University College London

MATH0084
Project in Mathematics

Investigations on MFS and Lightning solver

Author:
Yinuo Huang
20018162

Supervisor:
Prof. Edward Johnson

March 02, 2024

Contents

Chapter 1

Introduction

Our method is based on ideal fluid which satisfy the Laplace's equation .

$$\nabla^2 f(z) = 0$$

$$w = u - iv$$

$$f(z) = \phi(x,y) + i\psi(x,y)$$

$$\frac{\partial u}{\partial x} + \frac{\partial v}{\partial y} = 0$$

$$\frac{\partial^2 \phi}{\partial x^2} + \frac{\partial^2 \phi}{\partial y^2} = 0$$

The theory is based on the ideal fluid. And the fluid is inviscid and incompressible. So the Laplace's equation could be used. We focus on a two-dimensional fluid, and thus, use the complex axis $z = x + yi$.

Then we have the domain D in unbounded and simply connected. Although the Laplace's equation is a simply PDE to solve, computational complexities arise when these equations are applied to non-smooth domains, notably corners. Normally we use BEM and FEM to simulate. But they still need strong computational power and expert knowledge to solve these singularities which is not smooth.Obviously we need a new technique to mitigate the computational intricacies associated with non-smooth corners. So we use a new method "lightning solver" to solve these problem by using rational function approximation method. This method rapidly and accurately computes these singular points, effectively resolving the singularity issues encountered in simulations. Whats more , we also need to use Method of fundamental solutions (MFS) to solve the smooth area except corner.

Due to rich complex analytical theory, we could solve the problem on potential flow easier.We could also use conformal mapping or Joukowski transformation we learn before to convert complex shapes into simpler

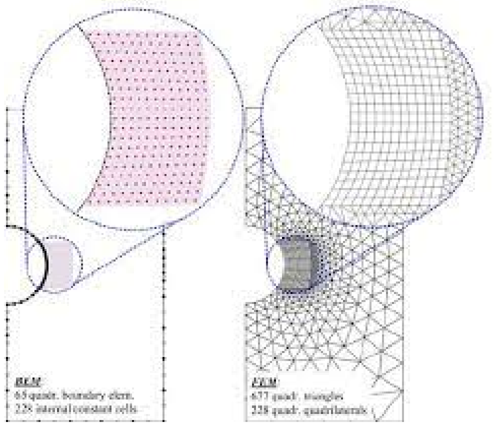


Figure 1.1: Enter Caption

forms for computation. This method typically yields bounded solutions and is a cornerstone in aerodynamics—for instance, translating aircraft wing shapes into circles. Although solutions outside the boundary layer are easily obtainable, internal fluid flow computations become complex due to the influence of the boundary layer. Progressive approximations and ensuring solution matches on both sides of the bound are essential. We also need to consider the effect of fluid separation. They usually cause a huge change after the corner appear. And we normally use Brown-Michael equation to model point vortices shed from sharp corners. Moreover we still need to solve the problem on multiply connected flow domains by transcendental Schottky–Klein prime function, Kirchhoff–Routh path function and so on. But how these corners play a

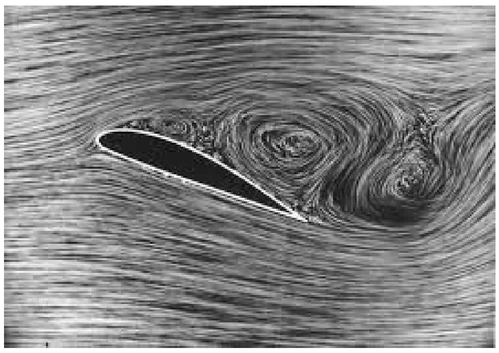


Figure 1.2: Enter Caption

important role in real life? Corners play a significant role in real-world applications, notably in aerodynamics. For instance, air is expected to leave the aircraft wings smoothly, but the presence of corners can introduce significant perturbations, affecting the overall airflow and vortices, and subsequently, the lift. Hence, precise corner modeling is imperative to achieve accurate simulation results.

Overall ensuring the utmost precision in corner computations is paramount in fluid engineering. The implementation of the Lightning Solver markedly enhances model accuracy and simulation efficiency. In the subsequent discussions, the detailed implementation of the Lightning Solver and its potential impacts on advancing the frontiers of fluid dynamics simulations will be elaborated upon, promising a future where complex fluid dynamics problems can be resolved with unprecedented efficiency and precision.

