Simulating Fluid Flow

Across a Circular Object in a Wind Tunnel Environment

with

Computational Fluid Dynamics

Wes Scarpa

Simulations in Physics Final Project

December 2, 2024

Introduction:

Understanding how air flows around objects is crucial in many scientific and engineering fields, including aerospace, automotive design, and environmental modeling. Wind tunnels are a traditional tool for studying such interactions, enabling the analysis of aerodynamic properties like drag, lift, and pressure distribution. However, physical wind tunnels are often expensive and time-intensive to operate. Computational Fluid Dynamics (CFD) offers a powerful alternative, leveraging numerical methods to simulate fluid flows efficiently and flexibly. This project builds upon CFD principles to develop an interactive simulation of two-dimensional, non-turbulent flow around a circular obstacle.

The simulation models the behavior of an incompressible fluid using the Navier-Stokes equations, which describe fluid motion under the influence of forces like viscosity and pressure gradients. To solve these equations, the project employs Chorin's Projection Method, a widely used numerical approach that decouples velocity and pressure calculations for computational efficiency. The wind tunnel is discretized into a grid, with each segment representing localized pressure and velocity values. Finite-difference methods approximate the spatial and time derivatives required for the simulation, enabling accurate modeling of steady-state flow.

A key aspect of the simulation is the interaction between the fluid and solid objects. To ensure the fluid behaves accurately it is important to impose proper boundary conditions. Both the obstacle and the wind tunnel walls require set pressure and velocity conditions. For the walls of the wind tunnel and the surface of the obstacle Dirichlet velocity boundary conditions ensure a non-slip condition of the fluid over the surfaces. Neumann boundary conditions on pressure give solidity to the obstacle by making sure there is no fluid flow across the boundary of a solid. Neumann and Dirichlet boundary conditions ensure a realistic fluid-solid interaction of the fluid to the obstacle and tunnel walls.

The project emphasizes interactivity, allowing for the adjustment of several parameters to explore their effects on fluid behavior. These include initial wind tunnel velocity (the wind speed at the inlet), the radius and location of the circular obstacle, the time step for numerical calculations, the number of velocity iterations, and the resolution of the computational grid. By providing these controls, the simulation serves as a versatile platform for investigating how different initial conditions influence steady-state flow patterns.

The primary outputs of the simulation are visualizations of velocity vector fields, pressure distributions, and flow patterns around the circular obstacle. These visualizations are essential for understanding fluid behavior at different Reynolds Numbers, where the flow remains non-turbulent. The results are validated against theoretical predictions and benchmark cases, ensuring the accuracy and reliability of the simulation.

This project highlights the practical applications of CFD in modeling and analyzing fluid dynamics, bridging theoretical principles with computational techniques. By integrating interactive features, rigorous numerical methods, and detailed visualizations, the simulation serves as a valuable tool for studying non-turbulent fluid flow and demonstrates the power of computational simulations in addressing real-world challenges.

Methods:

The main computational methods used by this simulation are to solve the Navier-Stokes equation. The wind tunnel simulator solves the incompressible Navier-Stokes equations in a two-dimensional domain using a finite-difference approach and Chorin's Projection Method. The numerical methods for solving the Navier-Stokes equations were largely based on a Computational Fluid Dynamics project found on GitHub that was adapted into this simulation (Creyon). The simulation models the steady-state flow of an incompressible non-turbulent fluid around an obstacle, discretizing the computational domain into a structured grid and applying explicit time-stepping. The equations governing the fluid flow are:

Momentum Equation: $\frac{\partial \boldsymbol{u}}{\partial t} + (\boldsymbol{u} \cdot \nabla)\boldsymbol{u} = -\frac{1}{\rho}\nabla p + v\nabla^2 \boldsymbol{u} + f$

Incompressibility Constraint: $\nabla \cdot u = 0$

Where:

$\boldsymbol{u}$: Velocity vector

p: Pressure

f: External forcing term (set to 0 in this simulation)

v: Kinematic viscosity (set to 0.1) – significantly higher than air but air is very easily turbulent so this value allows for a wider range of simulations

ϱ: Fluid density (1 for this fluid)

∇: Nabla operator (describing gradient, divergence, and convection terms)

∇^2: Laplace operator

Before going through the steps for Chorin's Projection method to solve the Navier-Stokes equations, let's define the set up and basic methods we will need.

The wind tunnel is discretized into a uniform square grid of size N by N with 2-D arrays that hold values for pressure, horizontal, and vertical velocity for every cell of the grid. The spatial derivatives for each value are approximated using finite differences:

```
# estimates derivative of f(velocity/pressure) in x direction at the center of the tile
def central_difference_x(f):
    diff = np.zeros_like(f)
    diff[1:-1, 1:-1] = (
        f[1:-1, 2:  ]
        -
        f[1:-1, 0:-2]
    ) / (
        2 * element_length
    )
    return diff
```

This method takes an array f, which can be pressure or velocity, and returns an array that holds the derivative in the x at every point not along the borders by taking the difference

between the value one square to the left and the value one square to the right and dividing by the distance from the center of the block on either side (2*element_legnth).

```
# estimates derivative of f(velocity/pressure) in y direction at the center of the tile
def central_difference_y(f):
    diff = np.zeros_like(f)
    diff[1:-1, 1:-1] = (
        f[2:  , 1:-1]
        -
        f[0:-2, 1:-1]
    ) / (
        2 * element_length
    )
    return diff
```

This similarly takes the derivative of array f in the y direction.

