A Direct Numerical Simulation of a Computational Fluid Dynamics Application Using the Navier-Stokes Equations and Finite Differences

Carl Bernsten
Carroll College, Helena, MT

Follow this and additional works at: https://scholars.carroll.edu/mathengcompsci_theses

Part of the Dynamic Systems Commons, and the Hydraulic Engineering Commons

Recommended Citation
https://scholars.carroll.edu/mathengcompsci_theses/52

This Thesis is brought to you for free and open access by the Mathematics, Engineering and Computer Science at Carroll Scholars. It has been accepted for inclusion in Mathematics, Engineering and Computer Science Undergraduate Theses by an authorized administrator of Carroll Scholars. For more information, please contact tkratz@carroll.edu.
A Direct Numerical Simulation of a Computational Fluid Dynamics Application Using the Navier-Stokes Equations and Finite Differences

By
Carl Berntsen
April 14, 2009
This thesis for honors recognition has been approved for the
Department of Math, Engineering, and Computer Science.

[Signatures and dates]
Director
Date
Reader
Date
Reader
Date
Abstract

When engineers design objects that interact with fluids, they must spend a considerable amount of time understanding the relationship between the fluid and the object. One method is to perform scale tests that involve a complicated apparatus to simulate the fluid flow and provide a way of obtaining measurements. This procedure is expensive and time consuming, sometimes costing thousands of dollars an hour and taking many weeks. As a result, engineers use Computational Fluid Dynamics (CFD) so that an object and a fluid flow can be simulated, thus minimizing time and cost. I will explore a method used to make a two dimensional CFD program using the Navier-Stokes equations for fluid flow, and the Finite Differences Method for CFD. I will then use my program to demonstrate how an airfoil and air interact to produce lift and drag.
# Table of Contents

- List of Figures ................................................................. iii
- Introduction ........................................................................ 4
- Assumptions of Incompressibility ....................................... 4
- Grid Structure .................................................................... 5
- Object Maker ..................................................................... 7
- Object Boundary Conditions ............................................ 9
- Navier-Stokes Equations .................................................. 10
- Discretization ................................................................... 12
- Density and Pressure ........................................................ 13
- Boundary Conditions ....................................................... 16
- Constants .......................................................................... 17
- Results ............................................................................... 17
- Conclusion ......................................................................... 23
- Actual Values ...................................................................... 24
- Results ............................................................................... 25
- Conclusion ......................................................................... 27
- Bibliography ....................................................................... 28
- Appendix - Code - NS2thesis.m ........................................... 29
- Appendix - Code - Objectmaker.m ....................................... 34
List of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Figure 1</td>
<td>The Sections of the Flow Field</td>
<td>7</td>
</tr>
<tr>
<td>Figure 2</td>
<td>An Airfoil made of two circles and a trapezoid</td>
<td>8</td>
</tr>
<tr>
<td>Figure 3</td>
<td>Output for a 10x10, 20x20, 40x40, 80x80 and 160x160 finite element field</td>
<td>9</td>
</tr>
<tr>
<td>Figure 4</td>
<td>A control volume and the velocity vector</td>
<td>13</td>
</tr>
<tr>
<td>Figure 5</td>
<td>Progression of Speed with No-Slip Boundary Conditions along the North and South Edges</td>
<td>18</td>
</tr>
<tr>
<td>Figure 6</td>
<td>Y-Velocity, Pressure, and X-Velocity of the No-Slip Boundary acting on the Airfoil</td>
<td>18</td>
</tr>
<tr>
<td>Figure 7</td>
<td>Initial Condition, Wave Propagating, and Steady State Solution of the Super Slick Boundaries w/o Airfoil</td>
<td>19</td>
</tr>
<tr>
<td>Figure 8</td>
<td>Y-Velocity, Pressure, and X-Velocity of the Super-Slick Boundary acting on the Airfoil</td>
<td>20</td>
</tr>
<tr>
<td>Figure 9</td>
<td>Initial Condition, Wave Propagating, and Steady State Solution of the Set Boundaries w/o Airfoil</td>
<td>20</td>
</tr>
<tr>
<td>Figure 10</td>
<td>Y-Velocity, Pressure, and X-Velocity of the Set Boundary Acting on the Airfoil</td>
<td>21</td>
</tr>
<tr>
<td>Figure 11</td>
<td>Angle of Attack Initial Condition, Pressure of Steady State, and Velocity of Steady State</td>
<td>22</td>
</tr>
<tr>
<td>Figure 12</td>
<td>Y-Velocity, Pressure, and X-Velocity of the Initial Value Simulation with Set West Edge</td>
<td>22</td>
</tr>
<tr>
<td>Figure 13</td>
<td>Y-Velocity, Pressure, and X-Velocity of the Initial Value Simulation with Set West, North, and South Edge</td>
<td>23</td>
</tr>
<tr>
<td>Figure 14</td>
<td>Initial Condition of Simulation with Real Values, Pressure Propagation, and Velocity Propagation</td>
<td>26</td>
</tr>
<tr>
<td>Figure 15</td>
<td>Pressure, Y-Velocity, and Velocity of Simulation with Real Values</td>
<td>26</td>
</tr>
</tbody>
</table>
The goal of Computation Fluid Dynamics (CFD) software is to use the capabilities of the desktop computer to create virtual fluid testing areas to obtain meaningful results that replace costly and time consuming tests conducted in the field. These CFD applications are based on many different methods. The method we will use is referred to as the finite difference method which originates from the oldest technique of evaluating partial differential equations introduced by Euler in the 18th century. Throughout the paper we will gain insight as to how CFD programs based on the finite difference method work, and will compare the results with our expectations purely on a qualitative level.

Assumptions of Incompressibility

The computational resources at hand are responsible for many of the simplifications that are made throughout the simulation. The first is that the CFD will only be in two dimensions. Also, the fluid that is modeled is incompressible. This is because compressible fluids require very small spatial and time steps. If the method described throughout the paper were to simulate a compressible fluid, it would need $\Delta x$ and $\Delta y$ values near millimeters and the time step would have to be in the nanoseconds. The reason behind this is that the specific heats of constant volume and pressure, density, viscosity, velocity, and pressure allow the fluid to be very complicated even on a small scale. Thus, to model a two-by-two-meter square for a second would take $2 \times 10^{15}$ calculations for stored values, many more for all calculations made throughout the process. This would take a long time.