Chapter 2

Free Grid solution on Laplace Equation

2.1 Fundamental Solutions Method in Potential flow

In the context of fluid dynamics and potential flow studies, the Fundamental Solution Method (MFS) offers a significant numerical approach. This method is particularly suited for solving Partial Differential Equations (PDEs) such as Laplace's equation given by $\nabla^2 u = f$, where u represents the potential function to be determined and f denotes the known source distribution, embodying the essence of natural boundary conditions.

The main advantage of the MFS lies in its direct application to PDEs through the so-called "fundamental solutions" — analytical solutions to PDEs under specific point source stimulations. For instance, in two-dimensional space, the fundamental solution for the Laplace equation can be expressed as

$$\Phi(r) = -\frac{1}{2\pi} \log r,$$

where r is the distance from the point source to the point of interest. This expression reveals the essence of the potential function induced by a point source, thereby simplifying complex boundary conditions.

In MFS, the solution within a domain is constructed by superimposing the effects of several singular sources (or singularities), positioned outside the domain of interest. Mathematically, the approximate solution at any point P within the domain D is given by:

$$u_n(P) = \sum_{i=1}^{n} c_i \Phi(|P - Q_i|),$$

where Q_i represents the position of the i-th singularity, and $|P - Q_i|$ measures the distance between point P and singularity Q_i.

By adjusting the coefficients c_i, MFS can precisely capture the characteristics of the flow, making it an excellent tool for solving boundary value problems across various domains. Furthermore, as the method avoids the complexities involved in traditional discretization techniques, its efficiency is significantly enhanced, offering a clear pathway to understanding the underlying physics without the need for dense mesh grids.

The adaptability and accuracy of the method largely depend on the appropriate selection of the positions of the singularities and the coefficients c_i; these parameters crucially influence the convergence and stability of the solution. Typically, the strategic placement of singularities and the optimization of their associated coefficients enable the method to achieve high levels of accuracy, particularly in areas where traditional methods may struggle.

2.2 Question on circle

We consider the scenario where fluid navigates past a cylindrical obstacle situated between two parallel boundaries. For the sake of analysis, we assume the flow to be uniform along one direction, specifically in the positive x-axis, with the cylinder positioned at the coordinate origin. The setup is depicted as a two-dimensional cross-section, akin to that presented in Figure 1.1, with the boundaries delineated by the lines $y = \pm a$, symbolizing the parallel plates. In this model, $\Phi(P)$ represents the flow potential at any given point $P(x, y)$ within the domain D or on its boundary ∂D, where D encompasses the fluid-filled region. Accordingly, the potential Φ is governed by Laplace's equation within the domain:

$$\nabla^2 \Phi(P) = 0, \quad P \in D.$$

Consider a fluid flow scenario with boundary conditions given by:

$$\phi(P) = g_1(P), \quad P \in \partial D_1,$$

$$B\phi(P) = \left.\frac{\partial \phi}{\partial n}\right|_P = g_2(P), \quad P \in \partial D_2,$$

where $\partial D_1 + \partial D_2$ constitutes the boundary of domain D. It is noted that $\frac{\partial}{\partial n}$ denotes the normal flow component at the boundary. Given that

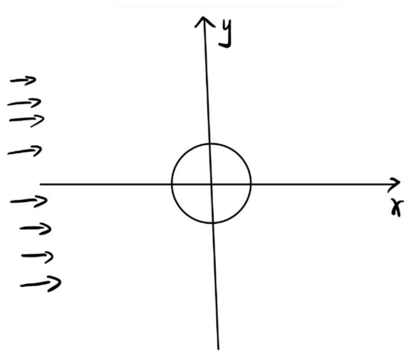


Figure 2.1: Fluid around circle

the potential function ϕ is determined, one can compute the flow field components $u(P)$ and $v(P)$ at any point $P \in D$ using:

$$u(P) = \frac{\partial \phi}{\partial x}(P), \quad v(P) = \frac{\partial \phi}{\partial y}(P).$$

The formulation can also be cast in terms of the stream function $\psi(P)$, which satisfies the same Laplace's equation and boundary conditions as $\phi(P)$. Hence, the flow field components are given by:

$$u(P) = \frac{\partial \psi}{\partial y}(P), \quad v(P) = -\frac{\partial \psi}{\partial x}(P).$$

