**University of Massachusetts, Lowell**
Department of Mechanical Engineering


# Finite Element Solver for the Potential Flow Around an Airfoil

**Patrick Heng**

MECH 5200

Prof. D.J. Willis

04/16/25

## Abstract

In the following, we explore a MATLAB based finite element solver for the potential flow equation. The weak form of Poisson's equation will be derived and used in conjunction with a Galerkin approximation and linear basis functions to represent the solution. These approximations ultimately lead to a discrete matrix system which is readily solved numerically. As simple case studies, we will explore the flow around bluff bodies such as cylinders and NACA airfoils. By considering the velocity distribution around such bodies, we can better quantify the forces acting on them and choose designs accordingly.

## Academic Integrity

I hereby affirm that the following work submitted is my own and that I have not received help from others outside of this class. Furthermore, I shall reference any resources used outside of class lectures and notes.

– P.V. Heng

# Contents

# 1 Potential Flow Equation

Potential flow theory arises out of the full Navier-Stokes equations under the assumptions of an incompressible, inviscid, and irrotational fluid. While these assumptions are limiting, potential flow is still capable of modeling a wide class of flow behaviors. Perhaps the most famous application of potential flow is in solving for the velocity distribution over bluff bodies such as airfoils or cylinders. Mathematically, if we denote the fluid velocity as $\mathbf{v}$ and the velocity potential as $\phi$, then,

$$\mathbf{v} = \nabla\phi.$$

Incompressibility further implies,

$$\nabla \cdot \mathbf{v} = \nabla \cdot \nabla\phi = 0.$$

For the sake of generality, we will build a solver for the Poisson equation,

$$\nabla^2\phi = f,$$

where $f = 0$ for potential flow. Since we are interested in the flow over a bluff body centered at the origin, we impose a unit freestream condition at the far left and right domain boundaries. This means,

$$\phi(x_L, y) = x_L, \quad \phi(x_R, y) = x_R,$$

for some $x_L$ and $x_R$ far away from the origin. For computational purposes, we will assume symmetry about the $y = 0$ line and solve for the potential only on the upper half plane. We also wish for the flow not to interfere with the symmetry line or the top boundary, so Neumman conditions are imposed there, meaning,

$$\nabla\phi \cdot \mathbf{n} = 0.$$

Finally, the fluid cannot penetrate the bluff body, so another Neumman condition is imposed along the boundary of the body. We wish to test our finite element solver with these equations and boundary conditions. However, to do this, we need to derive the weak form of the potential flow equation, which is presented in the following section.

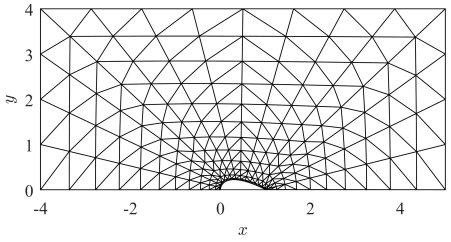



**Figure 1:** Computational domain for the potential flow around an airfoil

# 2 Weak Form of the Governing Equations

The strong formulation of Poisson's equation is given as,

$$\nabla^2 u - f = 0,$$

by multiplying by some test function, $w$, and integrating over the domain, $\Omega$,

$$\int_\Omega w\nabla^2 u \, d\Omega - \int_\Omega wf \, d\Omega = \int_\Omega wR \, d\Omega = 0,$$

where $R$ is a residual which is orthogonal to $w$ over $\Omega$. By applying integration by parts, we can expand the Laplacian term, moving a derivative from $u$ to $w$,

$$\int_\Gamma w\nabla u \cdot \mathbf{n} \, d\Gamma - \int_\Omega \nabla w \cdot \nabla u \, d\Omega - \int_\Omega wf \, d\Omega = \int_\Omega wR \, d\Omega.$$

For convenience, we break the boundary, $\Gamma$, into separate Dirichlet, $\Gamma_D$, and Nuemann, $\Gamma_N$, conditions.

$$\int_{\Gamma_D} w\nabla u \cdot \mathbf{n}\, d\Gamma + \int_{\Gamma_N} w\nabla u \cdot \mathbf{n}\, d\Gamma - \int_\Omega \nabla w \cdot \nabla u\, d\Omega - \int_\Omega wf\, d\Omega = \int_\Omega wR\, d\Omega.$$

If we have,

$$u|_{\Gamma_D} = g_D, \quad \nabla u \cdot \mathbf{n}|_{\Gamma_N} = g_N,$$

and we impose $w|_{\Gamma_D} = 0$, then, we arrive at the weak form of the Poisson equation,

$$\int_\Gamma wg_N\, d\Gamma - \int_\Omega \nabla w \cdot \nabla u\, d\Omega - \int_\Omega w\hat{f}\, d\Omega = \int_\Omega wR\, d\Omega,$$

where $\hat{f}$ is the modified forcing function such that the Dirichlet conditions are satisfied. Observe that the manipulations we have made thus far have been exact and no approximations have been made. However, to actually solve these equations, we will further choose $w$ to be our basis functions, $\varphi_i$, and expand $u$ in terms of these same basis functions,

$$u \approx \sum_j u_j \varphi_j,$$

where the $u_j$'s are unknown. With this, we arrive at the Galerkin approximation of the Poisson equation,

$$\int_\Gamma \varphi_i g_N\, d\Gamma - \sum_j u_j \int_\Omega \nabla\varphi_i \cdot \nabla\varphi_j\, d\Omega - \int_\Omega \varphi_i \hat{f}\, d\Omega = \int_\Omega \varphi_i R\, d\Omega.$$

