Lift and Drag Calculations with a 2D Finite Difference Navier–Stokes Solver

Jeremiah Avitia,[1,*] Nadim Mourad,[1,†] Parshv Patel,[2,‡] Boris Zubrilin,[1,§] and Orhan Hosten-Mittermaier[1,¶]

[1] *Department of Physics, University of California, Berkeley*
[2] *College of Computing, Data Science, and Society, University of California, Berkeley*

Abstract: In aerospace design, determining the airflow characteristics and force production of each component of an aircraft is vital to evaluating its efficacy and ensuring safety. Specifically for wing design, accurate computational modeling of lift and drag production can save vast amounts of time and resources that would otherwise have gone to experimental testing of many wing designs. We developed a finite difference numerical solver for the Navier–Stokes partial differential equations (PDEs) to model incompressible fluid flow around solid airfoils in two dimensions. We created a framework to flexibly produce smooth airfoils by interpolating through a set of control points defined by continuous numerical parameters. Our primary challenge was the numerical stability of results at higher airspeeds, with computational limits constraining the spatial and temporal step lengths we could use. As a result, we were unable to obtain stable results above 0.1 m/s airspeed. To validate our simulations, we reproduced canonical conclusions such as increased drag production at higher airspeed and from thicker airfoils, as well as zero lift production from symmetric airfoils at 0° angle of attack. Finally, we demonstrate that our approach predicts significant lift production from airfoils defined using our procedure and four-digit NACA series airfoils. Of all the wing designs we tested, NACA airfoil 5315 produced the best lift-to-drag ratio of 1.479 at 0.1 m/s airspeed.

I. INTRODUCTION AND PHYSICS MOTIVATION

Motivation. Understanding the aerodynamic forces acting on aircraft components is fundamental to aerospace engineering and vehicle design. When air flows around an airfoil, the resulting pressure distribution generates two primary forces: lift, which acts perpendicular to the freestream velocity and enables flight, and drag, which acts parallel to the flow and opposes motion. The ratio of lift to drag (L/D) serves as a critical metric for aerodynamic efficiency, directly influencing fuel consumption, range, and overall aircraft performance.

Relevance of Computational Methods. Analytically solving the Navier–Stokes equations that govern fluid dynamics in general cases remains one of the outstanding challenges in mathematical physics—indeed, the existence and smoothness of solutions in three dimensions constitutes one of the Millennium Prize Problems.

Consequently, the primary methods for analyzing airflow are experimental wind tunnel testing and computational fluid dynamics (CFD) simulations based on numerical approximations to the governing equations. While wind tunnel experiments provide ground-truth measurements, they are expensive, time-consuming, and impractical for exploring large design spaces. CFD simulations offer a complementary approach that enables rapid iteration through design parameters at a fraction of the cost, and has therefore gained massive prominence in the aerospace design realm.

Computational aerodynamics serves several critical functions in modern engineering and physics:

1. Design optimization: CFD enables systematic exploration of wing geometries to maximize L/D ratios, reducing the need for expensive prototype fabrication and wind tunnel testing, thus accelerating the design cycle.

2. Understanding flow physics: Numerical simulations reveal detailed velocity and pressure fields that are difficult or impossible to measure experimentally, providing insight into phenomena such as boundary layer separation, vortex shedding, and turbulent transition.

Report roadmap. In this work, we present a two-dimensional finite difference Navier–Stokes solver for computing lift and drag forces on arbitrarily shaped airfoils. Our solver models incompressible viscous airflow—a valid approximation for airspeeds below around Mach 0.3, or about 100 m/s. [1] This regime encompasses the conditions of low-speed aircrafts such as Cessna single-engine planes, making our approach relevant for practical aerodynamic analysis.

Specifically, we:

1. Implemented a velocity projection engine using the finite difference discretization of the Navier–Stokes equations

2. Use Jacobi iteration to resolve the pressure field at each timestep

3. Enforce physically meaningful boundary conditions

* jeremiahcal9@berkeley.edu
† nadim_mourad@berkeley.edu
‡ parshvpatel_0910@berkeley.edu
§ borisbat@berkeley.edu
¶ orhanhm@berkeley.edu

4. Developed a flexible airfoil generation protocol to define diverse airfoil designs from continuous parameters
5. Implemented a dynamic time stepping scheme to enhance numerical stability
6. Calculated forces by assigning pressures to each section of the airfoil
7. Validated our model by reproducing conventional lift and drag results
8. Experimented with airfoil designs for high lift generation

In this report, we first describe the underlying theory behind CFD analysis of airflow. Then the required setup to treat airflow and fluid-solid interactions with the finite difference method is explored. We explain in detail the computational methods and assumptions behind our core physics simulation and force calculations, along with our approach to airfoil definition. We then present results validating our drag and lift predictions before exploring the L/D capabilities of a variety of airfoils. Finally, we discuss the findings, limitations we encountered, and steps for future work.

