(0, :) = psi(3, :);

        error_array = abs(psi - psi_old) ./ psi_old;
        error_array(isnan(error_array)) = 0;

        error_term = max(error_array);

        if (error_term <= error_limit)
            error_flag = false;
        end

    end



    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    % Plot for t = 0
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    figure(1)
    set(gcf, 'visible', 'off')
    set(gcf, 'Position',  [0, 0, 1080, 1080])
    ax(1) = subplot(3,1,1);
    hold on
    plot_data = flipud(rot90(psi));
    s = pcolor(plot_data);
    daspect([1 1 1]);
    colormap(ax(1), gray);
    set(s, 'EdgeColor', 'none');
    colorbar
    contour(plot_data, 32, 'black');
    title("Streamfunction");
    hold off





    ax(2) = subplot(3,1,2);
    hold on
    plot_data = flipud(rot90(omega));
```

```
    s = pcolor(plot_data);
    daspect([1 1 1]);
    colormap(ax(2), gray);
    set(s, 'EdgeColor', 'none');
    colorbar
    title("Vorticity");
    hold off



    ax(3) = subplot(3,1,3);
    hold on
    plot_data = flipud(rot90(temps));
    s = pcolor(plot_data);
    daspect([1 1 1]);
    colormap(ax(3), jet);
    set(s, 'EdgeColor', 'none');
    colorbar
    title("Temperature");
    hold off


    real_time = 0;
    time_string = sprintf('%0.8f seconds', real_time);
    xlabel({" ", " ", time_string});


    if part_two
        file_name = sprintf("./part-2-images/Final-Project-%d.
   png", 0);
    else
        file_name = sprintf("./images/Final-Project-%d.png", 0);
    end
    saveas(gcf, file_name);

    clf;

end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Time Steps
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


for time_step = starting_time_step:num_time_steps
```

```
    omega_old = omega;
    temps_old = temps;

    % omega_wall setup
    for i = 2:(width - 1)
        for j = 2:(height - 1)
            if solid_points(i, j)
                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

    % calculate u, v matrices
    for i = 2:(width - 1)
        for j = 2:(height - 1)
            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);
        end
    end

    u(:,1)=U_inf;
    u(:,height)=U_inf;

    % Bulk fluid calculations
    for i = 2:(width - 1)
        for j = 2:(height - 1)
            if ~solid_points(i, j)
                laplacian_vorticity = (omega_old(i - 1, j) +
omega_old(i + 1, j) + omega_old(i, j - 1) + omega_old(i, j +
1) - 4 * omega_old(i, j)) / (h * h);

                delta_u_omega = 0;
                if (u(i, j) < 0)
                    delta_u_omega = u(i + 1, j) * omega_old(i +
1, j) - u(i, j) * omega_old(i, j);
                elseif (u(i, j) > 0)
                    delta_u_omega = u(i, j) * omega_old(i, j) -
u(i - 1, j) * omega_old(i - 1, j);
                end

                delta_v_omega = 0;
                if (v(i, j) < 0)
                    delta_v_omega = v(i, j + 1) * omega_old(i, j
 + 1) - v(i, j) * omega_old(i, j);
```

```
                elseif (v(i, j) > 0)
                    delta_v_omega = v(i, j) * omega_old(i, j) -
    v(i, j - 1) * omega_old(i, j - 1);
                end

                omega(i, j) = omega_old(i, j) + dt * (-
    delta_u_omega / h - delta_v_omega / h + nu *
    laplacian_vorticity);
            end
        end
    end


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


    % Gauss-Seidel relaxation of psi (with omega term)
    error_flag = true;
    while error_flag
        psi_old = psi;

        for i = 2:(width - 1)
            for j = 2:(height - 1)
                if ~solid_points(i,j)
                    psi(i, j) = psi_old(i, j) + (F / 4) * (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));
                end
            end
        end


        error_array = abs(psi - psi_old) ./ psi_old;
        error_array(isnan(error_array)) = 0;

        error_term = max(error_array);

        if (error_term <= error_limit)
            error_flag = false;
        end
    end

    % temperature update
    for i = 2:(width -1)
```

```
        for j = 2:(height - 1)
            if ~solid_points(i, j)
                laplacian_temps = (temps_old(i - 1, j) + temps_old(i + 1, j) + temps_old(i, j - 1) + temps_old(i, j + 1) - 4 * temps_old(i, j)) / (h * h);

                u_delta_T = 0;
                if u(i, j) < 0
                    u_delta_T = u(i, j) * (temps_old(i + 1, j) - temps_old(i, j));
                elseif u(i, j) > 0
                    u_delta_T = u(i, j) * (temps_old(i, j) - temps_old(i - 1, j));
                end

                v_delta_T = 0;
                if v(i, j) < 0
                    v_delta_T = v(i, j) * (temps_old(i, j + 1) - temps_old(i, j));
                elseif v(i, j) > 0
                    v_delta_T = v(i, j) * (temps_old(i, j) - temps_old(i, j - 1));
                end

                temps(i, j) = temps_old(i, j) + dt * (-u_delta_T / h - v_delta_T / h + alpha * laplacian_temps);

            end
        end
    end



    if mod(time_step, frame_multiple) == 0
        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        % Plot Streamfunction, Vorticity, and Temperature
        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

        ax(1) = subplot(3,1,1);
        hold on
        plot_data = flipud(rot90(psi));
        s = pcolor(plot_data);
        daspect([1 1 1]);
        colormap(ax(1), gray);
        set(s, 'EdgeColor', 'none');
```