As it stands, an incompressible fluid can be modeled with a $\Delta x$ and a $\Delta y$ value of 2.5 cm and a time step of 50 $\mu$s. Thus, for a two-by-two-meter square, one second would take nearly fifteen million times as few calculations as the one above, meaning much less calculation time. Unfortunately, the equations and understanding required to model an incompressible fluid are
not much easier. This is because the term "incompressible" is false in itself. Every fluid compresses; it is just that "incompressible" fluids compress a very small amount. To model an incompressible fluid, we must look at the equations for a compressible fluid and make assumptions about the behavior of the fluid to simplify the equations.

There are many properties of fluids that change as they interact with objects. Velocity, pressure, density, temperature, and viscosity, to name a few, will all change. Velocity change is very evident to casual observers even as they watch water flow around rocks in a stream. However, it is not easily discernable that where the water collides with a stone, there is a slight increase in density as the fluid compresses slightly at the impact. This compression not only corresponds to a change in pressure but also to a change in temperature. With this change in temperature, there is also a change in viscosity. For the simulation we can make assumptions to limit the number of these quantities that will change.

The fact that it is an incompressible fluid means that the change in density is negligible. Another simplification stems from the fact that most incompressible fluids are good insulators. This allows us to assume that any compression or decompression is an adiabatic process. This means that the temperature and viscosity do not change. Therefore, the quantities we will calculate for the simulation are the x-velocity, y-velocity, and pressure.

Grid Structure

The way the program calculates these quantities is through the difference method and Euler's method of approximation. This means that the test area will be in the form of a square grid that represents a two-meter-by-two-meter area. The individual cells within the grid have the dimensions of $\Delta x$ and $\Delta y$. The values for $\Delta x$ and $\Delta y$ will not be specified at this time because they will change as the number of elements within the field is changed. Because the
grid is square, any object within the grid must be square or made up of smaller squares. This explains why the airfoil shape will be made up of small squares dependant on the size of the finite areas that make up the grid. The process for creating the airfoil will be further explained in the Object Maker section.

The differences are performed by taking values from neighboring cells to obtain the values needed. This process takes differentials and turns them into simple algebraic formulas, for example:

\[
\frac{\partial n_i}{\partial x} = \frac{n_{i+1} - n_{i-1}}{2\Delta x}
\]

where \( n \) represents any variable, and \( i \) represents the location of the cell in the \( x \)-direction. This difference is an example of a centered difference because we look one step forward and one step backward. However, this brings us to our first complication. Suppose we are at the first column of the grid, where \( i = 1 \). The difference above becomes as follows:

\[
\frac{\partial n_1}{\partial x} = \frac{n_2 - n_0}{2\Delta x}
\]

This is a problem because \( n_0 \) does not exist. Thus, along that edge we must use forward differences that are as follows:

\[
\frac{\partial n_i}{\partial x} = \frac{n_{i+1} - n_i}{\Delta x}
\]

and thus

\[
\frac{\partial n_1}{\partial x} = \frac{n_2 - n_1}{\Delta x}
\]

The reverse of this type of difference is a backward difference which will be used when we are at the upper limit of \( i \). Because these differences are less accurate, we will avoid using them whenever possible.
In the simulation $i$ will range between 1 and $lx$, where $lx$ is the number of finite
differences in the x-direction. Likewise, $j$ will represent the location of the cell in the y-direction
and will range between 1 and $ly$, where $ly$ is the number of finite differences in the y-direction.
Similarly to $i$, at the limits of $j$ either forward or backward, differences are necessary for
calculations. Because of this, the test field must be split into nine different sections where the
various differences are used. Figure 1 shows a schematic of the breakdown. These sections

![Diagram of flow field sections](image)

**Figure 1: The Sections of the Flow Field**

will be frequently referred to based on the directions of a compass. We have the West Edge,
Northwest Corner, North Edge, Northeast Corner, East Edge, Southeast Corner, South Edge,
Southwest Corner, and last the central section will be called the main body. Using the finite
difference method and the breakdown of the grid, we can calculate x-velocity, y-velocity, and
pressure for the flow field.

Object Maker

Before putting an object into the flow field, we must understand how the fluid will react
to an object. The fluid will not flow through the object. This may seem obvious, but for the
simulation it is very important. This means that x-velocity, y-velocity, and pressure do not exist within the object and therefore will not be calculated within the object. To achieve this we must clearly show where the object exists or where the fluid exists. We can accomplish this by creating an additional object grid that contains this information. The object grid elements will have the same dimensions as the test field grid elements and, most importantly, wherever a one is in the object grid, the object is in the field grid. The object grid at its simplest is an arrangement of zeros and ones.

To assemble the object grid, we use "if" commands to say whether the spot on the grid is a one or zero. This can be a complicated process unless one uses simple shapes such as boxes, circles, and triangles. Airfoils are not made of these shapes. They are intricate shapes that have many defining characteristics such as chord and camber lengths. The airfoil used in the simulation cannot be this complex. We use two circles and a trapezoid to create the shape we want. We do this by creating a large circle followed by a smaller circle. In between the circles is the trapezoid whose sides will be tangent to the two circles sides. Even though this shape is not an airfoil by the actual definition, it will serve as a simplified airfoil shape for the simulation. Figure 2 shows how the circles combined with the trapezoid form the airfoil shape. The

![Figure 2: An Airfoil made of two circles and a trapezoid](image)
underlying figure is the actual shape produced by the object maker. It is important to note that the shape is not symmetric about the x-axis, a characteristic of an actual airfoil and necessary for creating lift.

The way the object maker function creates these shapes is directly related to the size of the grid. This is because a lower number of grid elements in the field results in the object being represented by a lower number of squares. Figure 3, shows the dependence the shape has on the number of squares. From this perspective, it is apparent that the higher number of grid elements we have, the more accurate the fluid/object interaction will be.

Object Boundary Conditions

The presence of the object complicates the main body calculations. Along the boundary of the object, whenever the simulation attempts to calculate a centered difference, it attempts to look into the object for information. Because of this, we must use forward or backward differences depending on what the cell requires. We accomplished this for the edges and corners of the flow field by writing the differences into the program. This process cannot be used for the object’s boundaries because the shape is dependent on the number of grid elements in the flow field.
The way we can handle this is to assemble the difference equations we need. We can start by recognizing that a centered difference is actually the combination of a forward difference and a backward difference. We can say:

\[
\frac{\partial n_i}{\partial x} = \frac{n_{i+1} - n_{i-1}}{2\Delta x} = \frac{(n_{i+1} - n_i)_f + (n_i - n_{i-1})_b}{\Delta x_f + \Delta x_b}
\]

where the subscript \(f\) or \(b\) represents information provided by the forward or backward difference.