II. BACKGROUND AND RELATED WORK

Fluid flow in 2D is described by the Navier–Stokes PDEs, which can be expressed as a vector equation (Eq. 1) representing momentum conservation, along with the mass conservation equation for incompressible flow (Eq. 2):

$$\frac{\partial \vec{\mathbf{u}}}{\partial t} + (\vec{\mathbf{u}} \cdot \nabla)\vec{\mathbf{u}} = -\frac{1}{\rho}\nabla \mathbf{p} + \nu \nabla^2 \vec{\mathbf{u}} + \vec{a} \tag{1}$$

$$\nabla \cdot \vec{\mathbf{u}} = 0 \tag{2}$$

where $\vec{\mathbf{u}}(x, y, t)$ is the vector field of the x and y velocity fields $\mathbf{u}$ and $\mathbf{v}$, $\mathbf{p}$ is the pressure field, ρ is the fluid density (constant for incompressible flow), ν is the dynamic viscosity of the fluid, and $\vec{a}$ is any body accelerations $\frac{\vec{F}_b}{\rho}$ on the fluid.

The other equation of note is the pressure force on a surface in 2D:

$$\vec{F}_{\text{net}} = \int_C \mathbf{p} \cdot \vec{n}_{\text{in}}\, ds \tag{3}$$

where $\mathbf{p}$ is the pressure at the surface and $\vec{n}_{\text{in}}$ is the inward unit normal to the surface. We use this to calculate the net force acting on the airfoil.

To numerically approximate the continuous equations, a discretization scheme must be implemented. Because we chose not to use specialized CFD or PDE libraries in Python, we opted for the finite difference method for its lower conceptual barrier and feasibility to implement with just scientific programming. [2] Our choice of approximation for each derivative, specifically the use of a mix of central and upwind-biased stencils, was informed by successful early CFD work by NASA. [3, 4] For our treatment of CFD-specific problems like pressure solving and correct choice of boundary conditions, we consulted various CFD textbooks [1, 5] In formulating our approach to defining airfoil shapes with continuous parameters, we ensured that profiles akin to conventional airfoils were encompassed in the set of possible airfoils.

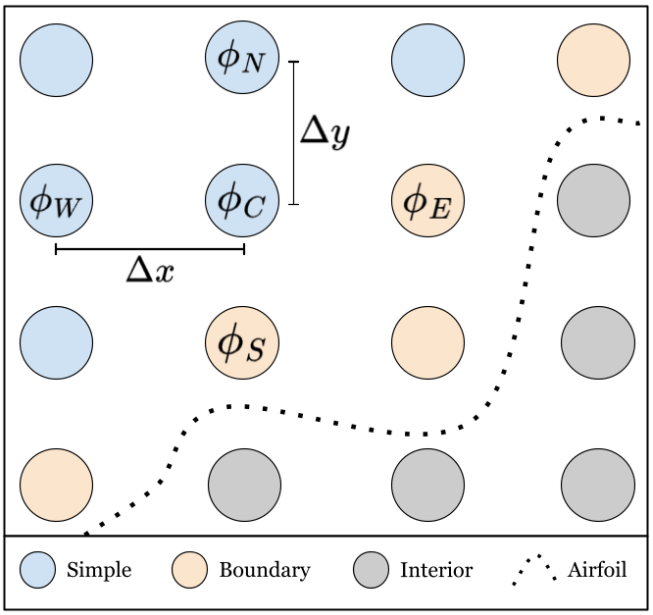

FIG. 1. **System setup and definitions.** Notational conventions for grid spacing and five point stencils are indicated. Classification of fluid points is determined by the airfoil boundary, with boundary fluid points being those with interior points in their five point stencil.

III. SIMULATION SETUP

To implement the finite difference method for the Navier–Stokes equations, we initialize a rectangular grid with uniform spacing (Δx, Δy) in each coordinate direction. Each of the field variables ($\mathbf{u}$, $\mathbf{v}$, $\mathbf{p}$) has a value stored at each grid point, at each timestep. Much of the analysis relies on updating field variable values or other related quantities at each grid point based on the immediately neighboring quantities in each coordinate direction, composing a "five point stencil" of relevant fluid points. For a given field variable ϕ at timestep n, we denote the relevant value at each point in the stencil as follows: ϕ_C^n, (central) ϕ_N^n, (north) etc. (Fig. 1)

We represent a continuous airfoil as a dense set of points forming a closed boundary. Field variable values

at points on on the interior of the wing are neglected since there is no physical interpretation for velocity or pressure within the wing. Furthermore, we distinguish between simple grid points, which have only fluid points in their five point stencil (including at simulation boundary by way of a boundary condition), and boundary points, which have at least one interior point neighbor and therefore require special treatment.