```
        colorbar
        contour(plot_data, 32, 'black');
        title("Streamfunction");
        hold off






        ax(2) = subplot(3,1,2);
        hold on
        plot_data = flipud(rot90(omega));
        s = pcolor(plot_data);
        daspect([1 1 1]);
        colormap(ax(2), gray);
        set(s, 'EdgeColor', 'none');
        colorbar
        title("Vorticity");
        hold off



        ax(3) = subplot(3,1,3);
        hold on
        plot_data = flipud(rot90(temps));
        s = pcolor(plot_data);
        daspect([1 1 1]);
        colormap(ax(3), jet);
        set(s, 'EdgeColor', 'none');
        colorbar
        title("Temperature");
        hold off




        real_time = dt * time_step;
        time_string = sprintf('%0.8f seconds', real_time);
        xlabel({" ", " ", time_string});


        if part_two
            file_name = sprintf("./part-2-images/Final-Project-%
    d.png", time_step);
```

```
        else
            file_name = sprintf("./images/Final-Project-%d.png",
    time_step);
        end

        saveas(gcf, file_name);

        clf;

    end


    total_transfer(time_step, 1) = dt * time_step;
    transfer_sum = 0;
    for i = 1:width
        for j = 1:height
            if solid_adj_points(i, j)
                transfer_sum = transfer_sum + (temps(i, j) -
    temps_old(i, j)) / h;
            end
        end
    end
    total_transfer(time_step, 2) = -k * transfer_sum;




    if mod(time_step, security_number) == 0
        if part_two
            data_file_name = sprintf("./part-2-data/workspace-
    time-step-%d.mat", time_step);
        else
            data_file_name = sprintf("./data/workspace-time-step
    -%d.mat", time_step);
        end
        save(data_file_name);
    end


    if print_time_step_frequently
        disp("Time step " + time_step + " of " + num_time_steps)
    elseif mod(time_step, frame_multiple) == 0
        disp("Time step " + time_step + " of " + num_time_steps)
    end
end
```

```
figure(2)
plot(total_transfer(:, 1), total_transfer(:, 2));
xlabel("Time (s)");
ylabel("Total Heat Transfer (W)");
title("Total Heat Transfer from Cylinder");

if part_two
    file_name = "./Total-Heat-Transfer-Part-2.png";
else
    file_name = "./Total-Heat-Transfer.png";
end
saveas(gcf, file_name);


toc;
```

A.2 Video Generation Code (Python)

```python
import re
import cv2
import os

import time
start_time = time.time()



image_folder = "C:\\Users\\samda\\Documents\\GitHub\\Heat-Transfer\\Final-Project-v2\\images"
output_folder = "C:\\Users\\samda\\Documents\\GitHub\\Heat-Transfer\\Final-Project-v2\\"
video_name = output_folder + "final-project.mp4"
fps = 10


part_two = false
if part_two)
    image_folder = "C:\\Users\\samda\\Documents\\GitHub\\Heat-Transfer\\Final-Project-v2\\part-2-images"
    output_folder = "C:\\Users\\samda\\Documents\\GitHub\\Heat-Transfer\\Final-Project-v2\\"
    video_name = output_folder + "final-project-part-2.mp4"




##############################################################
#  Generate Video
##############################################################

def generate_video():
    def atoi(text):
        # https://stackoverflow.com/questions/5967500/how-to-correctly-sort-a-string-with-a-number-inside
        return int(text) if text.isdigit() else text

    def natural_keys(text):
        # https://stackoverflow.com/questions/5967500/how-to-correctly-sort-a-string-with-a-number-inside
        return [ atoi(c) for c in re.split(r'(\d+)', text) ]

```

```python

    images = [img for img in os.listdir(image_folder) if img.endswith(".png")]
    images.sort(key=natural_keys)
    frame = cv2.imread(os.path.join(image_folder, images[0]))
    height, width, layers = frame.shape

    fourcc = cv2.VideoWriter_fourcc(*'mp4v')
    video = cv2.VideoWriter(video_name, fourcc, fps, (width, height))

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

    cv2.destroyAllWindows()
    video.release()

    print("\n--- Video Done ---")
    print("--- %.6f seconds ---" % (time.time() - start_time))




if __name__ == "__main__":
    generate_video()
```