Suppose the fluid cell the simulation is currently calculating is one step west of an object cell. This means that the forward component of the difference is zero and the backward component is not. If the program is calculating a cell to the east of the object, then the opposite is true. Using this process we can assemble all of the first derivative differences we need.

Another important characteristic of the object boundary is that fluid velocity at the surface of an object has been experimentally shown to equal zero (Cline). This is helpful for the assembly notion because in the equations for x-velocity and y-velocity, there are second derivatives which cannot be assembled as straightforwardly as first derivatives can. This also simplifies the simulation in that the only variable calculated at the boundaries is now pressure.

Navier-Stokes Equations

As stated in the introduction, the equations used to update the field values are based on our knowledge of the compressible fluids. The Navier-Stokes equations will provide the backbone to the simulation in that they provide the function for updating x-velocity and y-velocity as the field flow changes. The principle behind the equation is the conservation of momentum. If we imagine a region of fluid with a surface area, the time rate of change in momentum within the region is equal to the rate of momentum flowing into the region and the
forces acting on the surface through shear stresses. The resulting equations in component form are as follows:

\[
\frac{\rho Du}{Dt} = -\frac{\partial P}{\partial x} + \frac{\partial}{\partial x} (\lambda \nabla \cdot v) + \nabla \cdot \left( \mu \frac{\partial v}{\partial x} \right) + v \cdot (\mu \nabla u) + \rho x
\]

\[
\frac{\rho Dv}{Dt} = -\frac{\partial P}{\partial y} + \frac{\partial}{\partial y} (\lambda \nabla \cdot v) + \nabla \cdot \left( \mu \frac{\partial v}{\partial y} \right) + v \cdot (\mu \nabla v) + \rho y
\]

where \( u \) represents the x-velocity, \( v \) represents the y-velocity, \( \lambda \) and \( \mu \) are viscosity, \( P \) is pressure, \( \rho \) is density, \( \rho x \) and \( \rho y \) are outer body accelerations such as gravity, and \( \frac{Du}{Dt} \) is shorthand notation for \( \frac{\partial}{\partial t} + v \cdot \nabla \) (Kuethe & Chow 457). The properties of incompressible fluid greatly simplify this equation. An incompressible fluid is defined to be a fluid where \( \nabla \cdot v = 0 \).

This means that a fluid cannot be expanding outward or contracting inward in every direction at the same time. To further simplify the equation, we will not take the outer body forces into account. Thus, for an incompressible fluid, the Navier-Stokes equations are as follows:

\[
\rho \left( \frac{\partial u}{\partial t} + u \frac{\partial u}{\partial x} + v \frac{\partial u}{\partial y} \right) = -\frac{\partial P}{\partial x} + \mu \left( \frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} \right)
\]

\[
\rho \left( \frac{\partial v}{\partial t} + u \frac{\partial v}{\partial x} + v \frac{\partial v}{\partial y} \right) = -\frac{\partial P}{\partial y} + \mu \left( \frac{\partial^2 v}{\partial x^2} + \frac{\partial^2 v}{\partial y^2} \right)
\]

In this form it is easier to see that the Navier-Stokes equation is basically a force to mass and acceleration relationship (Kuethe & Chow 457). The term involving pressure is taking into account the forces caused by a pressure difference. The \( \mu \nabla^2 v \) component is the force due to the viscosity of the fluid. The actual viscosity of the fluid is a measure of how easily the fluid will flow. Lastly the expanded \( (\nabla \cdot \nabla) v \) is known as the unsteady acceleration of the fluid and accounts for the inertial force.

Because our simulation is a time step simulation, we must solve these equations for the time dependant derivative.
\[
\frac{\partial u}{\partial t} = -\frac{1}{\rho} \frac{\partial P}{\partial x} + \frac{\mu}{\rho} \left( \frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} \right) - \left( \frac{\partial u}{\partial x} + \frac{\partial v}{\partial y} \right)
\]

\[
\frac{\partial v}{\partial t} = -\frac{1}{\rho} \frac{\partial P}{\partial y} + \frac{\mu}{\rho} \left( \frac{\partial^2 v}{\partial x^2} + \frac{\partial^2 v}{\partial y^2} \right) - \left( \frac{\partial v}{\partial x} + \frac{\partial v}{\partial y} \right)
\]

Discretization

For the simulation, we want to find the actual values of \( u \) and \( v \). To do this we will use Euler's Method of Approximation:

\[
f_{i,j,l+1} = f_{i,j,l} + \Delta t \cdot \frac{\partial f_{i,j,l}}{\partial t}
\]

and \( f \) represents both \( u \) and \( v \), and eventually \( P \) as well. Notice that the subscripts \( i, j, \) and \( l \) are used to represent the \( x \)-direction step, \( y \)-direction step, and time step respectively.

Advancing a step requires completion of the discretization of the differentials. Using centered differences the Navier-Stokes equations put into Euler's Method of Approximation are as follows:

\[
u_{i,j,l+1} = u_{i,j,l} + \Delta t \left( -\frac{1}{\rho} \frac{P_{l+1,j,l} - P_{l-1,j,l}}{2\Delta x} \right.
\]

\[
+ \frac{\mu}{\rho} \left( \frac{u_{i+1,j,l} - 2u_{i,j,l} + u_{i-1,j,l}}{\Delta x^2} + \frac{u_{i,j+1,l} - 2u_{i,j,l} + u_{i,j-1,l}}{\Delta y^2} \right)
\]

\[
- \left( \frac{\partial u_{i,j,l}}{\partial x} + \frac{\partial v_{i,j,l}}{\partial y} \right)
\]

The equation for \( v \) is not much different from this. We can see how if \( i = 0 \) or \( i = lx \) there would be numerous spots that would refer to locations at 0 or \( lx + 1 \), which are both beyond the boundaries. The same occurs for the \( y \) direction. This is why we must use special differences at the boundaries.
Density and Pressure