The Laplace operator is a second-order partial differential operator which describes the divergence of the gradient of a function. To create a discrete approximation of the Laplace operator in the context of a 2D grid, it approximates the second derivatives of the function in both directions (x and y), and is defined as: $\nabla^2 f(x,\ y)\ =\ \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2}$.

In discrete terms, this can be computed as the sum of the differences between neighboring grid points: $\nabla^2 f(x,\ y)\ =\ \frac{f(i-1,j)+f(i+1,j)+f(i\ ,j-1)+f(i\ ,j+1)-4\cdot f(i,j)}{d^2}$ with d being the element length.

```python
#laplace operator
def laplace(f):
    diff = np.zeros_like(f)
    diff[1:-1, 1:-1] = (
        f[1:-1, 0:-2]
        +
        f[0:-2, 1:-1]
        -
        4
        *
        f[1:-1, 1:-1]
        +
        f[1:-1, 2:  ]
        +
        f[2:  , 1:-1]
    ) / (
        element_length**2
    )
    return diff
```

This method takes an array f and computes and returns the array of the discrete Laplacian of f at every point in the wind tunnel, not along the borders.

To solve for the pressure field, vertical and horizontal vector fields the simulation iterates using Chorin's Projection Method. The idea behind Chorin's projection is to break up solving the Navier-Stokes equations into computable steps. The first step being taking the original momentum equation: $\frac{\partial \boldsymbol{u}}{\partial t} + convection = - \ pressure\ gradient + diffusion$, rewrite it treating it as an ordinary differential equation: $\frac{\partial \boldsymbol{u}}{\partial t} = \ - (\boldsymbol{u} \cdot \nabla)\boldsymbol{u} - \frac{1}{\rho}\nabla p + v\nabla^2\boldsymbol{u}$. This differential can then be discretized to $\frac{\boldsymbol{u}^{(t+1)} - \boldsymbol{u}^t}{\Delta t} = - (\boldsymbol{u}^t \cdot \nabla)\boldsymbol{u}^t - \frac{1}{\rho}\nabla p^{t+1} + v\nabla^2\boldsymbol{u}^{t+1}$, using a fully explicit convection term and fully implicit diffusion and pressure gradient terms as recommended by sources (https://www.youtube.com/watch?v=JBmS--3L2eQ&t=6s). Now Chorin's Projection method uses an intermediate tentative velocity that disregards the pressure gradient for the first step and then adjusts for the pressure next step, thus the equation becomes: $\frac{\boldsymbol{u}^* - \boldsymbol{u}^t}{\Delta t} = - (\boldsymbol{u}^t \cdot \nabla)\boldsymbol{u}^t + v\nabla^2\boldsymbol{u}^*$ where $\boldsymbol{u}^*$ is the tentative velocity. Ultimately this is an operator

splitting where if $\frac{\boldsymbol{u}^{(t+1)}-\boldsymbol{u}^{t}}{\Delta t} = -(\boldsymbol{u}^{t} \cdot \nabla)\boldsymbol{u}^{t} - \frac{1}{\rho}\nabla p + v\nabla^{2}\boldsymbol{u}^{t+1}$ and $\frac{\boldsymbol{u}^{*}-\boldsymbol{u}^{t}}{\Delta t} = -(\boldsymbol{u}^{t} \cdot \nabla)\boldsymbol{u}^{t} + v\nabla^{2}\boldsymbol{u}^{*}$ then $\frac{\boldsymbol{u}^{t+1}-\boldsymbol{u}^{*}}{\Delta t} = -\frac{1}{\rho}\nabla p^{t+1}$. Now, the last element needed to find $\boldsymbol{u}^{t+1}$ and allow for the iteration of velocity over time is the pressure gradient term. The approach used in the simulation to solve for the pressure gradient term was to solve a pressure Poisson problem. To do this we take the divergence of both sides of $\frac{\boldsymbol{u}^{t+1}-\boldsymbol{u}^{*}}{\Delta t} = -\frac{1}{\rho}\nabla p^{t+1}$: $\nabla(\frac{\boldsymbol{u}^{t+1}-\boldsymbol{u}^{*}}{\Delta t}) = \nabla(-\frac{1}{\rho}\nabla p^{t+1})$. This simplifies to $(\frac{\nabla \boldsymbol{u}^{t+1}}{\Delta t}) - (\frac{\nabla \boldsymbol{u}^{*}}{\Delta t}) = -\frac{1}{\rho}\nabla^{2}p^{t+1}$. Since the updated value of velocity $\boldsymbol{u}^{t+1}$ should be incompressible by the incompressibility equation ($\nabla \cdot u = 0$) the equation will simplify further to $0 - (\frac{\nabla \boldsymbol{u}^{*}}{\Delta t}) = -\frac{1}{\rho}\nabla^{2}p^{t+1}$ therefore $\frac{\nabla \boldsymbol{u}^{*}}{\Delta t} = \frac{1}{\rho}\nabla^{2}p^{t+1}$ allowing the pressure field to be solved for.