Regardless of whether the function ϕ or ψ is used, the mathematical challenge remains equivalent. To approximate the solution $\phi(P)$ of the boundary-value problem, the Method of Fundamental Solutions (MFS) employs the expression:

$$\phi(c, Q; P) = \sum_{i=1}^{n} c_i \log |Q_i - P|, \quad P \in D,$$

where $c = [c_1, \ldots, c_n]^T$ and Q is a 2-vector indicating the singularities' coordinates, presumed to lie outside D. Following this, the coefficients c_i and the locations Q_i are chosen to optimally satisfy the boundary conditions in a least squares sense by minimizing the functional:

$$S(c, Q) = \sum_{i=1}^{m} \left[B\phi(c, Q; P_i) - BQ(P_i) \right]^2,$$

where $\{P_i\}_{i=1}^{m}$ are points on the boundary ∂D, and m is chosen such that $m > 3n$ to ensure an over-determined system for a stable solution. The singularities' positions Q_i and the coefficients c_i are thus optimized to fulfill the boundary conditions as accurately as possible.

It is important to note that this approach inherently involves a nonlinearity with respect to the singularities' coordinates Q_i. Nevertheless, efficient algorithms for nonlinear least squares problems are well established, mitigating this challenge. In particular, the ability of the MFS to adapt the singularities' placement offers a significant advantage, automatically tailoring the solution to the specifics of the flow problem at hand.

2.3 Method

The flow is around a unit circular cylinder. Firstly, we need to set up the sample points on the boundary of our problem to implement the boundary conditions.

```
let angles = linspace(0, 2*pi, num_points);
let samplepoint = e^(i * angles);
let x = Re(samplepoint);
let y = Im(samplepoint);
```

Next, we need a scaling factor to determine the positions of our poles, scaling the sample points around the origin.

```
let pole_x1 = x * factor;
let pole_y1 = y * factor;
for i = 1 to (num_points/2) do

pole_x(i) = pole_x1(2*i-1);
pole_y(i) = pole_y1(2*i-1);
```

For the previous equation we talk about, we need to calculate this to implement $Ac = D$ and c is the coefficients of $f(z)$, So it is a least-squares problem now.

$$f(z) = \sum_{i=1}^{n} a_i \log |z - \hat{z}_i|, \quad \hat{z}_i \in D,$$

Let A be a matrix of size $M \times N$, $z \in \mathbb{R}^M$, $\hat{z} \in \mathbb{R}^N$, and $d \in \mathbb{R}^M$. Here M (200) is the number of samplepoint and N (1 00) is the number of polepoint. d is the boundary condition which is M $\times$ 1

$$A = \begin{pmatrix} \log|z_1 - \hat{z}_1| & \log|z_1 - \hat{z}_2| & \cdots & \log|z_1 - \hat{z}_N| \\ \log|z_2 - \hat{z}_1| & \log|z_2 - \hat{z}_2| & \cdots & \log|z_2 - \hat{z}_N| \\ \vdots & \vdots & \ddots & \vdots \\ \log|z_M - \hat{z}_1| & \log|z_M - \hat{z}_2| & \cdots & \log|z_M - \hat{z}_N| \end{pmatrix}$$

$$\mathbf{d} = \begin{pmatrix} d_1 \\ d_2 \\ \vdots \\ d_M \end{pmatrix}$$

where M is the number of sample points and N is the number of pole points.

Our boundary condition is $\mathrm{Im}[f(z)] = d(z)$.

This study proposes a novel methodology for the least-square method coefficients, which is matrix c ($c \in \mathbb{R}^N$). The results, illustrated through the contour plots, demonstrate significant improvement, particularly in the range from -5 to 5 on the specified scale.