To explain the pressure, density, and velocity relationship for an incompressible fluid, we must, again, examine the relationship for a compressible fluid. The density of a compressible fluid can be derived from the conservation of mass by looking at a control volume. Figure 4 shows the control volume with sides $\Delta x$ and $\Delta y$. The conservation of mass states that

![Figure 4: A control volume and the velocity vector](image)

the change in mass inside the volume must be equal to the amount of mass entering and exiting through the sides of the volume. This means the following:

$$\frac{\partial}{\partial t} m = \Phi_{net}$$

where $\Phi_{net}$ is the total flux through the boundaries. After substituting the values for change in mass and total flux we obtain the following equation:

$$\frac{\partial \rho}{\partial t} \Delta x \Delta y = [(\rho u \Delta y)_e - (\rho u \Delta y)_w] + [(\rho v \Delta x)_s - (\rho v \Delta x)_n]$$

where subscripts $n, s, e,$ and $w$ represent the north, south, east, and west edges of the control volume. By dividing both sides by $\Delta x \Delta y$ we get the following equation:

$$\frac{\partial \rho}{\partial t} = \frac{\rho u_l - \rho u_r}{\Delta x} + \frac{\rho v_b - \rho v_t}{\Delta y} = -\frac{\rho u_r - \rho u_l}{\Delta x} - \frac{\rho v_t - \rho v_b}{\Delta y}$$
As $\Delta x$ and $\Delta y$ approach infinitesimally small size the changes in the numerators also become infinitesimally small resulting in the following:

$$\frac{\partial \rho}{\partial t} = - \frac{\partial (\rho u)}{\partial x} - \frac{\partial (\rho v)}{\partial y} = - \nabla \cdot (\rho \mathbf{v})$$

where $\rho$ is density and $\mathbf{v}$ is the velocity vector (Kuete & Chow 28).

For compressible fluids, the pressure of a substance is related to the density. This relation is complicated by the fact that as a fluid compresses; it also increases in temperature, which also has an effect on the fluid's pressure. To keep things simple, we will not introduce temperature into the simulation. We can do this by making assumptions about heat transfer. For our fluid we will assume it is a good insulator, and therefore does not release heat quickly. The defining relationship for an adiabatic process is as follows:

$$PV^\gamma = \text{const.}$$

where $\gamma$ is the ratio of $C_p$, specific heat constant pressure, over $C_v$, specific heat constant volume. We use this equation to derive the pressure-density relationship we will be using for all of the simulations. The equation is helpful because we can use it to relate the initial values of pressure and volume to the values after the process has occurred. In equation form this is as follows:

$$P_0 V_0^\gamma = P_t V_t^\gamma$$

After some rearranging we get as follows:

$$P_t = P_0 \left( \frac{V_0}{V_t} \right)^\gamma$$

From the ideal gas law, we know the following:
\[ V_n = \frac{nM}{\rho_n} \]

where \( n \) is the number of moles and \( M \) is the molar mass of the fluid. After substituting in this relationship for the volumes we get the following:

\[ P = P_0 \left( \frac{\rho}{\rho_0} \right)^\gamma \]

where \( P_0 \) and \( \rho_0 \) are the standard pressure and density (Kueth & Chow 198). This equation, while derived under the consideration of compressible gases, is the equation we will also use for the incompressible fluid.

The reason we are going over this now is that the density of an incompressible fluid actually does change by a very small amount. Usually this change is negligible, but for our purposes we must take it into account to find the pressure-velocity relationship we need. Because the values of density for the incompressible fluid will remain very close to the initial value, we can use the Taylor series to reduce the above equation to include only the most informative term. The Taylor series of the adiabatic pressure function around the initial pressure is as follows:

\[ P(\rho) = P_0 \left( \frac{\rho_i}{\rho_0} \right)^\gamma + \gamma P_0 \frac{\rho_i^{\gamma-1}}{\rho_0^\gamma} (\rho - \rho_i) + H \]

where \( \rho_i \) is used to represent the initial density and \( H \) represents the high order terms. We will ignore the high order terms because contain only a small amount of information. We will also make the assumption that the initial density is equal to the absolute density of the fluid.

Because we have the time dependant change of density, we will also take the time derivative of the function to get the following:

\[ \frac{\partial P}{\partial t} = \gamma P_0 \frac{\rho_i^{\gamma-1}}{\rho_0^\gamma} \frac{\partial \rho}{\partial t} \]
Now we can substitute our equation for density in and expand the divergence to get the following:

\[
\frac{\partial \rho}{\partial t} = \gamma P_0 \frac{\rho_0^{\gamma - 1}}{\rho_0^\gamma} \left( u \frac{\partial \rho}{\partial x} + v \frac{\partial \rho}{\partial y} + \rho_0 \left( \frac{\partial u}{\partial x} + \frac{\partial v}{\partial y} \right) \right)
\]

We must remember that the density is assumed to never change and because of that \( \rho_i = \rho_0 \).

Thus our equation simplifies down to this compact form:

\[
\frac{\partial P}{\partial t} = -\gamma P \left( \frac{\partial u}{\partial x} + \frac{\partial v}{\partial y} \right)
\]

We now have the equations that we will use to update the fields values for \( u, v, \) and \( P \):

\[
\frac{\partial u}{\partial t} = - \frac{1}{\rho} \frac{\partial P}{\partial x} + \frac{\mu}{\rho} \left( \frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 v}{\partial y^2} \right) - \left( \frac{\partial u}{\partial x} + \frac{\partial v}{\partial y} \right)
\]

\[
\frac{\partial v}{\partial t} = - \frac{1}{\rho} \frac{\partial P}{\partial y} + \frac{\mu}{\rho} \left( \frac{\partial^2 v}{\partial x^2} + \frac{\partial^2 v}{\partial y^2} \right) - \left( \frac{\partial v}{\partial x} + \frac{\partial u}{\partial y} \right)
\]

\[
\frac{\partial P}{\partial t} = -\gamma P \left( \frac{\partial u}{\partial x} + \frac{\partial v}{\partial y} \right)
\]

Boundary Conditions

For the four edges, the north and south edges are the edges that we actually have a choice in what type of boundary we want. There are three basic boundaries: a set boundary is where we set the velocity and calculate the pressure or visa-versa; a super slick boundary occurs when the program calculates the pressure and velocity using the appropriate differences, or a no-slip boundary where the velocity along the boundary is assumed to be zero in the x-direction and y-direction (Cline).
Constants

For the test simulation, arbitrarily selected values will be used in place of the actual. This is in an important step because it allows us to test for bugs and to achieve steady state solutions. The solutions become very qualitative and not quantities. For our purposes of exploration, this is justifiable. The test area is again a two-meter-by-two-meter box with 80 finite elements along each edge. The time step of 50 $\mu s$ has been established by running the simulation at smaller and smaller time steps until the variation between runs is insignificant. The table below shows the values used for the constants that exist within the equations of motion.