IV. METHODS

Our general procedure to analyze an airfoil is as follows. We alternate between stepping velocities forward, then constructing the pressure field needed to make the velocity field of the following timestep comply with the the mass conservation equation (Eq. 2). This is repeated while increasing airspeed with a constant x-directed body acceleration until the desired average horizontal airspeed $u_{\max}$ is reached. Then, time stepping is stopped and forces on the airfoil are calculated.

Velocity step. Inspecting the momentum equation (Eq. 1), we see that it is composed of a first order time derivative and many spatial derivatives of the velocity and pressure fields. Therefore, by discretizing the time derivative with a forward difference and approximating all spatial derivatives based on the values from the current timestep, we obtain an expression that can be rearranged to find the velocity field in the following timestep:

$$\frac{\vec{\mathbf{u}}^{n+1} - \vec{\mathbf{u}}^{n}}{\Delta t} + (\vec{\mathbf{u}}^{n} \cdot \nabla)\vec{\mathbf{u}}^{n} = -\frac{1}{\rho}\nabla \mathbf{p} + \nu \nabla^2 \vec{\mathbf{u}}^{n} + \vec{a}$$

By approximating first order spatial derivatives with central differences (except for a backward difference approximation for the x-direction derivatives in the convective term $(\vec{\mathbf{u}} \cdot \nabla)\vec{\mathbf{u}}$), and approximating second order derivatives using the second order central difference, the only unknown is $\vec{\mathbf{u}}^{n+1}$. With this, we compute the subsequent velocity field as follows:

$$\vec{\mathbf{u}}^{n+1} = \vec{\mathbf{u}}^{n} - \Delta t \left[(\vec{\mathbf{u}}^{n} \cdot \nabla)\vec{\mathbf{u}}^{n} + \frac{1}{\rho}\nabla \mathbf{p} - \nu \nabla^2 \vec{\mathbf{u}}^{n} - \vec{a} \right]$$

(See Appendix A for the discretized scalar equations)

Listing 1. Python implementation of x-velocity projection

```
u_star = u_C - dt * (
          u_C * (u_C - u_W) / dx
          + v_C * (u_N - u_S) / (2*dy)
          + (p_N - p_S) / (rho*2*dx)
          - nu * (u_E - 2 * u_C + u_W) / (dx**2)
          - nu * (u_N - 2 * u_C + u_S) / (dy**2)
          - ax)
```

Pressure step. The incompressible formulation of the Navier–Stokes equations in 2D have three unknowns ($\mathbf{u}$, $\mathbf{v}$, $\mathbf{p}$) and three equations, meaning the system is closed without an equation of state to relate pressure and density. Motion is governed by the two momentum conservation equations, so pressures must be specified using the additional constraint of the mass conservation equation. To do this, we act as if we are taking another motion step with the new velocities, then take the divergence of the following equation, asserting that $\nabla \cdot \vec{\mathbf{u}}^{*} = 0$ as mass conservation dictates:

$$0 = \nabla \cdot \vec{\mathbf{u}}^{*} = \nabla \cdot \vec{\mathbf{u}}^{n+1} - \nabla \cdot \Delta t \left[\frac{1}{\rho}\nabla \mathbf{p} + (\vec{\mathbf{u}}^{n+1} \cdot \nabla)\vec{\mathbf{u}}^{n+1} - \nu \nabla^2 \vec{\mathbf{u}}^{n+1} - \vec{a} \right]$$

Rearranging,

$$\nabla^2 \mathbf{p} = \rho \nabla \cdot \left[\frac{\vec{\mathbf{u}}^{n+1}}{\Delta t} + (\vec{\mathbf{u}}^{n+1} \cdot \nabla)\vec{\mathbf{u}}^{n+1} - \nu \nabla^2 \vec{\mathbf{u}}^{n+1} - \vec{a} \right]$$

The right hand side of this Poisson equation (denoted RHS) is a scalar field defined at every grid point. We compute the spatial derivatives using our existing approximation methods and take the divergence of the resulting vector field with first order central difference approximations. To find a pressure field that fits the Poisson equation we use Jacobi iteration, passing in the pressure field of the previous time step as an initial guess. Specifically, we discretize the pressure Poisson using second order central differences, rearrange for the central value, and recursively update according to this update rule:

$$\frac{\mathbf{p}_E - 2\mathbf{p}_C + \mathbf{p}_W}{\Delta x^2} + \frac{\mathbf{p}_N - 2\mathbf{p}_C + \mathbf{p}_S}{\Delta y^2} = \mathbf{RHS}$$

$$\mathbf{p}_C = \frac{\Delta y^2(\mathbf{p}_E + \mathbf{p}_W) + \Delta x^2(\mathbf{p}_N + \mathbf{p}_S) - \Delta x^2 \Delta y^2 \cdot \mathbf{RHS}}{2(\Delta x^2 + \Delta y^2)}$$

This method slowly "relaxes" the pressure field toward a solution to the Poisson equation. We consider the process to have converged when the maximum change of any value between iterations is either below a fraction based on the current scale of pressures, or below an absolute tolerance. To prevent code from stagnating, we impose a maximum number of iterations.