For our problem, we have $g_N = 0$, thus,

$$\sum_j u_j \int_\Omega \nabla\varphi_i \cdot \nabla\varphi_j\, d\Omega + \int_\Omega \varphi_i \hat{f}\, d\Omega = \int_\Omega \varphi_i R\, d\Omega = 0.$$

In matrix form,

$$\mathbf{Ax} = \mathbf{f},$$

where,

$$\mathbf{A} = A_{ij} = \int_\Omega \nabla\varphi_i \cdot \nabla\varphi_j\, d\Omega,$$
$$\mathbf{x} = u_j,$$
$$\mathbf{f} = f_i = -\int_\Omega \varphi_i \hat{f}\, d\Omega.$$

Since we are solving the Laplace equation, the majority of $\mathbf{f}$ will be zeroes, but it will have non-zero contributions at the Dirichlet nodes.

# 3 Discussion of FEM

## Strong and Weak Forms of Equations

The strong and weak forms of PDE's concerns the regularity or smoothness of the solution and what is 'admissible' as a solution [1]. A brief summary of these considerations is presented as follows.

**Table 1:** Comparison of Strong and Weak Form for Second Order PDE's

| Strong Form | Weak Form |
|---|---|
| • If a solution exists, it is twice differentiable, i.e. only 'classical' solutions are admissible<br>• Solution is satisfied point-wise<br>• Physically, the strong form often represents a governing PDE, meaning a differential equation is solved for the unknown | • Weak solutions only need to be once differentiable<br>• Larger class of admissible solutions<br>• Solution is satisfied in an integral sense, 'almost everywhere' (except on the set of measure zero)<br>• Physically, the weak form often represents the integral form of a conservation law, which means solving an integral equation for the unknown<br>• A test function is used to 'measure' the solution; the weak form must hold for all differentiable test functions |

### Comparison of FDM, FVM, and FEM

Finite difference methods (FDM) are based on Taylor series expansions of the solution at selected *points* in the domain. Finite volume methods (FVM) are based on satisfying conservation for each *cell* in the domain, not necessary giving information about a single point, but rather, about the average behavior in the cell. Finite element methods (FEM) are based on the weak formulation of the governing equations and thus, the PDE is not necessarily satisfied at specific points, but rather, the equations are satisfied in an integral sense over all *elements* in the domain. That is, the FEM solution is such that the error over each element is as small as possible (orthogonal residual). Due to the use of Taylor expansions in FDM, the solution must satisfy the strong form of the PDE and have derivatives up to the order of the PDE. In comparison, FEM often represents the solution as piecewise polynomials. This means that not all derivative orders exist and solutions may have discontinuous derivatives that do not strongly satisfy the PDE. The use of piecewise polynomials also makes for simple high order interpolation of FEM solutions, a task which is not trivial in FDM. While FEM has many advantages, classical continuous FEM does not necessary respect conservation laws, which is the motivation for discontinuous Galerkin (DG) methods. DG uses the concept of fluxes from FVM to ensure that conservation is satisfied in the FEM framework, allowing for extension to hyperbolic problems.

## 4 2D Linear Hat Basis

As a simple demonstration of the finite element method, we will use a linear approximation over each triangular element to represent the solution. This means that each node will have a 'hat' function associated with it; these functions are 1 at a given node and 0 at all other nodes. This gives a 'global' basis function which looks similar to the following figure where the red dots are the nodes and the grey surface is the linear interpolation function.

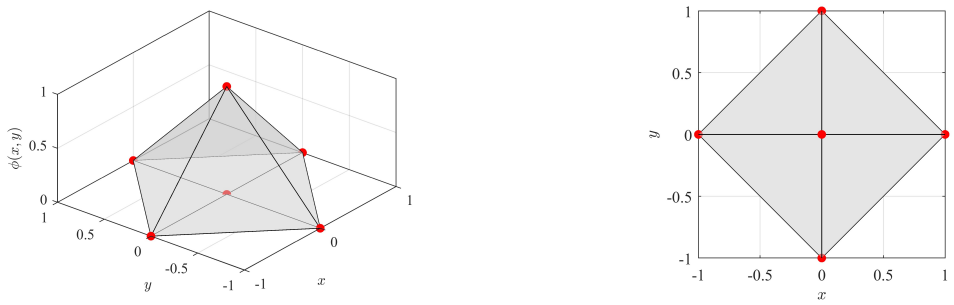



**Figure 2:** Global linear 'hat' basis function for a node connecting 4 triangular elements

For computational purposes, it is more convenient to formulate the FEM problem in terms of 'elemental' basis functions rather than global basis functions. This means at instead of looking at the entire domain, we restrict ourselves to a single element and find the basis functions on that element. For a triangular element with linear interpolation, there are 3 possible polynomials which are 1 at a given node and 0 at all other nodes. Examples of these elemental basis functions are shown in the following figure.