<table>
<thead>
<tr>
<th>Value</th>
<th>Test Simulations</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\rho$ - Density</td>
<td>$1E^{-5} \text{ kg/m}^2$</td>
</tr>
<tr>
<td>$\mu$ - Viscosity</td>
<td>$1.8E^{-5} \text{ Pa} \cdot \text{s}$</td>
</tr>
<tr>
<td>$P$ - Pressure</td>
<td>$1 \text{ Pa}$</td>
</tr>
<tr>
<td>$\gamma = C_P/C_V$</td>
<td>$2E^{-5}$</td>
</tr>
</tbody>
</table>

Results

The first test was of an empty field with the north and south edges being no-slip boundaries. The west edge had a constant x-velocity of 40 m/s and y-velocity of 0 m/s. Due to the velocity being specified, the pressure along this edge was calculated using forward differences. The eastern edge was a super-slick boundary that allowed the incoming velocity to "pass" through it and out of the test area. Figure 5 shows graphs of speed as a wave moves
through the test area. As the fluid's velocity increases through the main body, the velocity at the north and south edges remains at zero. This causes the formation of symmetric boundary layers. This is akin to the real life situation of a wind tunnel, where interactions with the walls slows the fluid down except in the middle of the channel.

For the next test, we will insert the airfoil into the flow. We should still see the boundary layers coming off the north and south edges. Because, velocities at the object’s boundaries are to remain zero as well, we should also see boundary layers forming around the object. In addition, we know that there will be a high pressure area in front of the airfoil where
the fluid diverges and a low pressure behind the airfoil where it converges. At these areas we should see activity on our y-velocity graph as the fluid flows up, over, and down, under the airfoil. Figure 6 confirms our estimation as to how the fluid interacts with the airfoil.

Next we will use a super-slick boundary along the north and south edges. For a fluid field without an object inside it, we should see the air pick up speed and eventually reach the steady state solution in which every point has the same value as the western boundary. Figure 7 below shows the initial condition, a point midway through the wave's propagation through the field, and then the steady state solution.

Once we insert the airfoil into the flow, we would hope to see similar results to the no-slip simulation. We would expect the airfoil to form a boundary layer, high pressure at the leading edge, low pressure at the trailing edge, and splitting the y-velocity so it goes around the airfoil. Figure 8 shows that the results are consistent with our expectations.
The last type of boundary we will examine is the set type. This is where the west, north, and south boundaries are set to have the same velocity values. Because the velocities are set, they obviously do not need to be calculated, thus minimizing the calculation time. Figure 9 shows the velocity at the first time step, after the velocity has propagated through the field for a while, and the steady state solution. While minimizing calculation time is always a plus, the tradeoff is that the boundaries must be sufficiently far away from the object to not interfere with the boundary layer that forms around the airfoil. From the results of the super slick boundary test, it looks as if the boundary layer does extend to the north and south boundaries.
but only by a small amount. Figure 10 shows the airfoil's y-velocity, pressure, and x-velocity of the steady state of the set boundaries.

![Figure 10: Y-Velocity, Pressure, and X-Velocity of the Set Boundary Acting on the Airfoil](image)

The results from the set boundary simulation are nearly identical to the results of the super slick simulation. These results point to the program's validity because if the test were actually performed in a lab, we would expect the results to be nearly the same. The only difference is in the boundary layer thickness as predicted. The effect is minor, but we can see that at the northeastern and southeastern corners of the super slick simulation the x-velocity does not equal 40 m/s as it does in the set boundary simulation. The difference is acceptable when we consider the set boundary test reaches the same steady state in nearly 1000 fewer time steps.

The flexibility in the simulation's boundary conditions allows for applications that were not even thought of during the writing stage. Often it is necessary to examine an airfoil at different orientations, or angles of attack. If we set the west and south edge's x and y velocities to 28.3 m/s and allow the north and east edges to be super slick we can simulate the airfoil's having an angle of attack of 45 degree and a speed of 40 m/s. Figure 11 shows the initial
condition, the pressure, and the steady state velocity of the angle of attack simulation.

Another step we can take to decrease the calculation time is to set the initial condition of the field equal to the western edge set condition. This will only work if when we do this we also set the velocity in and around the object to zero. Figure 12 shows the y-velocity, pressure, and x-velocity that result from this simulation. The results of this simulation are identical to the result of the super slick north and south boundary simulation except in half the time. We can do the same for the set boundaries as well. Figure 13 shows these results.
Figure 13: Y-Velocity, Pressure, and X-Velocity of the Initial Value Simulation with Set West, North, and South Edge

This is very useful for when we test the real values because they will significantly decrease the calculation time of the simulations.

Conclusion

Even though we have no way to confirm the validity of the simulation, the results are encouraging. The x-velocity of the fluid behaved as expected by being zero at the border of object and then increasing away from the object. There was high pressure at the leading edge of the airfoil, low pressure in the wake, and there was a higher pressure below the airfoil than above. This is as expected because it is through this pressure difference that airfoils produce lift and drag. The y-velocity shows how the fluid reacted to the pressure difference by going around the fluid. All in all, even though the constants do not correspond to an actual fluid, the basis for the simulation seems very solid and will provide a solid foundation for moving on to the real values.
Actual Values

The Table below shows the values used in Simulation One and the actual values for water that will be used in Simulation Two. Water will be simulated because it is well known as an incompressible fluid.