Listing 2. Python implementation of Jacobi iteration

```
it = 0
while it < max_iters:
    # Jacobi step using matrix representation
    p_new = jacobi_matrix @ p - (diff_prod / (2 *
        diff_sum) * RHS)

    max_delta = np.max(np.abs(p_new - p_n))
    p = p_new
    it += 1

    pressure_scale = np.percentile(p_new, 95) - np.
        percentile(p_new, 5)
    if max_delta <= tol or max_delta <= rel_tol *
        pressure_scale: # Convergence check
       break
```

Boundary conditions. A handful of special assumptions are needed to treat the velocities and pressures of fluid points on the simulation border and bordering the wing, where the five point stencil is incomplete. We utilize periodic boundary conditions for both velocities and pressures at the edge of the simulation, meaning that the south edge of the simulation is considered to be the spatial neighbor to the north edge, as with east and west. This approximates an open system that continues indefinitely, as long as we take care to make the simulation large enough that flow disruptions from the airfoil don't substantially affect the boundaries. For boundary points bordering the airfoil, we implement the "no-slip" condition by setting u and v to 0 after every velocity update step. This is the accepted boundary condition for solids and approximates physical reality, in which viscous fluids are not able to move with respect to the solid at the interface. [1, 2] The only remaining challenge is pressure derivatives along the airfoil. We resolve this with the common homogeneous Neumann boundary condition, which dictates that airfoil-facing pressure derivatives are 0. This is achieved by setting the central pressure value as a "ghost pressure" at each undefined neighboring pressure value in the airfoil interior. With these boundary conditions in place, fluid flow physics are fully resolved.

Dynamic time stepping. The most prominent challenge we faced was the numerical stability of our results, which is a common issue for Navier–Stokes analysis. Specifically, the finite difference method does not incorporate strict local mass/energy/momentum conservation the same way that more advanced methods like the finite volume method do, making simulations error prone. (see Discussion: Limitations) One source of instability/non-convergence of Navier–Stokes numerical analysis is when time steps are long enough that fluid velocity substantially effects the fluid two grid points away, either by directly carrying over in a physical sense, or via viscous effects on the surrounding fluid. This poses an issue, since our analysis assumes that grid points only affect their immediate neighbors. We enforce the Courant-Friedrichs-Lewy (CFL) condition, which dictates that a safe time step length is

$$\Delta t = C \cdot \min\left(\frac{\Delta x}{\mathrm{v}_{\max}}, \frac{\Delta x^2}{\nu}\right)$$

where, $\mathrm{v}_{\max}$ is the maximum airspeed, and $C \in (0, 1)$ is a safety constant. Before each velocity projection step, we update Δt according to this condition, since it provides a balance of flexibility and numerical safety. We found $C = 0.3$ to work well for stability and simulation speed.

Airfoil definition. Our airfoil definition approach was heavily influenced by our initial goal of implementing a gradient descent scheme to optimize airfoils for L/D. Namely, we define airfoils by interpolating through a set of control points, which are themselves specified by continuous numerical parameters. Parameters can be any real number but are scaled to the range $[0, 1]$ via a sigmoid. The resulting values represent distances from the main chord of control points that are spaced equally along the airfoil length. We interpolate through these control points with a continuous interpolation method to generate a smooth airfoil boundary that can be easily and systematically varied to produce vastly different shapes.

In our code, we allow for either cubic spline or Pchip interpolation. With careful human oversight, cubic spline interpolation can generate very convincing airfoils. However, given a less ideal set of parameters, cubic spline interpolation overshoots and oscillates, producing very non-physical designs. Therefore, we primarily utilize the more stable Pchip interpolation method, which preserves monotonicity and therefore prevents oscillation not present in the control points.

Additionally, we experimented with established four-digit NACA airfoil designs, which were created by passing 100-200 known height parameters to our airfoil class, bypassing the sigmoid parameter scaling, and interpolating with Pchip.

Force calculation. To calculate forces on the airfoil, we must assign a pressure to each section of the airfoil. To do this, we compute the nearest fluid point to each point on the wing boundary. Using a Voronoi diagram approach, we group airfoil points sharing a nearest fluid neighbor into wing sections and assign the fluid point pressure to the airfoil section. Since we necessarily use a dense grid for stable results, the airfoil is split into approximately 100-1000 sections, justifying the approximation that each section is linear. Therefore, the pressure force (Eq. 3) on each linear-approximated section simplifies to

$$\vec{F}_i = \mathrm{p}_{\mathrm{nn}} \Delta s \cdot \vec{n}_{\mathrm{in}}$$

where p_{nn} is the pressure at the nearest neighbor fluid point, Δs is the length of the airfoil section, and $\vec{n}_{\mathrm{in}}$ is the unit inward normal. By summing each of these pressure forces we get the net force vector, from which drag and lift are taken directly as the x and y components of the vector.