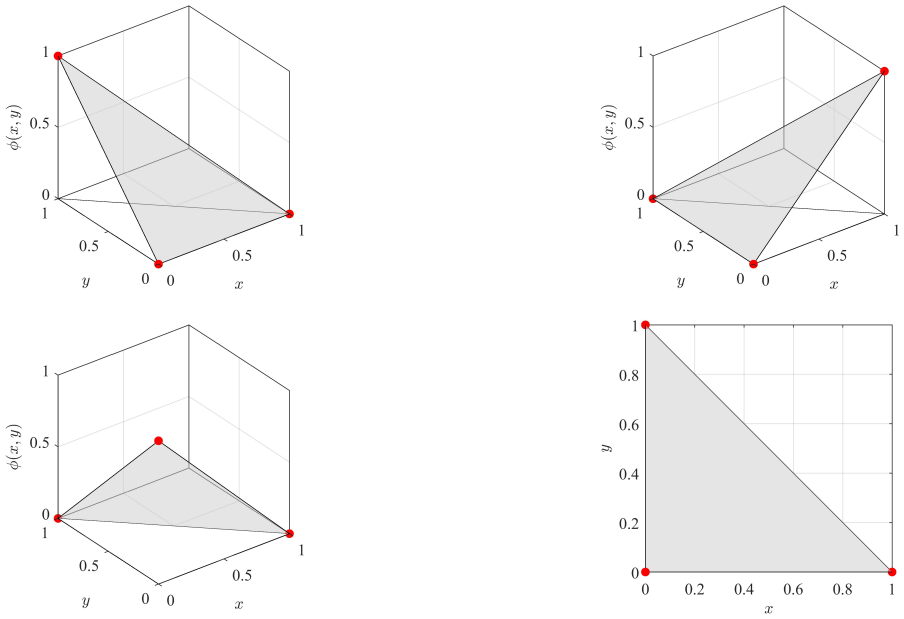



**Figure 3:** Elemental linear basis functions for a reference triangular element

# 5 Discretization

## Internal Nodes

For an arbitrary internal node, $i$, the equation it must satisfy is,

$$\sum_j u_j \int_\Omega \nabla\varphi_i \cdot \nabla\varphi_j \, d\Omega = -\int_\Omega \varphi_i \hat{f} \, d\Omega = 0,$$

for all basis functions, $\varphi_j$. In practice, this equation encodes the $i$th row of the $\mathbf{A}$ matrix and the sum only runs over the basis functions of neighboring nodes. Therefore, most of the of entries in the row are 0 and

the only non-zero contributions come from neighboring nodes, which allows the nodes to 'communicate'. However, since the mesh is generally unstructured, this does not necessarily mean that the non-zero entries are close together in the $\mathbf{A}$ matrix. While the global matrix formulation is how the system is actually solved, it is not intuitive to work with the global basis functions, which is why we will develop the elemental basis formulation in the following sections.

## Elemental Basis Matrices

Herein, we will interpret $\varphi_i$ to mean the elemental basis functions. Over a given element, a linear basis function assumes the form,

$$\varphi_i = a_i x + b_i y + c_i,$$

where $i = 1, 2, 3$ for the (arbitrary) local node numbering. The coefficients are solved by imposing the constraint that the basis function is 1 at node $i$ and 0 at all other nodes. For example, suppose an element has the set of 3 nodal coordinates, $(x_1, y_1)$, $(x_2, y_2)$, and $(x_3, y_3)$. This gives a set of 3 linear systems,

$$\begin{cases} a_1x_1 + b_1y_1 + c_1 &= 1 \\ a_1x_2 + b_1y_2 + c_1 &= 0 \\ a_1x_3 + b_1y_3 + c_1 &= 0 \end{cases}, \quad \begin{cases} a_2x_1 + b_2y_1 + c_2 &= 0 \\ a_2x_2 + b_2y_2 + c_2 &= 1 \\ a_2x_3 + b_2y_3 + c_2 &= 0 \end{cases}, \quad \begin{cases} a_3x_1 + b_3y_1 + c_3 &= 0 \\ a_3x_2 + b_3y_2 + c_3 &= 0 \\ a_3x_3 + b_3y_3 + c_3 &= 1 \end{cases}.$$

Which is put succinctly in matrix form,

$$\underbrace{\begin{pmatrix} x_1 & y_1 & 1 \\ x_2 & y_2 & 1 \\ x_3 & y_3 & 1 \end{pmatrix}}_{\mathbf{P}} \underbrace{\begin{pmatrix} a_1 & a_2 & a_3 \\ b_1 & b_2 & b_3 \\ c_1 & c_2 & c_3 \end{pmatrix}}_{\mathbf{Q}} = \underbrace{\begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}}_{\mathbf{I}}.$$

$\mathbf{P}$ and $\mathbf{Q}$ multiply to give the identity, therefore, we conclude $\mathbf{Q} = \mathbf{P}^{-1}$, which is easily solved with the inv() command in MATLAB. The elemental stiffness matrices are then given by the inner product between the gradients of the elemental basis functions. In general,

$$\nabla\varphi_i = \begin{pmatrix} \dfrac{\partial}{\partial x}\left(a_i x + b_i y + c_i\right) \\ \dfrac{\partial}{\partial y}\left(a_i x + b_i y + c_i\right) \end{pmatrix} = \begin{pmatrix} a_i \\ b_i \end{pmatrix},$$

Thus, by dotting each of the gradients,

$$\nabla\varphi_i \cdot \nabla\varphi_j = a_i a_j + b_i b_j.$$

which becomes the entry of the $i$th row and $j$th column in the elemental stiffness matrix. Since there are 3 basis functions per element, there are $3^2$ possible pairs of $(i, j)$, meaning the elemental stiffness matrix will be a $3 \times 3$ matrix of the form,

$$\mathbf{A}^e = \int_\Omega \nabla\varphi_i \cdot \nabla\varphi_j \, d\Omega = A^e_{ij} = \int_\Omega \begin{pmatrix} a_1a_1 + b_1b_1 & a_1a_2 + b_1b_2 & a_1a_3 + b_1b_3 \\ a_2a_1 + b_2b_1 & a_2a_2 + b_2b_2 & a_2a_3 + b_2b_3 \\ a_3a_1 + b_3b_1 & a_3a_2 + b_3b_2 & a_3a_3 + b_3b_3 \end{pmatrix} d\Omega.$$