Now we have the graph,

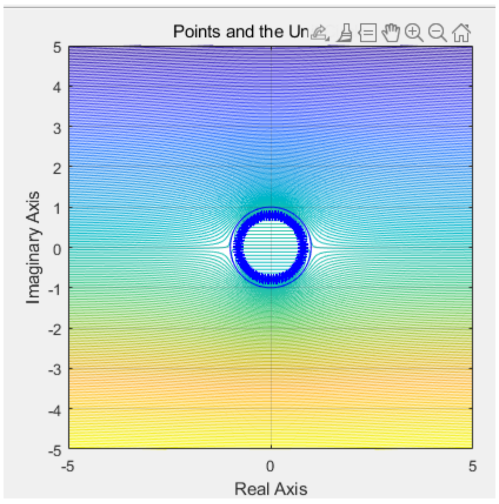


Figure 2.2: -5 to 5

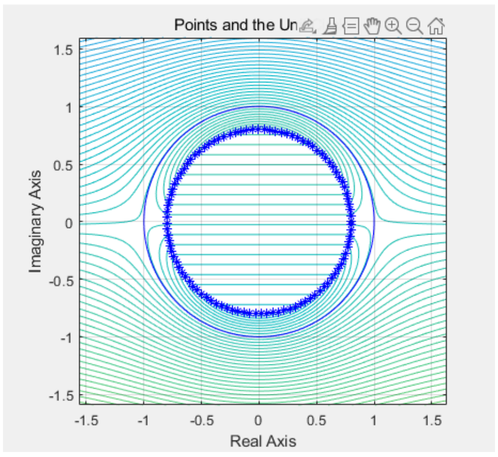
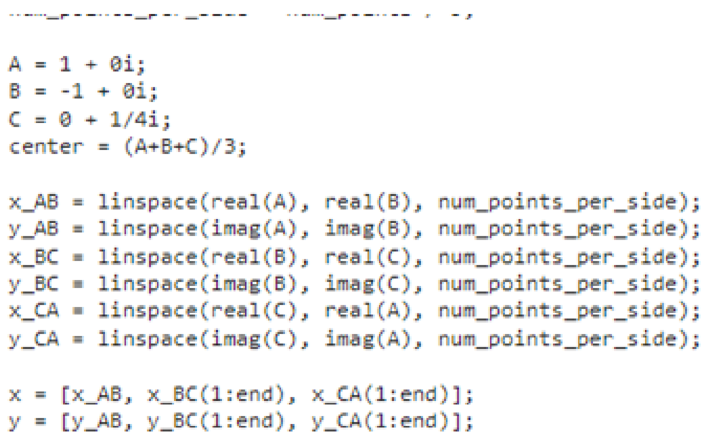


Figure 2.3: More precision

We could see that the streamline flow around the circle and do not across the boundary. This graph is perfect for our assumption.

2.4 Question on triangle

Now, let us begin the discussion on cases with singularities. The immediate shape that comes to mind is a triangle. Let us examine whether selecting poles in the same manner is still applicable for non-smooth boundaries. To construct a triangle,



```
A = 1 + 0i;
B = -1 + 0i;
C = 0 + 1/4i;
center = (A+B+C)/3;

x_AB = linspace(real(A), real(B), num_points_per_side);
y_AB = linspace(imag(A), imag(B), num_points_per_side);
x_BC = linspace(real(B), real(C), num_points_per_side);
y_BC = linspace(imag(B), imag(C), num_points_per_side);
x_CA = linspace(real(C), real(A), num_points_per_side);
y_CA = linspace(imag(C), imag(A), num_points_per_side);

x = [x_AB, x_BC(1:end), x_CA(1:end)];
y = [y_AB, y_BC(1:end), y_CA(1:end)];
```

Figure 2.4: Build the sample point

in the complex coordinate system, we choose three vertices at (1,0), (-1,0), and (0,1/4), with a factor of 0.8. Subsequently, we construct the model using a third of the number of pole points as sample points. Below is the image constructed under these conditions.

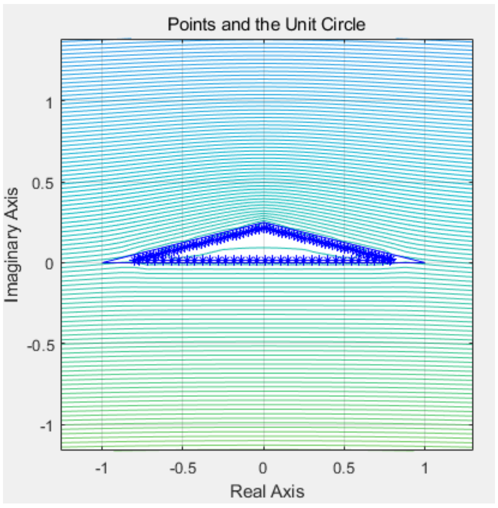


Figure 2.5: Factor = 0.8

We observe that a portion of the streamlines crosses the boundary. Next, we adjust the factor to see if it correlates with this parameter. We first try a factor of 0.6 and find that most streamlines are drawn inside the triangle, yet the boundary is still breached by the streamlines.