Hyperparameters. The fluid characteristics were specified using the NASA standard atmospheric model values for an altitude of $\sim 3000\,\mathrm{m}$ to correspond with our initial goal of modeling low-speed airplanes at cruising altitude ($3000\,\mathrm{m}$). [6] Namely, we use $\rho = 0.90925\,\mathrm{kg/m^3}$ and $\nu = 1.8630 \cdot 10^{-5}\,\mathrm{m^2/s}$. Physical simulation size was set to $2 \times 1\,\mathrm{m}$, chord length was set to $0.7\,\mathrm{m}$, and max cross section (thickness) between 0.1 and $0.2\,\mathrm{m}$ to correspond with the typical scale of values for small planes.

Grid step was chosen to balance runtime and stability, and we settled on an 800×400 grid as the most dense grid we could feasibly run many simulations on. (see Discussion: Limitations) Relative tolerance for pressure iteration was defined as $(\mathrm{p}_{95} - \mathrm{p}_5) \cdot 10^{-3}\,\mathrm{Pa}$, where p_n is

the nth percentile pressure. Similarly, we set the absolute tolerance at 10^{-7} Pa based on evidence that smaller relative and/or absolute tolerances provided little reduction in mean absolute divergence at the cost of longer runtime. Pressure iterations rarely exceed 100 per timestep, so we set the maximum iteration number at 1000.

Airfoil resolution was chosen at 10000 points to be conservative with smoothly defining the boundary, since runtime is not significantly impacted.

V. RESULTS

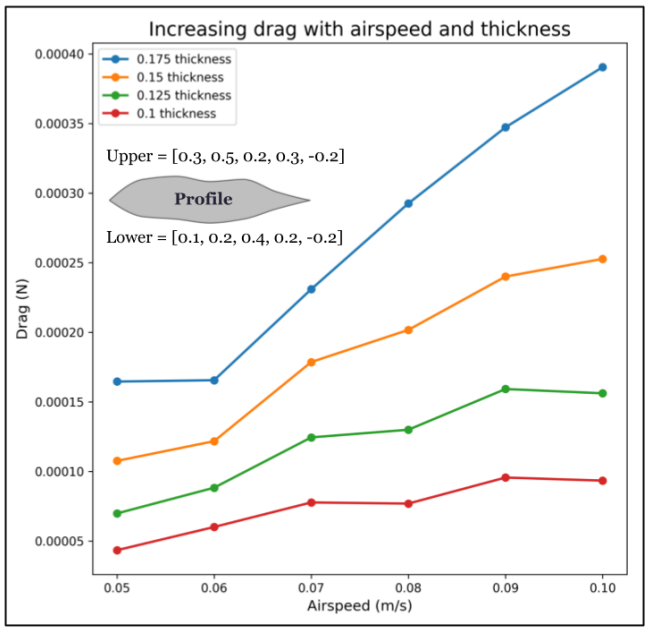

FIG. 2. **Drag validation.** Drag values of representative random airfoil profile at thickness ranging from 0.1 m to 0.175 m and airspeeds from 0.05 to 0.1 m/s. Speed ramped to 0.1 m/s using standard procedure and drag values were taken when airspeed exceeded each threshold value. Airfoil generated using shown upper and lower parameters with Pchip interpolation.

A. Reproducing expected results

First, we demonstrate that our simulation technique reproduces simple expected results as an initial validation step before making other physical conclusions. Namely, we establish the robustness of our drag calculations by examining the effects of airfoil cross section and airspeed on drag. In both cases, classic aerodynamic wisdom affirms that drag should increase (relatively linearly, in fact, for small changes) with either parameter. Probing a representative airfoil profile at various thicknesses and airspeeds, our simulation reproduces the expected results: drag increases with both thickness and airspeed. Especially with varying airspeed for any given thickness, the trend is visually linear. (Fig. 2)

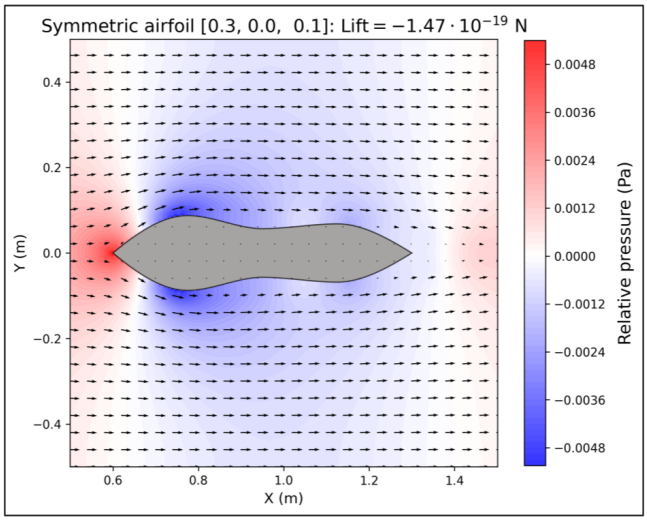

FIG. 3. **Lift validation.** Symmetric airfoil was generated with [0.3, 0.0, 0.1] upper and lower parameters, 0.175m thickness, Pchip interpolation. Lift was $-1.47 \cdot 10^{-19}$ N at 0.1m/s airspeed.