<table>
<thead>
<tr>
<th>Value</th>
<th>Simulation One</th>
<th>Simulation Two</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \rho ) - Density</td>
<td>( 1E^{-5} \text{ kg/m}^3 )</td>
<td>1000 ( \text{ kg/m}^3 )</td>
</tr>
<tr>
<td>( \mu ) - Viscosity</td>
<td>( 1.8E^{-5} \text{ Pa} \cdot \text{s} )</td>
<td>( 8.94E^{-5} \text{ Pa} \cdot \text{s} )</td>
</tr>
<tr>
<td>( P ) - Pressure</td>
<td>1 ( \text{ Pa} )</td>
<td>101.325 ( \text{ kPa} )</td>
</tr>
<tr>
<td>( \gamma ) - ( C_p/C_v )</td>
<td>( 2E^{-5} )</td>
<td>( 75.327/74.53 = 1.011 )</td>
</tr>
</tbody>
</table>

The effects of the constants are very drastic. They completely change how the equations behave. The equations of motion with the arbitrary values substituted in equal:

\[
\frac{\partial u}{\partial t} = -100000 \frac{\partial P}{\partial x} + 1.8 \left( \frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} \right) - \left( u \frac{\partial u}{\partial x} + v \frac{\partial u}{\partial y} \right)
\]

\[
\frac{\partial v}{\partial t} = -100000 \frac{\partial P}{\partial y} + 1.8 \left( \frac{\partial^2 v}{\partial x^2} + \frac{\partial^2 v}{\partial y^2} \right) - \left( u \frac{\partial v}{\partial x} + v \frac{\partial v}{\partial y} \right)
\]

\[
\frac{\partial P}{\partial t} = -2E^{-5} \left( \frac{\partial u}{\partial x} + \frac{\partial v}{\partial y} \right)
\]

We can see that the pressure plays a significant role in determining the flow, but the change in pressure from one step to the next is going to be small. The equations with the actual values are as follows:

\[
\frac{\partial u}{\partial t} = -0.001 \frac{\partial P}{\partial x} + 8.94E^{-7} \left( \frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} \right) - \left( u \frac{\partial u}{\partial x} + v \frac{\partial u}{\partial y} \right)
\]

\[
\frac{\partial v}{\partial t} = -0.001 \frac{\partial P}{\partial y} + 8.94E^{-7} \left( \frac{\partial^2 v}{\partial x^2} + \frac{\partial^2 v}{\partial y^2} \right) - \left( u \frac{\partial v}{\partial x} + v \frac{\partial v}{\partial y} \right)
\]

\[
\frac{\partial P}{\partial t} = -102440 \left( \frac{\partial u}{\partial x} + \frac{\partial v}{\partial y} \right)
\]
It appears that the last term now is the most dominant. This is true except when the flow has yet to begin. This is because if we have the initial velocities of the flow as 0 m/s, then this term will play a very small role. The dominant term to initiate a value change is the pressure gradient. The pressure equation shows that even small changes in velocity will result in large changes in pressure. In the test run, any changes in pressure were very small but had large consequences on the flow; now any changes in pressure will be large but have smaller effects on the flow.

Results

With the new values in place for water, the simulation ran very differently. The western edge was given a much smaller initial value of 1 m/s. This is because an initial condition of 40 m/s, used previously, next to a cell of 0 m/s, as it would be at the first time step, would generate an increase in pressure of 1639.04Pa in 0.00001s. This kind of spike would eventually cause numerical errors.

Unlike before, when we ran the simulations with a number of different boundaries, we will only concentrate on the boundary conditions that yielded the best results. This is the set boundary along the western edge with the field set at 0 m/s initially and 1 m/s initially. We will test these simulations and compare the results to the simulations that used the test values.

Figure 14 shows the results from the western boundary set at 1 m/s and the initial field velocity set to 0 m/s. As we can see, there is a great deal of variation between these results and
the ones from the test simulations. The wave truly does seem to continue to propagate into the test field, and this manifests itself not only in the $x$-velocity but also in the pressure as well. Another problem is that it does not ever reach a steady state solution like the test simulations. The nail in the coffin for this test is that it took nearly six times as long to complete this simulation when compared to the test simulation.

To examine the airfoil’s effect on the flow, we are going to set the western boundary and the initial condition on the field to have an $x$-velocity of 1 m/s. Ideally, as it did for the test values, this will improve the calculation time of the simulation. Figure 15 shows the results of this test.

![Figure 14: Initial Condition of Simulation with Real Values, Pressure Propagation, and Velocity Propagation](image1)

![Figure 15: Pressure, Y-Velocity, and Velocity of Simulation with Real Values](image2)
The water seems to react more closely to as expected with the field set to an initial value. We see from the pressure that it has indeed formed a high pressure in front similar to how it did in the test simulations. The y-velocity also appears to behave closely to the same way it did. In both cases there seems to be a strange delay however. Before, the highest and lowest values for both the pressure and velocity were right at the object's boundary; here there is a band away from the object where the pressure and velocity magnitudes are higher than at the object's boundary. The velocity graph is interesting in that it shows an entire zone where the velocity is nearly zero.

Conclusion

The simulation with real values yielded results that were not expected. We have a sense of how water would behave if put under these conditions, and our results seemed not to be consistent with this behavior. Because of the success of the test values, it does not seem as if the program is causing the errors. In an attempt to "fix" the problem, smaller time and spatial steps were tried, going as low as 50 times smaller than the steps actually used in the test. This was to check if waves were caused by numerical approximation errors of using too large a step size. The results from these checks turned out to be remarkably similar. Even if the values improved significantly, the calculation time increased to the point of being excessive.

Despite the results, it is easy to see how with a computational fluid dynamics program already in place the next hardest part is preparing the object for testing. This saves considerable time in the design phase of the process. The ability to make changes to the shape of the object and instantly see how the shape affects the performance is amazingly helpful and cost effective. As computers become more and more advanced, these simulations will become more accurate leading to greater efficiency and additional design breakthroughs.
Bibliography


Interview with Dr. Cline, Professor at Carroll College, Helena Montana. Fall Semester of 2008
tic
clear all;

% we're going to set up a 2m x 2m test area. sx is the length of the test
% area in the x-direction, sy is in the y-direction.
sx = 2; %m
sy = 2; %m

% If we wish to use the test values set sim equal to 1, for real values
% sim equals 2.
sim = 2;

% This initializes the fact that a loop has not been completed. Sometimes
% it was necessary to loop through the program again because only 2500 time
% steps can be stored at a time. Multiple loops allow multiples of the
% time steps to be completed.

loop = 0;

% Sets the values of the fluid depending on whether real values or test
% values are to be used.
if(sim == 1)
    lr = 80; % The number of finite elements along each edge.
dt = 0.00005; % Time step
time = 2500; % Total number of time steps.
mu = 1.8e-5; % Viscosity of the fluid
rho = 1e-5; % Density
gam = 2e-5; % Ratio of specific heats
Po = 1; % Initial pressure
lpf = 3; % Number of times the program should loop through

elseif(sim == 2)
    lr = 80;
dt = 0.00002;
time = 2500;
mu = 8.94e-4;
rho = 1000;
gam = 1.011;
Po = 101325;
lpf = 2;
end

lx = lr;
ly = lr;
dx = sx/lx; %Finds the length of the spatial step
dy = sy/ly;
[room choice] = objectmaker(lr,lx,ly); %Sends the dimensions
% of the test area to the objectmaker function and recieves the airfoil
% array.