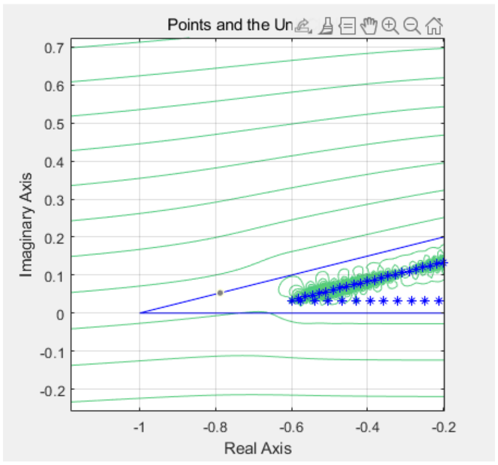


Figure 2.6: Factor = 0.6

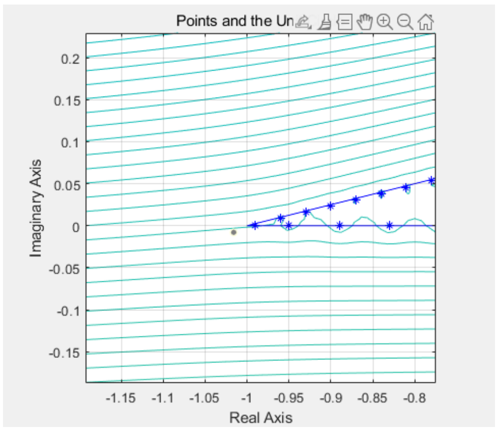


Figure 2.7: Factor = 0.99

We then attempt a factor of 0.99 to see if the same issue persists.

We discover that although the performance at a factor of 0.99 is much better than at 0.6, the problem remains. This indicates that the Finite Streamline Method (MFS) still struggles to handle singularities at corners effectively. Therefore, we consider incorporating new methods on top of MFS to address the singularities issue (Lightning Solvers).

Chapter 3

Lightning Solvers for Potential Flows

3.1 Lightning Solver introduction

This paper is concerned with the numerical solution of the planar Laplace equation

$$\nabla^2 f(z) = 0, \quad z \in D,$$

in contexts relevant to fluid dynamics. The domain D is assumed to be unbounded and simply connected (or periodic) and $z = x + iy$ is the spatial co-ordinate. Laplace's equation is sometimes considered the simplest two-dimensional (2-D) partial differential equation but, nevertheless, its numerical solution is challenging in many scenarios of practical interest. In particular, typical numerical methods struggle when the boundary of the flow domain ∂D is not smooth; for example, the solution of admits a singularity where the boundary has sharp corners, which hinders traditional techniques such as finite element methods and boundary element methods . Although these approaches have been successfully adapted to account for corners , their implementation is complex and requires expert knowledge. Recently, a new method that makes use of rational function approximation theory has been proposed for solving Laplace's equation in domains with corners . Dubbed the "lightning solver" (due to the analogy of lightning striking a corner because of a singular electric potential there), this new solution technique is extremely fast, accurate, and straightforward to implement. Herein, we use the lightning method to devise new strategies for solving potential flow problems in domains with corners.

Potential flows are foundational to fluid mechanics. In the fluid mechanics pedagogy, the first problem that students encounter is often incompressible and irrotational flow past a cylinder. Equipped with the solution to this simple flow, students progress to more complicated geometries via conformal mappings such as the Joukowski map . The ensuing solutions can usually be expressed in closed form and are, thus, highly interpretable. Much of classical aerodynamics was built on this approach and the associated solutions continue to be relevant to modern aerodynamics studies today . Idealised flow also commonly arises as an outer region problem in asymptotic analyses, and can then be matched to an inner boundary layer . Another way to improve the physical fidelity of potential flows is by incorporating the effects of flow separation. The Brown–Michael equation is a popular method for modelling point vortices shed from sharp corners whereas contour dynamics models the shedding and roll-up of vortex sheets . In Section 3.4, we shall see that the lightning approach of the present work can be used to rapidly compute point vortex trajectories in complicated domains. Free-streamline theory provides an alternative approach to modelling flow separation; we shall use the lightning solver to calculate the idealised, separated flow past a flat plate.

The mathematically tractable structure of potential flows has inspired a rich mathematical theory rooted predominantly in complex analysis. The theory of conformal mappings has enabled the study of potential flows in complicated domains beyond the aforementioned Joukowski map

Corners are obviously ubiquitous in real-life engineering applications and are often responsible for important physical behaviour; the local behaviour of flow past a sharp corner can have a significant impact on the global properties of the flow. For example, the Kutta condition implies that the flow at the sharp trailing edge of an aerofoil should depart the wing smoothly. In summary, the flow behaviour at corners must be accurately modelled for physically relevant results.