There are fewer obvious tests of the same nature for validating lift. However, symmetric airfoils placed at 0° angle of attack (as all our airfoils are) should have completely symmetric airflow patterns and therefore generate no lift. In practice, our simulation confirms this result, with symmetric airfoils generating negligible lift. A representative symmetric airfoil generates $-1.47 \cdot 10^{-19}$ N of lift at 0.1m/s airspeed, for example. (Fig. 3) Visually, the velocity and pressure fields are symmetric about the main chord, as expected.

These results support the assertion that our drag and lift calculations have physical relevance and are internally consistent, since our simulations produce the expected behavior of force magnitudes in cases where the expected result is clear.

B. Lift generation

We demonstrate that our model represents the process of lift generation by asymmetric airfoils. First, we tested our own airfoil definition protocol by choosing height parameters that produced a somewhat classic airfoil design with a longer top edge and teardrop shape. At 0.1 m/s, we observe this airfoil producing $4.483 \cdot 10^{-4}$ N of lift for a L/D of 1.225. Furthermore, a visual interpretation of "how" the wing generates lift corresponds with conventional wisdom: the longer, more curved upper boundary generates a low pressure zone that is needed to cause the airflow to wrap around the airfoil. (Fig. 4)

These results confirm that our simulation predicts lift

production in a manner analogous to what we would expect from a physical airfoil. Furthermore, we demonstrate that our own method of airfoil definition can be used to design airfoils with substantial lift production.

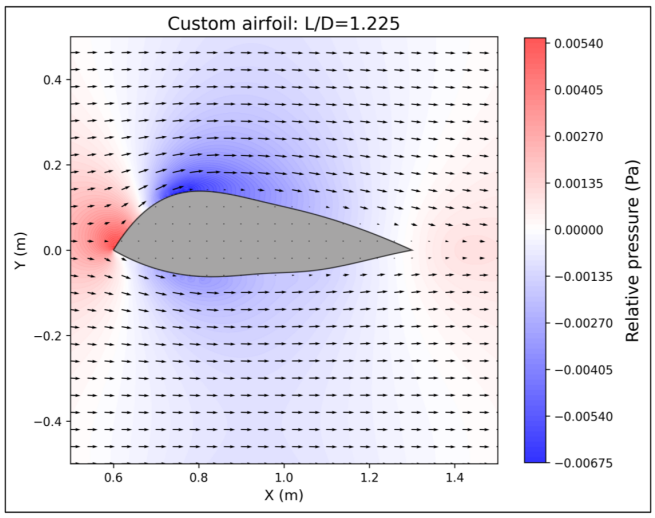

FIG. 4. **Interpolated wings produce lift.** State plot of velocity and pressure fields of an airfoil generated with our interpolation protocol at 0.1 m/s airspeed. Design is human-tweaked to upper ([0.3, 0.5, 0.25, 0.05, -0.2]) and lower ([-0.2, -0.1, -0.15, -0.2, -0.4]) parameters that define an airfoil that generates substantial lift. Interpolation is cubic, thickness is 0.2m. A stride of 16 is used for velocity field, meaning many velocities are not displayed. L/D ratio is 1.225 ($L = 4.483 \cdot 10^{-4}$ N, $D = 3.660 \cdot 10^{-4}$ N).

Based on the time scale of this project and concerns about runtime, we were not able to implement a control parameter optimization scheme like we had initially intended. In light of this, we instead tested 98 four-digit NACA airfoils, which have well-established strong lift production capabilities. From this, the best results came from NACA 5315, which produced a L/D of 1.497. (Fig. 5) NACA 4310 and NACA 6415 also had large L/D ratios of 1.464 and 1.462, respectively. Most airfoils with enough asymmetry displayed decent lift characteristics, with the average lift production being $3.200 \cdot 10^{-4}$ N and the average L/D being 0.720. Thus, our work replicates the known broad lift production capabilities of NACA airfoils.