Now we can iterate the velocity field throughout the wind tunnel with three steps:

1) Solve for tentative velocity: $\frac{\boldsymbol{u}^{*}-\boldsymbol{u}^{t}}{\Delta t} = -(\boldsymbol{u}^{t} \cdot \nabla)\boldsymbol{u}^{t} + v\nabla^{2}\boldsymbol{u}^{*}$

   $\boldsymbol{u}^{*} = (-(\boldsymbol{u}^{t} \cdot \nabla)\boldsymbol{u}^{t} + v\nabla^{2}\boldsymbol{u}^{*})\Delta t + \boldsymbol{u}^{t}$

2) Solve for pressure gradient: $\frac{\nabla \boldsymbol{u}^{*}}{\Delta t} = \frac{1}{\rho}\nabla^{2}p^{t+1}$

3) Solve for next velocity: $\frac{\boldsymbol{u}^{t+1}-\boldsymbol{u}^{*}}{\Delta t} = -\frac{1}{\rho}\nabla p^{t+1}$

   $\boldsymbol{u}^{t+1} = (-\frac{1}{\rho}\nabla p^{t+1})\Delta t + \boldsymbol{u}^{*}$

```
# Perform a tentative step by solving the momentum equation without the
# pressure gradient
u_tent = (
    u_prev
    +
    dt * (
        -
        (
            u_prev * d_u_prev__d_x
            +
            v_prev * d_u_prev__d_y
        )
        +
        kinematicViscosity * laplace__u_prev
    )
)
v_tent = (
    v_prev
    +
    dt * (
        -
        (
            u_prev * d_v_prev__d_x
            +
            v_prev * d_v_prev__d_y
        )
        +
        kinematicViscosity * laplace__v_prev
    )
)
```

This is step 1, solving for the horizontal and vertical tentative velocity components, in code.

```
# Compute a pressure correction by solving the pressure-poisson equation
rhs = (
    densityRho / dt
    *
    (
        d_u_tent__d_x
        +
        d_v_tent__d_y
    )
)


# iterate pressure poisson
for _ in range(nPressurePoissonIter):
    p_next = np.zeros_like(p_prev)
    p_next[1:-1, 1:-1] = 1/4 * (
        +
        p_prev[1:-1, 0:-2]
        +
        p_prev[0:-2, 1:-1]
        +
        p_prev[1:-1, 2:  ]
        +
        p_prev[2:  , 1:-1]
        -
        element_length**2
        *
        rhs[1:-1, 1:-1]
    )
```

This is step 2, solving for the pressure gradient, in code.

```
# Correct the velocities such that the fluid stays incompressible
u_next = (
    u_tent
    -
    dt / densityRho
    *
    d_p_next__d_x
)
v_next = (
    v_tent
    -
    dt / densityRho
    *
    d_p_next__d_y
)
```

This is step 3, correcting to find the next velocity fields after the time step, in code.

Lastly, we need to ensure the interaction between the fluid and the solid objects in the simulation, specifically the walls of the wind tunnel and the circular obstacle. To achieve this, we implement appropriate boundary conditions that govern how the fluid behaves at these surfaces.

For the walls of the wind tunnel, the boundary conditions ensure that the fluid does not penetrate through the solid boundaries, reflecting the physical no-penetration condition. We set the inlet velocity at the left boundary to a constant initial value. This models the fluid entering the wind tunnel with a specified velocity profile. For the walls of the wind tunnel, we apply Dirichlet boundary conditions, setting the velocity of the fluid to zero at the walls, effectively imposing a no-slip condition. This means that the fluid velocity at the boundary is equal to zero, simulating the interaction of the fluid with solid walls where the fluid adheres to the surface.

```
# Velocity Boundary Conditions: Homogeneous Dirichlet BC everywhere
# except for the horizontal velocity at the left, which is prescribed
u_tent[0, :] = 0.0  #bottom
u_tent[:, 0] = vxInitialLeft  #left
u_tent[:, -1] = 0.0 #right
u_tent[-1, :] = 0.0 #top
v_tent[0, :] = 0.0  #bottom
v_tent[:, 0] = vyInitialLeft #left
v_tent[:, -1] = 0.0 #right
v_tent[-1, :] = 0.0 #top
```

```
# Velocity Boundary Conditions: Homogeneous Dirichlet BC everywhere
# except for the horizontal velocity at the left, which is prescribed
u_next[0, :] = 0.0  #bottom
u_next[:, 0] = vxInitialLeft  #left
u_next[:, -1] = 0.0 #right
u_next[-1, :] = 0.0 #top
v_next[0, :] = 0.0  #bottom
v_next[:, 0] = vyInitialLeft #left
v_next[:, -1] = 0.0 #right
v_next[-1, :] = 0.0 #top
```

This applies the Dirichlet and prescribed boundary conditions for horizontal and vertical velocity. It is applied before and after the pressure correction to make sure the fluid is always behaving properly.

Boundary conditions across the surface of the wind tunnel must also be set for pressure. The outlet on the right side is set to a prescribed pressure of 0, so the fluid will flow out of the tunnel. All other sides of the tunnel use Neumann boundary conditions which are that the normal gradient of the pressure is 0 at the boundaries: $\frac{\partial p}{\partial n} = 0$. This enforces that there is a solid boundary, since there is no change in pressure across the surface of the border it defines an edge where there will be no flow across. To make the normal gradient equal to 0, the pressure at every point on the boundary is set equal to the pressure of its neighboring point that is normal to the surface.

```
# Pressure Boundary Conditions: Homogeneous Neumann Boundary
p_next[:, -1] = 0 #right
p_next[0,  :] = p_next[1,  :] #bottom
p_next[:,  0] = p_next[:,  1] #left
p_next[-1, :] = p_next[-2, :] #top
p_prev = p_next
```

This sets the outlet pressure to 0 and the pressure along all boundaries equal to the pressure of the neighboring cell normal to the boundary.