Additionally, since the matrix only consists of constants, it may be pulled out of the integral and the integral simply becomes computing the area of the element. Computationally, if we extract the following vectors from the coefficient matrix, $\mathbf{Q}$,

$$\mathbf{a} = (a_1, a_2, a_3)^{\mathrm{T}}, \quad \mathbf{b} = (b_1, b_2, b_3)^{\mathrm{T}},$$

then formation of the elemental stiffness matrix simply becomes a sum of outer products,

$$\mathbf{A}^e = \frac{1}{2}||\mathbf{r}_2 \times \mathbf{r}_1|| \left(\mathbf{a}\mathbf{a}^{\mathrm{T}} + \mathbf{b}\mathbf{b}^{\mathrm{T}}\right),$$

where the cross product accounts for the triangle area. The $\mathbf{r}$ vectors given explicitly by,

$$\mathbf{r}_1 = (x_3 - x_1, y_3 - y_1, 0)^{\mathrm{T}}, \quad \mathbf{r}_2 = (x_2 - x_1, y_2 - y_1, 0)^{\mathrm{T}}.$$

## Elemental Load Vectors

The elemental load vector, $\mathbf{f}^e$, is given by taking the inner product between the forcing function and the elemental basis functions,

$$f_i^e = -\int_\Omega \varphi_i \hat{f}\, d\Omega.$$

By using linear basis functions, this simplifies to a weighted average (multidimensional trapezoid rule),

$$f_i^e = -\frac{1}{6}||\mathbf{r}_2 \times \mathbf{r}_1|| f_i.$$

Since $f = 0$ in our problem, $\mathbf{f}^e$ is the zero vector for all elements and nodes except the Dirichlet nodes.

## Global Stiffness Matrix and Load Vector

Placement of the elemental stiffness matrices in the global stiffness matrix requires knowledge of the global node numbering scheme (connectivity matrix) and which indices correspond to the nodes of the current element. However, given that these indices are known from the connectivity matrix, then updating the global $\mathbf{A}$ matrix simply becomes matching the elemental indices with the global indices. For example, say an element has global indices $(n_1, n_2, n_3)$ and corresponding local indices $(1, 2, 3)$, then,

$$\mathbf{A}_{n_1,n_1} = \mathbf{A}_{n_1,n_1} + \mathbf{A}^e_{1,1}, \quad \mathbf{A}_{n_1,n_2} = \mathbf{A}_{n_1,n_2} + \mathbf{A}^e_{1,2}, \quad \ldots \quad \mathbf{A}_{n_i,n_j} = \mathbf{A}_{n_i,n_j} + \mathbf{A}^e_{i,j},$$

and,

$$\mathbf{f}_{n_1} = \mathbf{f}_{n_1} + \mathbf{f}^e_1, \quad \mathbf{f}_{n_2} = \mathbf{f}_{n_2} + \mathbf{f}^e_2, \quad \mathbf{f}_{n_3} = \mathbf{f}_{n_3} + \mathbf{f}^e_3.$$

More concretely, say we have a row of the connectivity matrix with $(n_1, n_2, n_3) = (101, 51, 73)$ corresponding to local indices $(1, 2, 3)$, then we can apply an update to the following submatrix of $\mathbf{A}$,

$$\begin{pmatrix} A_{101,101} & A_{101,51} & A_{101,73} \\ A_{51,101} & A_{51,51} & A_{51,73} \\ A_{73,101} & A_{73,51} & A_{73,73} \end{pmatrix} = \begin{pmatrix} A_{101,101} & A_{101,51} & A_{101,73} \\ A_{51,101} & A_{51,51} & A_{51,73} \\ A_{73,101} & A_{73,51} & A_{73,73} \end{pmatrix} + \begin{pmatrix} A^e_{1,1} & A^e_{1,2} & A^e_{1,3} \\ A^e_{2,1} & A^e_{2,2} & A^e_{2,3} \\ A^e_{3,1} & A^e_{3,2} & A^e_{3,3} \end{pmatrix}.$$

This update is more simply stated in MATLAB syntax, let 'n' be a row of the connectivity matrix and 'A_e' be the corresponding elemental stiffness matrix, then,

$$\mathrm{A(n, n) = A(n, n) + A\_e;}.$$

Likewise, for the global load vector,

$$\mathrm{f(n) = f(n) + f\_e;}.$$

In practice, we loop over each element by first selecting a row of the connectivity matrix, then calculate the elemental stiffness and load vectors corresponding to those nodes. The above stamping procedure is then performed to place the elemental matrices and vectors into the global system. Once the loop is done, the global $\mathbf{A}$ matrix and $\mathbf{f}$ vector will account for all of the interactions between the nodes and the only step that remains is to make sure the boundary conditions are satisfied.