% Simulation Constraints. Boundary controls if there are walls or not.
% Pressure controls if pressure is calculated.
boundary = 2; %1-walls, 2-no walls
pres = 1; %1-pressure calculate

% Initial Conditions
ul = 1;
u = 1*ones(lx,ly,time);
v = zeros(lx,ly,time);
P = Po*ones(lx,ly,time);
speed = zeros(lx,ly,time+1);

u(1,:) = ul;

% If the field is set to have velocity this will set the velocity at the
% boundaries of the object equal to zero.
for i = 1:lx
for j = 1:ly
    if(room(i,j) == 1)
        u(i-1:i+1,j,:) = 0;
        v(i-1:i+1,j,:) = 0;
        u(i,j-1:j+1,:) = 0;
        v(i,j-1:j+1,:) = 0;
    end
end
end

end

for lp = 1:lpf
    if(loop ~= 0)
        u(:,:,1) = u(:,:,time);
        v(:,:,1) = v(:,:,time);
        P(:,:,1) = P(:,:,time);
    end
end

for l=1:time
    for i = 2:lx-1
        for j = 2:ly-1
            % Allows calculations to be performed wherever the object isn't.
            if(room(i,j)==0)

                % Object Boundary Conditions
                % Assembles the Differences Needed in the Main Body
                if(room(i+1,j) == 0)
                    uw = u(i+1,j,1)-u(i,j,1);
                    dxw = dx;
                else
                    uw = 0;
                    dxw = 0;
                end
                if(room(i-1,j) == 0)
                    ue = u(i,j,1)-u(i-1,j,1);
                    dxe = dx;
                else
                    ue = 0;
                    dxe = 0;
                end
                if(room(i,j+1) == 0)
                    vn = v(i,j+1,1)-v(i,j,1);
                    dyn = dy;
                else
                    vn = 0;
                    dyn = 0;
                end
                if(room(i,j-1) == 0)
                    vs = v(i,j,1)-v(i,j-1,1);
                    dys = dy;
                else
                    vs = 0;
                    dys = 0;
                end
                if(pres == 1)
                    P(i,j,1+1) = P(i,j,1)+dt*...
                    *(u(i,j,1)*(uw+ue)/(dxw+dxe)+(vn+vs)/(dyn+dys)));
                end

            end
        end
    end
end

% Main Body Velocity Equation
if(room(i,j+1) == 1 & room(i,j-1) == 1 & room(i+1,j) == 1 & room(i-1,j) == 1)

    u(i,j,1+1)=u(i,j,1) + dt* ...  
    -u(i,j,1)*(u(i+1,j,1)-u(i-1,j,1))/(2*dx) ...  
    -v(i,j,1)*(u(i,j+1,1)-u(i,j-1,1))/(2*dy) ...  

Berntsen 30
\[-1/\rho^*(P(i+1,j,1) - P(i-1,j,1))/(2*\Delta x) \ldots \]
\[+ \mu/\rho^*(u(i+1,j,1) - 2*u(i,j,1) + u(i-1,j,1))/\Delta x^2 + \ldots \]
\[+ u(i,j+1,1) - 2*u(i,j,1) + u(i,j-1,1))/\Delta y^2) \]

\[v(i,j+1) = v(i,j,1) + dt* \ldots \]
\[-u(i,j,1)^*(v(i+1,j,1) - v(i-1,j,1))/(2*\Delta x) \ldots \]
\[-v(i,j,1)^*(v(i,j+1,1) - v(i,j-1,1))/(2*\Delta y) \ldots \]
\[-1/\rho^*(P(i,j+1,1) - P(i,j-1,1))/(2*\Delta y) \ldots \]
\[+ \mu/\rho^*((v(i+1,j,1) - 2*v(i,j,1) + v(i-1,j,1))/\Delta x^2 + \ldots \]
\[(v(i,j+1,1) - 2*v(i,j,1) + v(i,j-1,1))/\Delta y^2) \]

end
end
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
South Edge Boundary (i<i<lx and j=1)
if(pres == 1)
P(i,1,l+1) = P(i,1,l) + dt* \ldots
-gam*P(i,1,l)^*(u(i+1,1,l) - u(i-1,1,l))/(2*\Delta x) \ldots
+(v(1x,2,l) - v(1x,1,l))/(\Delta y));
end

if(boundary == 2)

u(i,1,l+1) = u(i,1,l) + dt* \ldots
-u(i,1,l)^*(u(i+1,1,l) - u(i-1,1,l))/(2*\Delta x) \ldots
-v(i,1,l)^*(v(i,2,l) - v(i,1,l))/(\Delta y) \ldots
-1/\rho^*(P(i+1,1,l) - P(i-1,1,l))/(2*\Delta x) \ldots
+ \mu/\rho^*((u(i+1,1,l) - 2*u(i,1,l) + u(i-1,1,l))/\Delta x^2 + \ldots
(u(i,3,l) - 2*u(i,2,l) + u(i,1,l))/(\Delta y^2)));

v(i,1,l+1) = v(i,1,l) + dt* \ldots
-u(i,1,l)^*(v(i+1,1,l) - v(i-1,1,l))/(2*\Delta x) \ldots
-v(i,1,l)^*(v(i,2,l) - v(i,1,l))/(\Delta y) \ldots
-1/\rho^*(P(i,2,l) - P(i,1,l))/(\Delta y) \ldots
+ \mu/\rho^*((v(i+1,1,l) - 2*v(i,1,l) + v(i-1,1,l))/\Delta x^2 + \ldots
(v(i,3,l) - 2*v(i,2,l) + v(i,1,l))/(\Delta y^2));