Similarly, we have to apply both velocity and pressure boundary conditions to the circular obstacle. Taking the same approach we set the velocity across the surface of the circle to 0 and the normal gradient of the pressure to 0.

```
#no slip condition on boundary by setting velocity to 0
def apply_circle_boundary_conditions(u, v, X, Y, center, radius):

    distance_from_center = np.sqrt((X - center[0])**2 + (Y - center[1])**2)
    mask = distance_from_center <= radius
    u[mask] = 0
    v[mask] = 0
    return u, v
```

To define the surface of a circle that is represented by a square grid point, a boolean array is created that holds a true value for every grid point whose center falls within the circle's boundary. Then all it sets the velocity of every marked cell to 0 giving it proper Dirichlet boundary conditions.

```python
#Pressure boundary for the object in the windtunnel: no derivative,
# set the pressure at the border to average of the pressures around the tile:
# thus derivative with be close to 0
#(zero normal gradient) for pressure on the circle
def apply_pressure_boundary_conditions(p, X, Y, center, radius):

    distance_from_center = np.sqrt((X - center[0])**2 + (Y - center[1])**2)
    mask = distance_from_center <= radius

    for i in range(1, p.shape[0] - 1):
        for j in range(1, p.shape[1] - 1):
            if mask[i, j]:
                # Average pressure from neighboring cells outside the circle
                neighbors = []
                if not mask[i+1, j]: neighbors.append(p[i+1, j])
                if not mask[i-1, j]: neighbors.append(p[i-1, j])
                if not mask[i, j+1]: neighbors.append(p[i, j+1])
                if not mask[i, j-1]: neighbors.append(p[i, j-1])
                p[i, j] = np.mean(neighbors)
    return p
```

For the Neumann condition, the pressure on the border of the object is set to the average of the cells outside the circle that it touches. This approximates the gradient of pressure to 0 but is not perfectly precise causing the boundary of the circle to have odd shapes when the resolution is low, making simulations at low resolution less accurate.

Establishing these boundary conditions allows the fluid to properly interact with the solid objects as expected. To ensure they are working correctly and that the simulation was producing accurate and realistic results, the program was verified in several ways.

Verification of Program:

To verify the accuracy and functionality of the simulation, several tests were conducted and compared with known results and theoretical expectations.

One of the primary methods of verification involved increasing the number of grid points (nPoints) and analyzing how this impacted the resolution and accuracy of the flow visualization. When the simulation was run with a grid of 75 by 75 points (nPoints=75), and the time step (dt) was decreased to 0.0001 to maintain stability according to the safety factor, the results

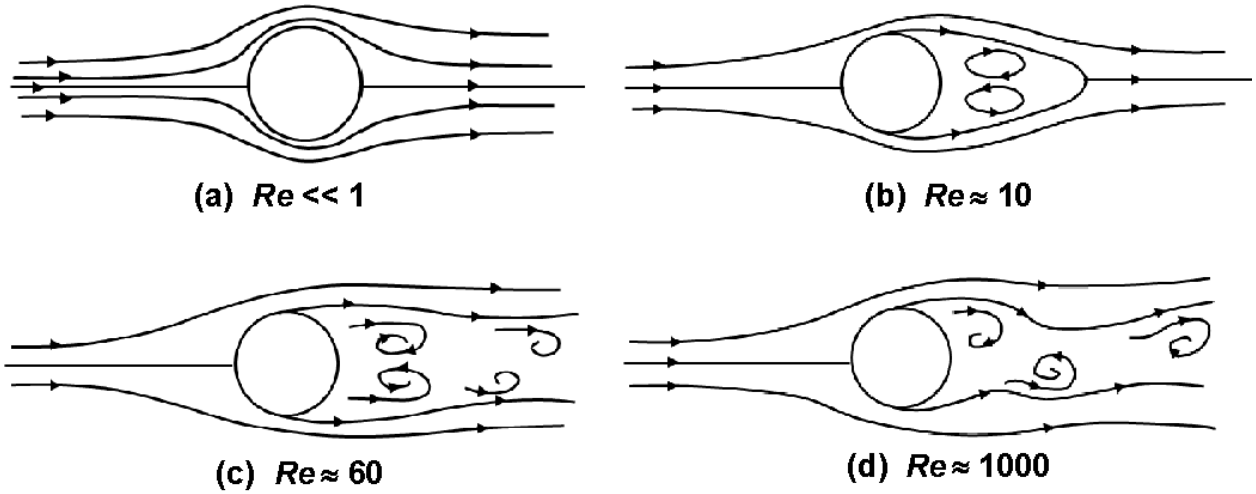
demonstrated significant improvements in the simulation's accuracy and visual representation. The higher resolution produced a much more detailed and smoother pressure gradient, with a clearer depiction of fluid flow. The increased resolution enhanced the accuracy of the velocity vectors and provided a more visually accurate representation of the fluid behavior.

Additionally, the circular object impeding the flow became more defined, with the shape appearing more circular. This refinement allowed the flow and pressure distribution around the object to be smoother and better resolved, reflecting the physical behavior of fluid dynamics. The improvement also enabled the simulation to produce accurate models for higher Reynolds Numbers, as without increasing the resolution there are overflow errors when the Reynolds Number gets too high. This allows comparisons of flow patterns at specific Reynolds Numbers to the expected behaviors given the object shape. These comparisons confirmed that the results aligned with theoretical predictions, demonstrating the accuracy and reliability of the model as the grid resolution increased. These results provide confidence that the numerical method and implementation of boundary conditions are accurate for simulating fluid dynamics in the wind tunnel.

This improvement in the simulation's performance with higher resolution grid points supports the idea that the model produces more accurate results as the grid is refined. The smoother pressure gradient and more defined object shape validate that the code is functioning correctly, as expected by physical principles. Looking at figure 1 in the data section we can see how the simulation converges to being more accurate when the resolution is increased.