VI. DISCUSSION

Validation and metrics. As described in Results, our primary and strongest form of validation was initial testing to confirm the legitimacy of our drag and lift calculations. Beyond this, we monitored the mean absolute divergence of the velocity field, which should be near zero according to the mass conservation equation (Eq. 2) and our method of enforcing it by correctively resolving pressures. In simulations without a numerical

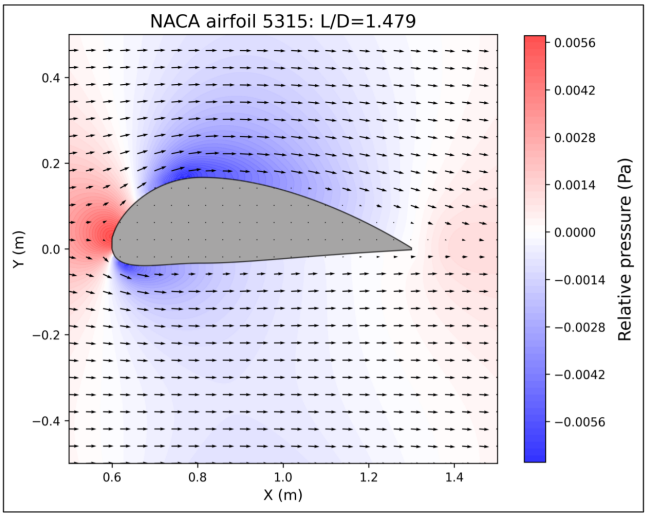

FIG. 5. **NACA L/D search.** State plot of velocity and pressure fields of NACA 5315 at 0.1 m/s airspeed. A stride of 16 is used for velocity field, meaning many velocities are not displayed. Of 98 tested NACA airfoils, NACA 5315 displayed the best L/D of 1.479 ($L = 7.836 \cdot 10^{-4}$ N, $D = 5.298 \cdot 10^{-4}$ N).

blowup, mean absolute divergence across the velocity field stabilized around $1.5 \cdot 10^{-3}$ during the speed ramping process, which we deem acceptable. Lowering the tolerances for Jacobi iteration provided marginal benefit at the cost of considerably more iterations, indicating we are near the limit of what is possible with this technique and our computational power.

Uncertainty. The uncertainty/error in our results is very hard to quantify given the lack of ground truth for our variable and non-physical 2D airfoils. However, our procedure is internally consistent in the sense that simulations preserve the relative ordering of lift and drag in cases where the expected result is obvious (ie. thicker wings of the same profile). Thus, substantial differences between the force production or L/D of two airfoils is likely to suggest an underlying physical result. However, we do not have enough evidence that our simulations are accurate enough to preserve the true ordering of airfoils with minor differences in characteristics (for example, in distinguishing the highest L/D of the NACA airfoils we tested).

Ablations. Force calculations, airflow patterns, and velocity field divergence are consistent with repeated testing of the same initial conditions and ramping procedure, indicating that rounding error buildup is not a major issue. However, varying the grid size does produce different predictions for physical results like pressure forces and airflow. This is likely because the grid spacing used affects the physical implications of the no-slip condition: sparser grids enforce zero velocity farther away from the wing than denser grids. To

fully resolve this, we would hope to have experimental data on the length scales that the no-slip condition is valid, but this is likely very fluid- and material-specific and therefore difficult to test. In the absence of this, after determining that an 800×400 grid was optimal for our purposes, we kept grid spacing constant for all comparative testing and results.

Limitations. The biggest limitation we faced was the numerical stability of our results. We struggled to produce stable results at airspeeds above 0.1 m/s, and numerical blowups sometimes occurred under this speed. We speculate that this is because the finite difference method operates on the assumption that the velocity and pressure fields vary linearly between grid points in a given timestep and for the same grid point from timestep to timestep. Our dynamic time stepping regime does a good job of choosing time steps that uphold the temporal assumption. Thus, our issues likely arise from the spatial linearity assumption.

As airspeed increases, the length scale for changes in field variables flow decreases, meaning a smaller grid size is needed for the linearity assumption to produce stable results. Simulation time increases with roughly $O(n^3)$ since halving grid spacing demands four times as many grid points and causes timesteps to be half as long to enforce the CFL condition. While we can comfortably run 800 × 400 grids, 3200×1600 grids already become prohibitively computationally intense. Unfortunately, this means that our capability to model higher airspeeds is effectively capped with our current set of methods and computational resources.

VII. CONCLUSIONS AND FUTURE WORK

In summary, we present a finite difference Navier–Stokes solver to model airflow around solid 2D airfoils. We also developed a method for flexibly defining airfoils based on a set of continuous numerical parameters. Although we were only able to simulate airspeeds up to ~ 0.1 m/s, our model nevertheless predicts non-negligible lift production in a physically intuitive manner from airfoil designs created with our protocol and four-digit NACA series airfoils.