## Boundary Nodes

Boundary conditions are generally applied after the stamping of the global stiffness matrix. At the Neumann nodes, $u_i$ satisfies the equation,

$$\int_\Gamma \varphi_i g_N\, d\Gamma - \sum_j u_j \int_\Omega \nabla\varphi_i \cdot \nabla\varphi_j\, d\Omega = -\int_\Omega \varphi_i \hat{f}\, d\Omega = 0.$$

Since $g_N = 0$, the boundary integral vanishes, meaning that the $i$th row of the $\mathbf{A}$ matrix is associated with the equation,

$$\sum_j u_j \int_\Omega \nabla\varphi_i \cdot \nabla\varphi_j\, d\Omega = -\int_\Omega \varphi_i \hat{f}\, d\Omega = 0.$$

This is the same equation as an internal node, which means the rows corresponding to Neumann conditions are simply left alone and automatically satisfy the boundary condition. At the far left and right boundaries, we have a uniform free stream velocity, meaning the Dirichlet nodes at those boundaries satisfy,

$$u_i = x_i.$$

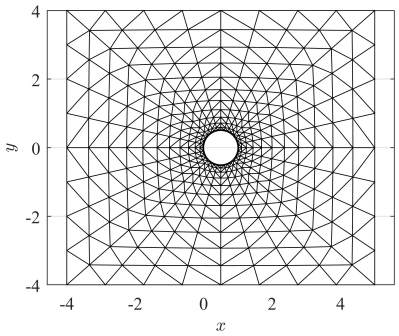
where $x_i$ is the $x$ coordinate of the node. This means wiping out the corresponding row of the $\mathbf{A}$ matrix and replacing it with the same row of the identity matrix; the global load vector is also set to $f_i = x_i$. In MATLAB code, if 'n' denotes the Dirichlet nodes, 'eye' denotes the identity matrix, and 'X' denotes the $x$ location of the nodes, then imposing the boundary conditions becomes,

$$\mathrm{A(n,:) = eye(n,:);},$$

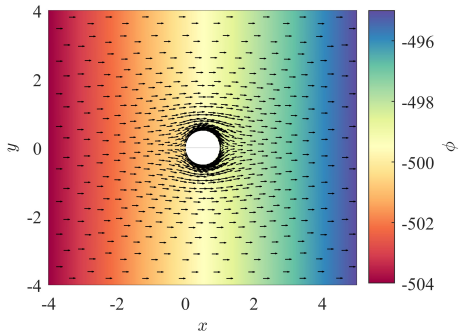
$$\mathrm{f(n) = X(n);}.$$

# 6 Flow Over a Cylinder

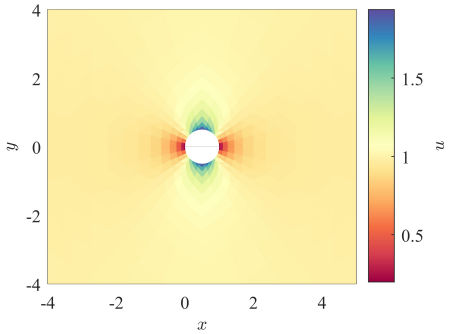
As a first test for the solver, we will analyze the flow over a cylinder with a unit diameter and forward stagnation point located at $(x, y) = (0, 0)$. For all simulations, the 'DomainSize' was set to 4, meaning $x \in [-4, 5]$ and $y \in [-4, 4]$. Additionally, the 'powerRef' setting was set to 2, giving a more refined mesh around the boundary of the bluff body. The following figures present the mesh, velocity potential, $\phi$, and velocity magnitude, $u$, for 'ref' settings of 3 (coarse) and 8 (fine). For plotting purposes, the velocity potential was offset by a constant $-500$. This is valid since the physical quantity of interest is the velocity, which is the gradient of the potential, so adding a constant to the potential does not affect the physical solution.




**(a)** Triangulated mesh




**(b)** Velocity potential

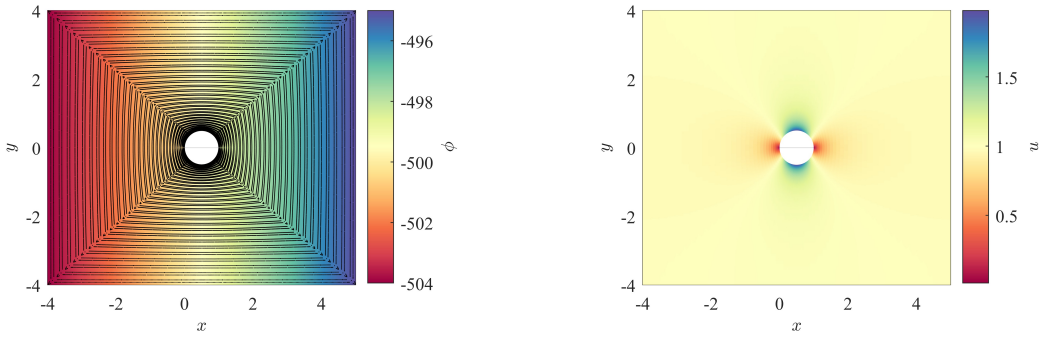



**(c)** Velocity magnitude

**Figure 4:** FEM potential flow solution for flow over a cylinder with ref = 3

From this, we see that the potential increases from $-504$ to $-495$ across the domain, a 9 unit increase. This is logical since for uniform flow in absence of a body, the potential should increase linearly across the

domain, so an 9 unit long domain should result in a 9 unit increase in the potential. The presence of the body changes this slightly, but we should still expect the potential to be essentially uniform except at positions very close to the body. The velocity field is also reasonable since we know from potential flow theory that there are stagnation points (places where $u = 0$) at the front and back of the cylinder. This as validated by the red regions near $(x, y) = (0, 0)$ and $(x, y) = (1, 0)$. Additionally, since the flow cannot penetrate the cylinder, it must accelerate around it, which leads to the blue regions at the top an bottom of the cylinder. To further test that the refinement is working properly, we present another simulation with ref = 8, the mesh is excluded as it is too fine to visualize.