For further verification, the results of the simulation were compared to previous studies on the fluid flow and pressure fields over a circular object to ensure that the simulation was producing the expected patterns and values for a given initial condition. To compare, several trials were done plotting the flow over a circle at the center of the wind tunnel with varied initial conditions of Reynolds Number (based on mostly velocity), grid resolution, time step, and iterations.

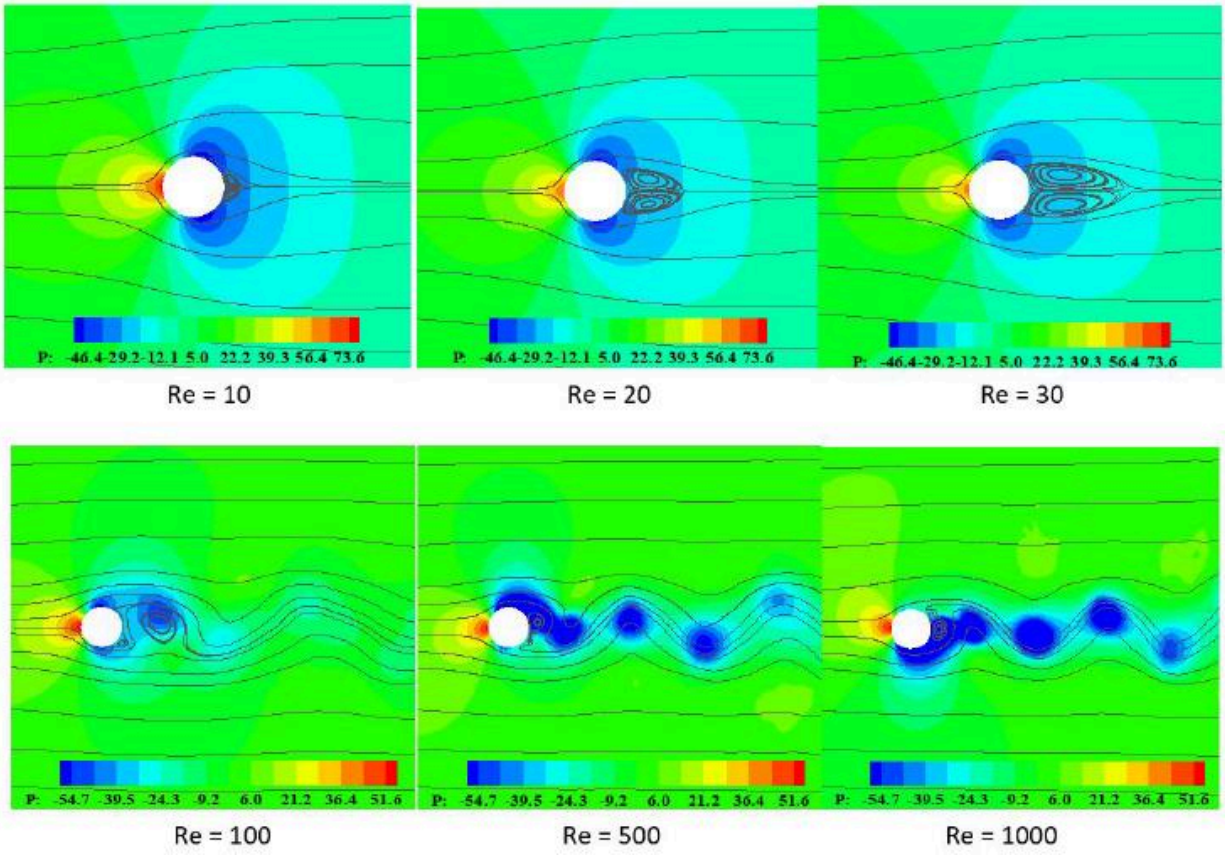
Typically fluid becomes turbulent at a Reynolds Number around 4000, but when the fluid flows over a circular object it significantly lowers the Reynolds Number that the fluid becomes disturbed at. Based on a study of fluid flow past a circular cylinder, we expect the patterns of flow to change predictable at certain Reynolds Numbers. The flow patterns and the Reynolds Numbers that were found in the study are shown in the image below (Sato and Kobayashi).



This shows that low Reynolds Numbers will result in symmetric and smooth flow over the object and increasing the Reynolds Numbers will create vortices that are predictable and symmetric until further increasing the Reynolds Number when the flow becomes turbulent and less predictable. This study simulates the flow around an object in an unbounded fluid stream, instead of a wind tunnel, meaning we expect the transitional Reynolds Numbers to be significantly higher as the boundary conditions help prevent disturbance in the fluid flow. Also, the simulation calculates the Reynolds Number using the average velocity across the wind tunnel, instead of the flow of the unbounded stream that the study uses, further explaining that we will expect to find different transitional Reynolds Numbers. The closer the walls are to the object the more it will resist turbulence so to find the transitional Reynolds Number it is necessary to choose a constant center and radius. Also, since this simulation is only meant to visualize non-turbulent flow, the Reynolds Number transition calculated will only be for the first

transition when vortices start to appear. Picking a center at (0.5, 0.5) and a radius of 0.2, we find the transition point to be just below a Reynolds Number of 200. To see this look at figure 2 in the data section as the vortices just start to appear at a Reynolds Number of 195.