end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
North Edge Boundary (i<i<lx and j=ly)
if(pres == 1)
P(i,ly,1+l) = P(i,ly,1) + dt* \ldots
-gam*P(i,ly,1)^*(u(i+1,ly,1) - u(i-1,ly,1))/(2*\Delta x) \ldots
+(v(ly,2,l) - v(ly,1,l))/(\Delta y));
end

if(boundary == 2)

u(i,ly,1+l) = u(i,ly,1) + dt* \ldots
-u(i,ly,1)^*(u(i+1,ly,1) - u(i-1,ly,1))/(2*\Delta x) \ldots
-v(i,ly,1)^*(v(i,ly,2,l) - v(i,ly,1,l))/(\Delta y) \ldots
-1/\rho^*(P(i+1,ly,1) - P(i-1,ly,1))/(2*\Delta x) \ldots
+ \mu/\rho^*((u(i+1,ly,1) - 2*u(i,ly,1) + u(i-1,ly,1))/\Delta x^2 + \ldots
(u(i,ly,3,l) - 2*u(i,ly,2,l) + u(i,ly,1,l))/(\Delta y^2)));

v(i,ly,1+l) = v(i,ly,1) + dt* \ldots
-u(i,ly,1)^*(v(i+1,ly,1) - v(i-1,ly,1))/(2*\Delta x) \ldots
-v(i,ly,1)^*(v(i,ly,2,l) - v(i,ly,1,l))/(\Delta y) \ldots
-1/\rho^*(P(i,ly,2,l) - P(i,ly,1,l))/(\Delta y) \ldots
+ \mu/\rho^*((v(i+1,ly,1) - 2*v(i,ly,1) + v(i-1,ly,1))/\Delta x^2 + \ldots
(v(i,ly,3,l) - 2*v(i,ly,2,l) + v(i,ly,1,l))/(\Delta y^2));

end

end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
East Edge Boundary (i=lx 1<j<ly) (back x)

Berntsen 31
% For the real values, an error would occur along the east edge so the fix
% was introduced to take the values from the column adjacent to the east
% edge.
if(sim == 1)
  for j=2:ly-1
    if(pres == 1)
      P(lx,j,1+1) = P(lx,j,1)+dt*(
        -gam*P(lx,j,1)*((3*u(lx,j,1)-4*u(lx-1,j,1)+u(lx-2,j,1))/(2*dx))
        +(v(lx,j+1,1)-v(lx,j-1,1))/(2*dy));
    end
    u(lx,j+1)=u(lx,j) + dt* (...
        -u(lx,j,1)*((3*u(lx,j,1)-4*u(lx-1,j,1)+u(lx-2,j,1))/(2*dx))
        -(v(lx,j+1,1)-v(lx,j-1,1))/(2*dy));
    end
  end
else if(sim==2)
  P(lx,:,1+1)=P(lx-1,:,1+1);
  u(lx,:,1+1)=u(lx-1,:,1+1);
  v(lx,:,1+1)=v(lx-1,:,1+1);
end

% Northeast Corner Boundary (i=lx and j=ly) (back x) (back y)
if(sim == 1)
  if(pres == 1)
    P(lx,ly,1+1) = P(lx,ly,1)+dt*(
        -gam*P(lx,ly,1)*((u(2,ly,1)-u(lx,ly,1))/(dx))
        +(v(lx,ly,1)-v(lx,ly-1,1))/(dy));
  end
  if(boundary == 2)
    u(lx,ly+1)=u(lx,ly-1+1);
    v(lx,ly+1)=v(lx,ly-1+1);
  end
end

% Southeast Corner Boundary (i=lx j=l) (back x) (for y)
if(sim == 1)
  if(pres == 1)
    P(lx,l,1+1) = P(lx,l,1)+dt*(
        -gam*P(lx,l,1)*((u(2,l,1)-u(lx,l,1))/(dx))
        +(v(lx+1,l)-v(lx,l,1))/(dy)));
  end
  if(boundary == 2)
    u(lx,l+1)=u(lx,2+1);
    v(lx,l+1)=v(lx,2+1);
  end
end

% West Edge Boundary (is=1 and 1<j<ly)
for j = 2:ly-1
  if(pres == 1)
    P(1,j,1+1) = P(1,j,1)+dt*(
        -gam*P(1,j,1)*((u(2,j,1)-u(1,j,1))/(dx))
        +(v(1,j+1,1)-v(1,j-1,1))/(2*dy)));
  end
end

% Southwest Corner Boundary (i=1 j=1)
if(pres == 1)
  P(1,1,1+1) = P(1,1,1)+dt*(
        -gam*P(1,1,1)*((u(2,1,1)-u(1,1,1))/(dx))
        +(v(1,2,1)-v(1,1,1))/(dy)));
if(pres == 1)
P(1,ly,l+1) = P(1,ly,l)+dt*(...
  -gam*P(1,ly,l)*((u(2,ly,l)-u(1,ly,l))/(dx))...
  +(v(1,ly,l)-v(1,ly-1,l))/(dy)));
end

%% Graphing Stuff
speed(:,:,1) = ((u(:,:,1).^2+v(:,:,1).^2).^0.5);
pause(10^-16)
figure(1);
clf;
view(90,-90)
surface(speed(:,:,1),'EdgeColor','none')
stepurple(num2str(lp-1) of 'num2str(time)')
title(num2str((lp-1)*time+1))
axis([0 ly 0 lx min(min(min(speed))) max(max(max(speed))) min(min(min(speed)))
max(max(max(speed)))));
colorbar()

if(1 == time)
  loop = 1;
end
end
end
toc
function [wing option] = objectmaker(lr, lx, ly)

option = 4;
wing = zeros(lx, ly);
winglm = zeros(lx, ly);

% Good Airfoil
if(option == 4)
    for i=1:lr
        for j=1:lr
            dist = norm([i j]-[(lr/3) (2.4*lr/5)]);
            if(dist<.08*lr)
                wing(i,j)=1;
            end
            dist = norm([i j]-[(2.3*lr/3) (2.5*lr/5)]);
            if(dist<.03*lr)
                wing(i,j)=1;
            end
        end
    end
    for i=lr/3:2.3*lr/3
        for j=1:lr
            if(j >= 0.16*i+.346*lr)
                wing(round(i),round(j))=1;
            end
            if(j >= -0.062*i+.58*lr)
                wing(round(i),round(j))=0;
            end
        end
    end
    wing = fliplr(wing);
end