**(a)** Velocity potential **(b)** Velocity magnitude

**Figure 5:** FEM potential flow solution for flow over a cylinder with ref = 8

We see that the refinement works as intended as the solution looks similar to the less refined mesh, but is visually smoother since there are more elements representing the solution, especially near the boundary of the cylinder. However, from the velocity vector field, we may also observe distinct diagonal bands across the solution domain. This is due to a lack of refinement for elements along the diagonals which arises out of trying to fit triangular elements into a rectangular domain. This problem could be mitigated by using quadrilateral elements which may fit around the cylinder and rectangular domain better.

## Mesh Convergence

To further test the convergence of the FEM solver, simulations were run with mesh refinements ranging from 3 to 8 in steps of 0.5. The solution of the potential field at the forward stagnation point, $\phi_{stag}$, was monitored over these meshes and plotted in the following figure.

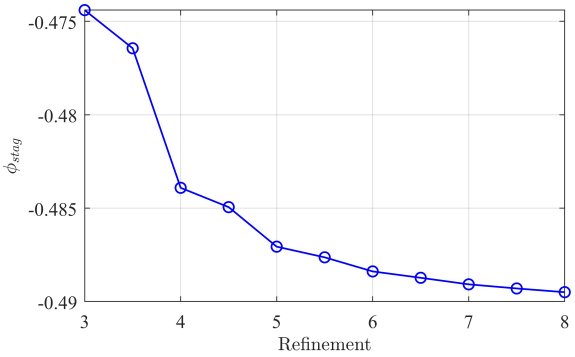


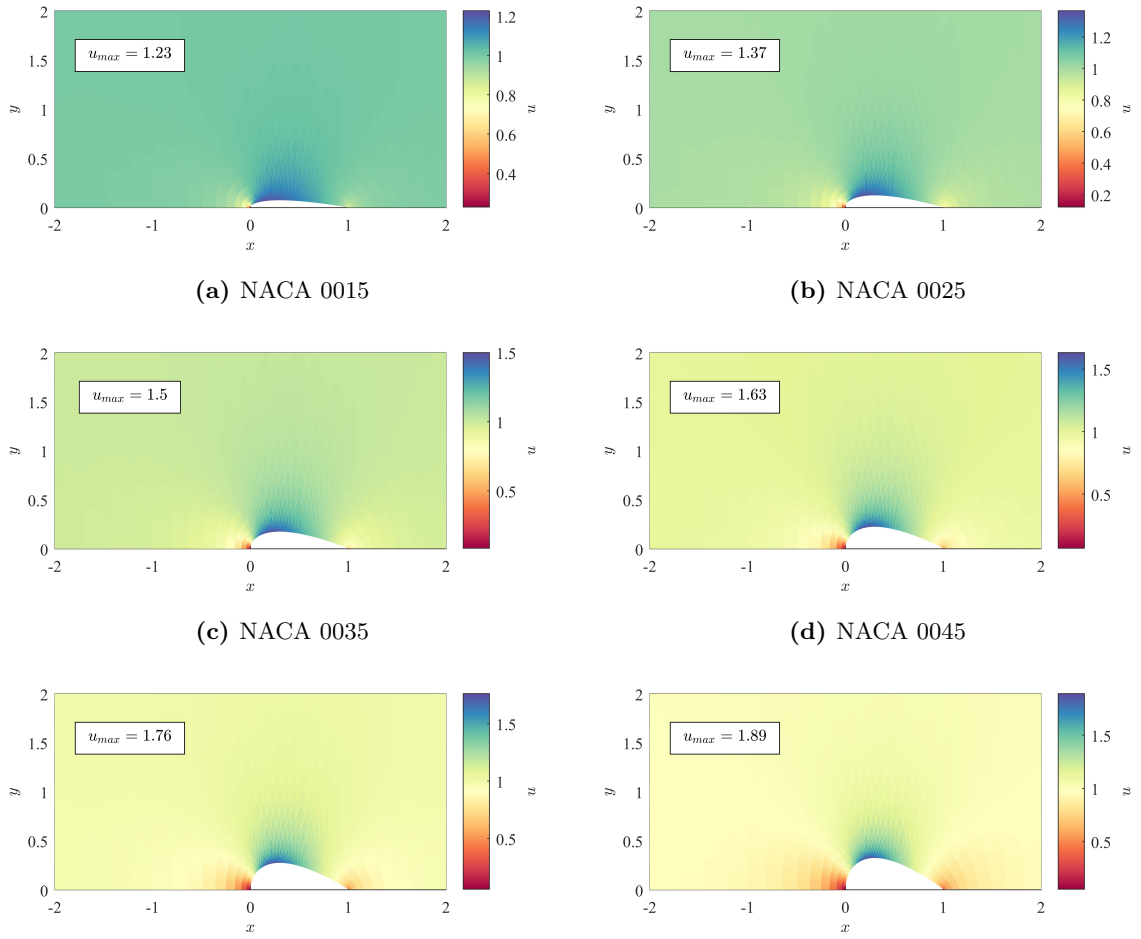

**Figure 6:** Mesh convergence test for $\phi$ at the forward stagnation point of a cylinder

For the finer solutions near ref = 8, we start to see an asymptotic convergence to a value near $\phi_{stag} \sim -0.49$ as successive refinements do not lead to significant changes in the potential field. Ideally, we should test

higher mesh refinements, but for any refinement above 8, the time to generate the mesh becomes prohibitive. To decrease these costs, further studies may try to increase the power refinement of the mesh while keeping the overall refinement low in the rest of the domain. For the coarser meshes, we see that the transition from ref = 3 to ref = 5 leads to a (relatively) significant change in the potential field. This means that the error in the solution decreases significantly for only a minor increase in the computation time. This suggests that a refinement of 5 is a reasonable middle ground for a low solution error and small computational intensity.