Additionally, to verify the visualization of the pressure field the simulation results were compared to another study on incompressible fluid interactions with rigid bodies (Tonon et al.). In the study they used computational fluid dynamics to visualize the streamlines and pressure fields over a cylindrical object at various Reynolds Numbers with the same setup as the study mentioned above, using an unbounded fluid stream. The pressure fields and streamlines found in this study are shown below.



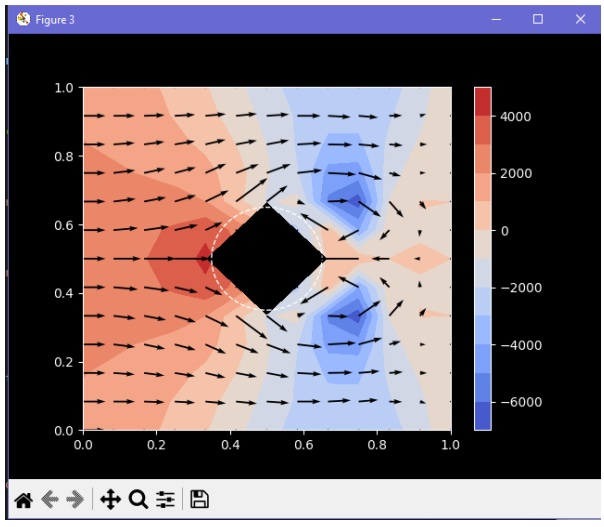
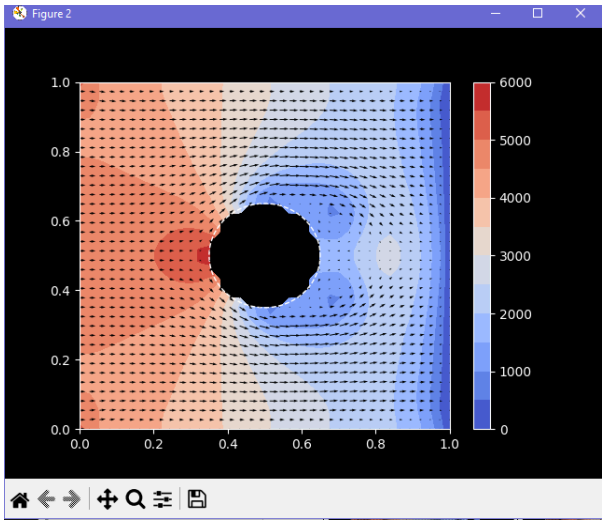
This is consistent with the study above as we see the same fluid streamlines at the same Reynolds Number. Thus when compared to this study, we expect similar pressure fields at the Reynolds Number transitional values that we found above. By looking at figure 3 and 4 we can

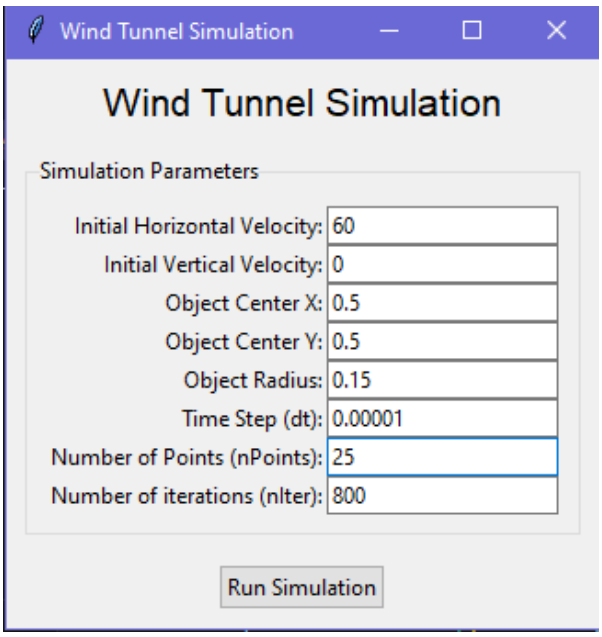
see a similar pressure field shape with different initial conditions. Picture 3a shows a very similar pressure field to the one found in the study for a Reynolds Number of 10 with high pressure at the front of the circle and the lowest pressure just past the midpoint of the circle on the top and bottom with increasing pressure as you move outward. Looking at figure 4, we see how the pressure field changes at higher reynolds when the vortices appear. The high pressure on the left side becomes smaller as the size of vortices and Reynolds Number increase just as is pictured in the second and third photo.

These studies serve as benchmarks for validating the simulation. The flow patterns and pressure gradients generated by the simulation were visually similar to those reported in the literature, reinforcing the accuracy of the numerical methods and boundary conditions used in the simulation.

Data:

Figure 1:

	
a) Re = 126.56	b) Re = 175.63



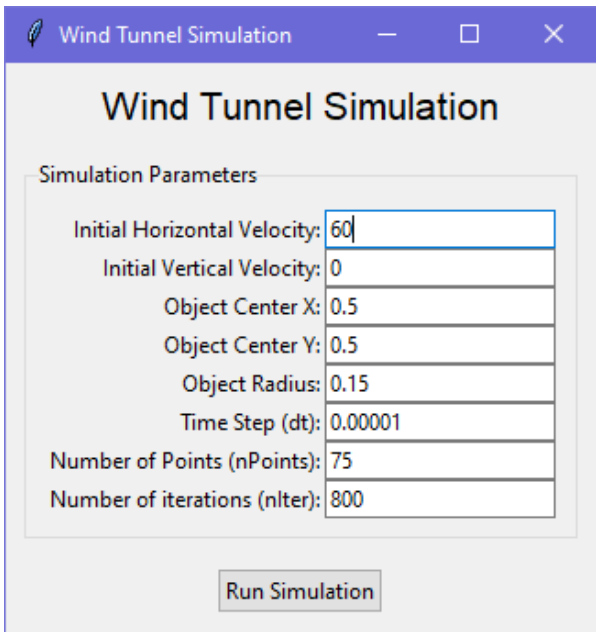
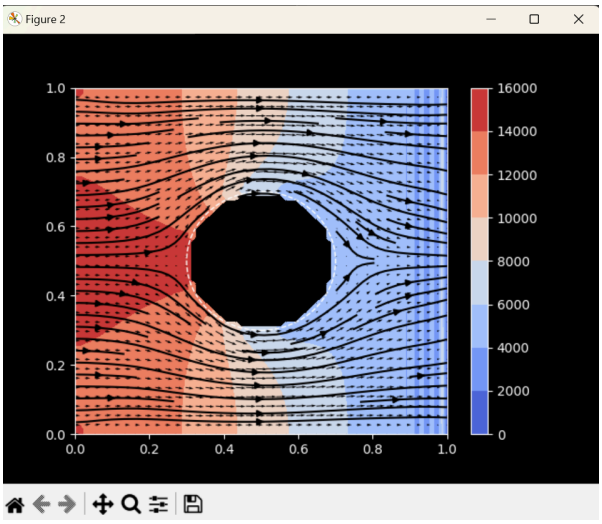
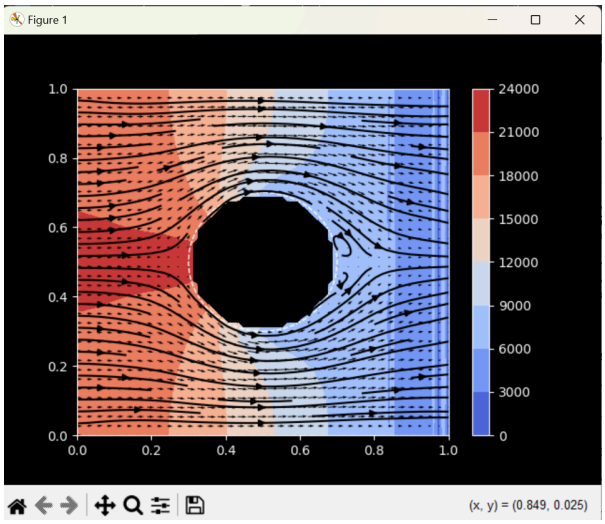


Figure 2:



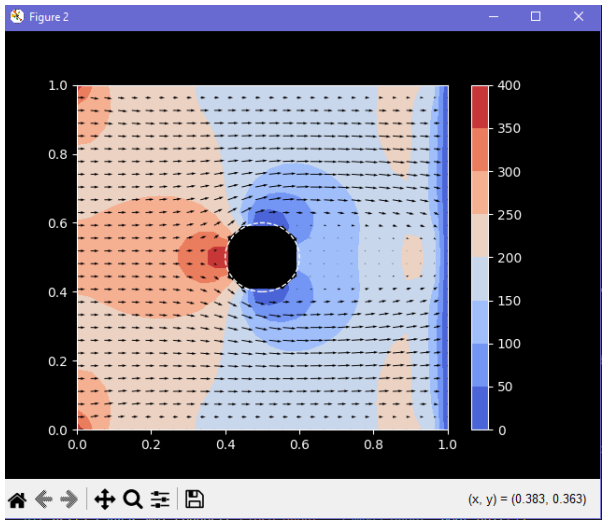
a) Re = 151.26



b) Re = 195.23

(Running with a resolution of 75x75 and time step 0.00001 varying the velocity to alter Reynolds Number)

Figure 3:

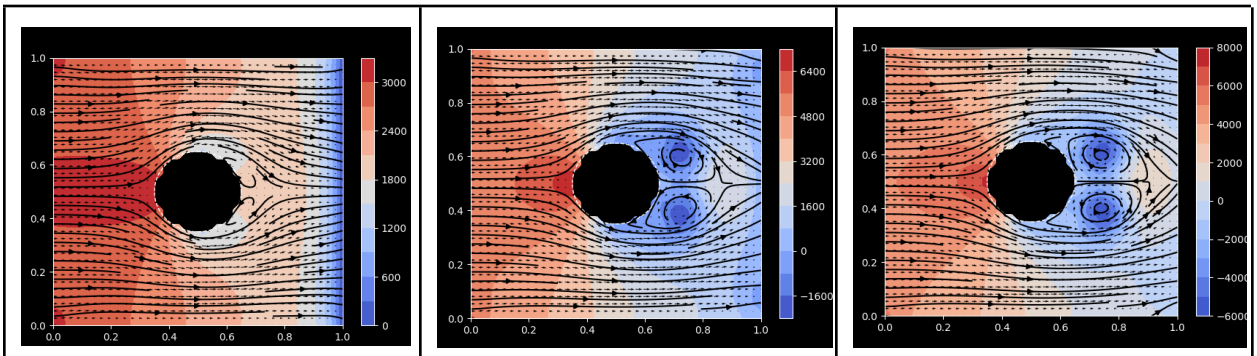


a) Re = 28.62

b) Re = 103.87

(Run with 61x61 resolution and time step 0.00001, removed streamline plot for clarity, and varied velocity to alter Reynolds Number)

Figure 4:



a) Re = 70.92

b) Re = 167.43

c) Re = 194.04

(Run with a resolution of 75x75 a time step of 0.00001 with 800 iterations a circle radius of 0.15 at varied initial velocity to alter Reynolds Number)

Analysis:

The simulation results were analyzed to identify both qualitative and quantitative relationships between flow variables, assess the accuracy of the numerical solutions, and evaluate the physical consistency of the model under various conditions.