Whether by increases in the computational power at our disposal, more advanced and stable computational methods for the Navier–Stokes equations, or both, the most obvious area for future work is increasing the airspeeds we can stably simulate to 50+ m/s for physical relevance to airplane flight. Otherwise, we hope to find better validation mechanisms for the numerical lift and drag values we predict, not just the relative ordering of drags/lifts we present. With these improvements in place, we could proceed to our initial goal of airfoil design using ML–based L/D optimization to assist with rapid aerospace design.

VIII. REPRODUCIBILITY AND CODE AVAILABILITY

Code runs on a Python 3.13 environment with recent versions of numpy, scipy, matplotlib, and pandas installed. All code is available at https://github.com/OrhanHM/CFD_sims. Consult the repository Readme for details on how to use the code and reproduce our results.

APPENDIX A

Here are the two scalar momentum conservation equations for 2D incompressible flow:

$$\frac{\partial u}{\partial t} + u\frac{\partial u}{\partial x} + v\frac{\partial u}{\partial y} = -\frac{1}{\rho}\frac{\partial p}{\partial x} + \nu\left(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2}\right) + \vec{a}_x$$

$$\frac{\partial v}{\partial t} + u\frac{\partial v}{\partial x} + v\frac{\partial v}{\partial y} = -\frac{1}{\rho}\frac{\partial p}{\partial y} + \nu\left(\frac{\partial^2 v}{\partial x^2} + \frac{\partial^2 v}{\partial y^2}\right) + \vec{a}_y$$

Our derivative approximation scheme uses a forward difference for the time derivative $\frac{\partial \phi}{\partial t}$, a backward difference for the x-direction convective term $u\frac{\partial \phi}{\partial x}$, and first and second order central differences for everything else. Thus, our equations become:

$$\frac{u_C^{n+1} - u_C^n}{\Delta t} + u_C^n \cdot \frac{u_C^n - u_W^n}{\Delta x} + v_C^n \cdot \frac{u_N^n - u_S^n}{2\Delta y} = -\frac{1}{\rho}\frac{p_E - p_W}{2\Delta x} + \nu\left(\frac{u_E^n - 2u_C^n + u_W^n}{\Delta x^2} + \frac{u_N^n - 2u_C^n + u_S^n}{\Delta y^2}\right) + \vec{a}_x$$

and

$$\frac{v_C^{n+1} - v_C^n}{\Delta t} + u_C^n \cdot \frac{v_C^n - v_W^n}{\Delta x} + v_C^n \cdot \frac{v_N^n - v_S^n}{2\Delta y} = -\frac{1}{\rho}\frac{p_N - p_S}{2\Delta y} + \nu\left(\frac{v_E^n - 2v_C^n + v_W^n}{\Delta x^2} + \frac{v_N^n - 2v_C^n + v_S^n}{\Delta y^2}\right) + \vec{a}_y$$

which are trivially rearranged for u_C^{n+1} and v_C^{n+1}. All other computations shown in continuous derivative forms throughout the report are approximated with the same scheme.

AUTHOR CONTRIBUTIONS

NM and OHM led initial research on computational techniques for Navier–Stokes numerical analysis. OHM,

NM, and BZ developed the finite difference setup and velocity stepping engine. BZ and OHM developed the iterative pressure solving code and experimented with its accuracy. PP and OHM developed our novel airfoil definition protocol. JA and NM developed the visualization/plotting code. JA led presentation slide design. OHM performed the analysis to generate our final results and figures. OHM and PP spearheaded the report writing process.

USE OF AI TOOLS (DISCLOSURE)

ChatGPT in various instances to gain an overview of the computational methods available for certain aspects of the project. The methods we chose to use were only implemented after confirming their validity with non-AI sources. The use of generative AI to write code was limited to the NACA airfoil generation function. Code was inspected to ensure airfoil curves were generated properly, and results were confirmed by comparing with online NACA generation tools. We did not use generative AI for the report-writing process.

ACKNOWLEDGMENTS

We would like to extend a thank you to Harper Sewalls for his guidance on the types of results that would accurately portray our work.

[1] J. H. Ferziger, M. Perić, and R. L. Street, *Computational Methods for Fluid Dynamics* (Springer Nature, Gewerbestrasse 11, 6330 Cham, Switzerland, 2020).
[2] L. Barba and G. Forsyth, Journal of Open Source Education **2**, 21 (2019).
[3] S. E. Rogers and D. Kwak, Applied Numerical Mathematics **8**, 43 (1991).
[4] Q. Kong, T. Siauw, and A. M. Bayen, in *Python Programming and Numerical Methods*, edited by Q. Kong, T. Siauw, and A. M. Bayen (Academic Press, 2021) pp. 337–351.
[5] T. J. Chung, *Computational Fluid Dynamics* (Cambridge University Press, The Edinburgh Building, Cambridge CB2 2RU, UK, 2002).
[6] *U.S. Standard Atmosphere, 1976*, Tech. Rep. (National Oceanic and Atmospheric Administration, 1976).