# 7 Flow Over an Airfoil

As an application, we will test the flow over NACA 4 digit airfoils which will be used to model the flow over the roof of a car. We wish place a wiring box for a sensor on the roof and therefore want the velocity at the top to be as small as possible. One of the main variables in NACA airfoils is its thickness, which would correspond to the height of the roof. This is determined by the last 2 digits of the 4 digit code, with a larger number corresponding to a larger thickness. Airfoils with codes 0015-0065 were tested and have their max velocities, $u_{max}$, presented. The max velocity occurs near the thickest part of the airfoil and thus is a good indicator for the velocity and forces that the sensor box must withstand.




**(a)** NACA 0015 **(b)** NACA 0025

**(c)** NACA 0035 **(d)** NACA 0045

**(e)** NACA 0055 **(f)** NACA 0065

**Figure 7:** FEM potential flow solution for flow over NACA airfoils with varying thicknesses (ref = 5)

From this, we see that the thicker airfoils have a larger $u_{max}$ since the curvature at the front is larger, meaning

the fluid will accelerate more. If the fairing box is installed perpendicular to the flow, we would ideally need its height to be as small as possible in order to reduce the curvature of the airfoil and acceleration of the fluid. Additionally, if the fairing box extrudes from the airfoil, it must be able to withstand the reaction force resulting from the momentum of the fluid. A higher velocity implies and a higher momentum and thus a higher force, therefore, we would ideally like the airfoil to be as thin as possible so that the sensor needs to withstand the least amount of force.

# References

[1] Evans L.C., *Partial Differential Equations*, 2nd Ed., American Mathematical Society, 2010.

[2] Haberman, R., *Applied Partial Differential Equations with Fourier Series and Boundary Value Problems,* 3rd Ed., Prentice Hall, 1997.