The flow patterns and pressure distributions produced by the simulation demonstrated expected physical behaviors. At low Reynolds Numbers, the flow exhibited smooth and laminar characteristics, with minimal turbulence and a symmetric pressure distribution around the circular object. As the Reynolds Number increased, the flow patterns transitioned to include visible wake regions and separation points behind the object, consistent with known fluid dynamic phenomena. The qualitative changes in flow and pressure gradients align with theoretical expectations, confirming the model's ability to capture the behavior of fluids across different regimes.

A quantitative analysis was performed to examine the relationships between key parameters such as Reynolds Number, grid resolution, and time step. It was observed that the resolution of the grid (nPoints) significantly impacted the accuracy of the pressure distribution and velocity fields. For example, increasing nPoints from 50 to 100 while appropriately reducing the time step (dt) to maintain stability resulted in smoother gradients and a more defined flow structure. This indicates a convergence of numerical solutions as grid resolution improves, aligning with expectations for finite difference methods.

The relationship between Reynolds Number and flow characteristics was also quantified. For example, at higher Reynolds Numbers, the simulation captured the expected formation of vortices wake behind the object. It is difficult to give a numerical accuracy of the Reynolds Numbers because the wind tunnel environment makes the transitional points unique.

To estimate the numerical accuracy of the simulation it is useful to assess the limitations of the simulation. The first limitation is the simulation fails to visualize turbulent flow.

To simulate turbulent flow, the resolution of the resolution must be much greater, the time step much smaller, and the number of iterations much greater making the simulation too computationally complex. Another limitation in the accuracy of the simulation is balancing the resolution, time step, number of iterations, and computational complexity. If the resolution is too low, the simulation will lose visual and numerical precision making the simulation inaccurate. When you increase the resolution, it is easy to have value overflow errors if the time step is not reduced causing the simulation to fail, thus the time step must decrease. When the time decreases too low, the fluid will not fully propagate across the wind tunnel so the iterations must increase, requiring more computing time. The simulation's accuracy is greatly dependent on the initial conditions but does converge as was shown by testing increased resolution. The time complexity of increasing resolution grows quadratically (n^2), while iterations scale linearly (n), making high-resolution simulations computationally expensive.

Conclusion:

The wind tunnel simulation successfully modeled the steady-state flow of an incompressible fluid around a circular obstacle using the Navier-Stokes equations and Chorin's Projection Method. The results demonstrated the capability of the simulation to produce accurate visualizations of flow patterns, velocity fields, and pressure distributions for non-turbulent fluid flow. By increasing grid resolution and refining initial conditions, the simulation achieved smoother gradients and more defined object boundaries, highlighting the convergence and accuracy of the numerical methods used. The findings aligned with theoretical expectations and benchmark studies, reinforcing the validity of the computational model.

Key observations include the predictable transition of flow patterns as Reynolds Numbers increased. At lower Reynolds Numbers, laminar flow was observed with symmetrical pressure fields, while higher Reynolds Numbers introduced wake regions and vortex formation. These transitions were consistent with fluid dynamics theory, although the wind tunnel

environment altered transitional thresholds compared to unbounded flow scenarios. These results underline the utility of the simulation in exploring aerodynamic properties and the effects of varying conditions.

This project provided valuable insights into the challenges of computational fluid dynamics (CFD). It demonstrated the importance of balancing resolution, time step, and computational complexity to ensure accurate and stable simulations. It also highlighted the limitations of the model, including its inability to simulate turbulent flow, which would require significantly greater computational resources and more advanced techniques.

Future work could extend this simulation by incorporating turbulence modeling, enabling exploration of higher Reynolds Numbers and complex flow behaviors. Expanding the model to three dimensions would provide more comprehensive insights into fluid dynamics around arbitrary geometries. Additionally, optimizing the code for parallel processing or GPU acceleration could significantly improve computational efficiency, allowing for higher resolution and more detailed simulations. These extensions would further enhance the applicability of the model for advanced aerodynamic studies and real-world engineering problems.

Sources

Ceyron.

“Machine-Learning-and-Simulation/English/Simulation_scripts/Lid_driven_cavity_python_simple.Py at Main · Ceyron/Machine-Learning-and-Simulation.” *GitHub*, https://github.com/Ceyron/machine-learning-and-simulation/blob/main/english/simulation_scripts/lid_driven_cavity_python_simple.py. Accessed 4 Dec. 2024.

Sato, Masami, and Takaya Kobayashi. *A Fundamental Study of the Flow Past a Circular Cylinder Using Abaqus/CFD*.

Solving the Navier-Stokes Equations in Python | CFD in Python | Lid-Driven Cavity. Directed by Machine Learning & Simulation, 2021. *YouTube*, https://www.youtube.com/watch?v=BQLvNLgMTQE.

Tonon, Patricia, et al. “(PDF) A NURBS-Based Finite Element Formulation for Incompressible Fluid Dynamics and Fluid-Structure Interaction with Rigid Bodies.” *ResearchGate*, Oct. 2024. *www.researchgate.net*, https://doi.org/10.1590/1679-78255772.

Weak Form for Navier-Stokes with Chorin’s Projection. Directed by Machine Learning & Simulation, 2022. *YouTube*, https://www.youtube.com/watch?v=JBmS--3L2eQ.