## Appendix A - Test Scripts

```
% ------------------------------------------------------------------
% Patrick Heng --> A matrix assembly and solution
% 3 Apr. 2025
% FEA/FEM script for potential flow problem over a bluff body,
% adapted from code provided by Prof. D.J. Willis.
% ------------------------------------------------------------------

% ==================================================================
% Start by defining your domain: Meshing parameters (done for you)
% ==================================================================
% The following parameters are used for defining the mesh of the
% object
%
% Shape: This selects an ellispoid or a NACA-4 Digit airfoil
%     -- Ellipsoid: Set the shape to a single value (the aspect
%                   ratio of  the ellipsoid shape. eg: Shape = [1];
%
%     -- NACA-4-Digit airfoil: Set the shape parameter a 2-valued
%                              entry,
%     eg: Shape = [1 0012], where the second value is the 4-digit
%         reference for the NACA airfoil.
%
% DomainSize: This is a basic parameter that defines how far the
% domain should extend (approx.) from your shape. A value of 4 should
% be sufficient, but you can play around with this.
%
% ref: This is the refinement of the mesh. The higher the value, the
% more elements and vertices you will have.
%
% powerRef: This is the mesh refinment control. The larger the
% number, the more refined the mesh is near the object. A value of
% 1 produces a uniform refiment. A value of 1.75-2.0 should work well
% for most of your problems.
%
% ==================================================================

close all; clear variables; clc;

% Colormaps from '200 Colormaps' on the MATLAB file exchange
% SWITCH THIS TO A DEFAULT COLORMAP ('parula') IF YOU DO NOT HAVE THE
% FILE INSTALLED
cmap=slanCM(100);


% Viewing window for velocity plot - set 0 for full view
window = 0;

% Flag to display max velocity text
show_max_velocity = true;


% ------------------------------------------------------------------
```

```
% ----- INPUTS -----

% Forcing function for Poisson equation - accounts for
% compressibility in the case of potential flow. Set to @ 0*X for
% incompressible flow (Laplace equation)
RHS = @(X,Y) 0*X;

% Mesh generation parameters
Shape = [1];
DomainSize = 4;
ref = 5;
powerRef = 2;


% ------------------------------------------------------------------------
% ----- MESH GENERATION -----
[TRI, Nodes, DirichletNodes] = getDiscreteGeometry(Shape, ...
                                    DomainSize, ref, powerRef);

% Plot triangulation
%
figure
trimesh(TRI, Nodes(:,1),Nodes(:,2),0*Nodes(:,1),edgecolor='k')
hold on
trimesh(TRI, Nodes(:,1),-Nodes(:,2),0*Nodes(:,1),edgecolor='k')
view([0,0,1])
axis equal


xlabel('$x$',interpreter='latex')
ylabel('$y$',interpreter='latex')
fontname('Serif'); fontsize(16,'points')
box on
%}

% ------------------------------------------------------------------------
% ----- A MATRIX ASSEMBLY -----
tic
% Determine the number of nodes and elements in the domain
num_nodes = length(Nodes);
num_elem = length(TRI);

% Initialize the global A matrix and RHS
global_A_mtx = spalloc(num_nodes, num_nodes, 12*num_nodes);
f = RHS(Nodes(:,1),Nodes(:,2));   % Evaluate forcing function
F = zeros(num_nodes, 1);                % Preallocate RHS
% Matrix to solve for elemental basis coefficients
basis_mtx = ones(3,3);
I = eye(3);                             % Identity matrix

% Stamp global A matrix with elemental stiffness matrices and
% elemental load vectors by looping through all elements
for i = 1:num_elem
    % Triangle node numbers
```

```
    node_ID = TRI(i,:);
    % Coordinates of triangle node numbers
    basis_mtx(:,1) = Nodes(node_ID,1);
    basis_mtx(:,2) = Nodes(node_ID,2);

    % Element basis function coefficients:
    % Linear element basis: phi(x,y) = ax + by + c
    basis_coeffs = basis_mtx\I;
    % Get a and b coefficients for stiffness matrix calculation
    a = basis_coeffs(1,:);
    b = basis_coeffs(2,:);

    % Calculate element size (area)
    side1 = [Nodes(node_ID(1),:) - Nodes(node_ID(2),:),0];
    side2 = [Nodes(node_ID(2),:) - Nodes(node_ID(3),:),0];
    tri_area = 0.5*norm(cross(side1,side2));

    % Inner product of basis function gradients (stiffness matrix)
    elem_mtx = tri_area*(a'*a + b'*b);

    % Place elemental stiffness matrix into global A matrix
    global_A_mtx(node_ID,node_ID) = global_A_mtx(node_ID,node_ID) ...
                                        + elem_mtx;
    % Place elemental load vector into global load vector
    F(node_ID) = F(node_ID) - tri_area*f(node_ID)/3;
end

% ---------------------------------------------------------------------
% ----- BC's -----

% For Dirichlet nodes, replace the rows corresponding to these nodes
% with the same rows of the identity matrix. For load vector, replace
% the Dirichlet nodes with desired boundary value
I = speye(num_nodes);
global_A_mtx(DirichletNodes,:) = I(DirichletNodes,:);
F(DirichletNodes) = Nodes(DirichletNodes,1);

% ---------------------------------------------------------------------
% ----- SOLVE ----

phi = global_A_mtx\F;
toc

% ---------------------------------------------------------------------
% ----- POST PROCESSING -----

% Scalar potential value at foward stagnation point (0,0)
phi_stag = phi(Nodes(Nodes(:,1)==0,2)==0)


% =====================================================================
% Gradient calculation
% =====================================================================
for i=1:num_elem
```

```
    element_number = i;

    N1 = TRI(i,1);
    N2 = TRI(i,2);
    N3 = TRI(i,3);

    X1 = Nodes(N1,1);
    X2 = Nodes(N2,1);
    X3 = Nodes(N3,1);

    Y1 = Nodes(N1,2);
    Y2 = Nodes(N2,2);
    Y3 = Nodes(N3,2);

    S1 = phi(N1);
    S2 = phi(N2);
    S3 = phi(N3);

    C = [1 X1 Y1; 1 X2 Y2; 1 X3 Y3]\[eye(3)];

    Gradient_IE(i,:) = [(S1*C(2,1)+S2*C(2,2)+S3*C(2,3)), ...
        (S1*C(3,1)+S2*C(3,2)+S3*C(3,3))];

    Centroid(i,:) = [(X1+X2+X3)/3, (Y1+Y2+Y3)/3];
end

% ======================================================================
% Figure 1
% ======================================================================
figure
trisurf(TRI, Nodes(:,1), Nodes(:,2), phi-500)
hold on
trisurf(TRI, Nodes(:,1),-Nodes(:,2),phi-500)

quiver(Centroid(:,1), Centroid(:,2), ...
            Gradient_IE(:,1), Gradient_IE(:,2),.75,'k')
quiver(Centroid(:,1), -Centroid(:,2), ...
            Gradient_IE(:,1), -Gradient_IE(:,2),.75,'k')

shading interp
%title(['Scalar Velocity Potential Distribution ' ...
%            '(Nodal) and the Velocity Vector Field'])
view([0 0 1])
axis equal
colormap(cmap)
cb = colorbar;
xlim([min(Nodes(:,1)),max(Nodes(:,1))])
ylim([-max(Nodes(:,2)),max(Nodes(:,2))])

box on
grid on
xlabel('$x$',interpreter='latex')
ylabel('$y$',interpreter='latex')
zlabel('$\phi$',interpreter='latex')
```

```
ylabel(cb,'$\phi$',interpreter='latex')
fontname('Serif'); fontsize(16,'points')

% ========================================================================
% Figure 2
% ========================================================================
figure
Vel = ((Gradient_IE(:,1).^2 + Gradient_IE(:,2).^2).^(.5))';
a = trisurf(TRI, Nodes(:,1), Nodes(:,2), Nodes(:,2)*0, Vel);
hold on

set(a,'edgealpha',0)
trisurf(TRI, Nodes(:,1), -Nodes(:,2), -Nodes(:,2)*0, Vel, ...
                linestyle='none')
%title('Velocity Distribution (Centroidal)')
view([0 0 1])
axis equal

colormap(cmap)
cb = colorbar;

if window == 0
    xlim([min(Nodes(:,1)),max(Nodes(:,1))])
    ylim([-max(Nodes(:,2)),max(Nodes(:,2))])
else
    xlim([-window,window])
    ylim([0,window])
end

if show_max_velocity == true
    u_max = max(Vel,[],'all');
    annotation('textbox',[0.19,0.65,0.1,0.1],...
        'string',['$u_{max}=' num2str(u_max,3) '$'],...
        interpreter='latex',backgroundcolor='w', ...
        verticalalignment='middle', ...
        HorizontalAlignment='center')
end

box on
grid on
xlabel('$x$',interpreter='latex')
ylabel('$y$',interpreter='latex')
zlabel('$u$',interpreter='latex')
ylabel(cb,'$u$',interpreter='latex')
fontname('Serif'); fontsize(16,'points')
```