3.2 Method

Our research endeavors to decipher Laplace's equation in a two-dimensional, boundless domain characterized by angular boundaries. Solutions to Laplace's equation are noted for their singular behavior in proximity to these angular points, typically represented by the expression z^α. For instance, the complex potential for fluid flow around an infinitely sharp wedge, where the internal angle correlates to $\pi(2-\frac{1}{\alpha})$, can be depicted by this expression. The absence of viscous drag combined with the boundary's pronounced sharpness at these points implies potential extremities in flow velocity—ranging from infinite speeds to a complete standstill.

These unique singularities highlight the efficacy of the advanced 'lightning solvers', known for their precise and efficient computation of potential flows, especially in regions marked by such geometric irregularities.

The foundation of the lightning method is laid in the theory of rational function approximation. A rational function is defined as the quotient of two polynomial expressions, designated as type (m, n) when the degree of the numerator's polynomial is at most m and that of the denominator's is at most n. An eminent advancement in rational approximation theory was introduced by Newman, who asserted that the absolute value function $|x|$ could be accurately approximated over the interval $x \in [-1, 1]$ with a root-exponential rate of convergence. This signifies that for any given degree n, there exist constants A and C which facilitate the formulation of type (n, n) rational approximations r_n that closely approximate the behavior of $|x|$. $A, C > 0$

$$\max_{-1 \leq x \leq 1} ||x| - r_n(x)| \leq Ae^{-C\sqrt{n}}. \tag{3.1}$$

Benefit from Gopal and Trefethen 's research, they have established that analogously effective approximants can be constructed for a wider class of singularities, typified by the z^α form. They proved that employing rational functions for these more complex singularities can also lead to root-exponential rates of convergence. Specifically, for any given degree n, one can identify positive constants A and C, and construct type (n, n) rational approximants, denoted r_n, which adhere to this accelerated convergence criterion.

$$\max_{z \in H} |z^\alpha - r_n(z)| \leq A e^{-C\sqrt{n}} \tag{3.2}$$

where H represents the closed upper half of the unit disc. Cause we observe the rational approximants r_n possess poles that are exponentially clustered near zero with exponentially decreasing residues. So we take the poles which is clustered exponentially close to the corners of the domain in our Newman part and solve the accuracy problem on singularities. The MFS part is still looks like the previous part we do before.

$$f(z) = \underbrace{\sum_{j=1}^{n_1} \frac{a_j}{z - z_j}}_{\text{Newman part}} + \underbrace{\sum_{j=1}^{n_2} c_j * |z - z_k|}_{\text{MFS part}} \tag{3.3}$$

The poles z_j is selected in the line from center to singularities.The poles z_k is selected inside from the boundary to center by a zoom factor. The sets of complex coefficients $\{a_i\}$ for $i = 1, \ldots, n_1$ and $\{c_i\}$ for $i = 1, \ldots, n_2$, as the unknowns in our analysis. $f(z)$ is represented as a rational function in partial fraction decomposition, maintaining analyticity except at the pole locations.

A pivotal aspect of the lightning method is the arrangement of poles in the Newman portion, deliberately positioned near the corners to leverage the root-exponential convergence promised by equation (2.2). Indications are that these poles should exhibit a "tapered" distribution in the vicinity of the vertices, a concept further expounded in. In such instances, the poles clustered around a specific corner are expected to display a distribution contingent on their respective distances.

We use the factor $= 0.99$, so the pole(z_k) near to boundary looks like the boundary. And the inner pole points(z_j) are exponentially cluster to vertex.

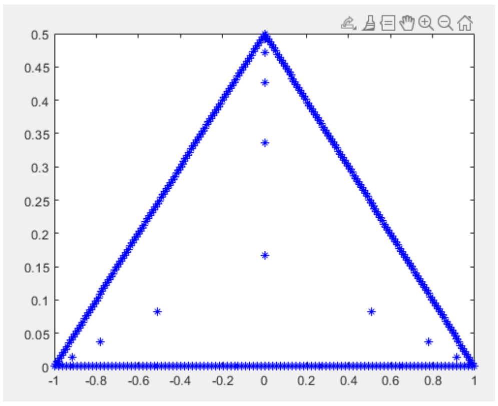


Figure 3.1: Triangle pole

For the linearity of Laplace's equation, we need to use least-squares method to fit a proper $a_j c_j$ to satisfy the boundary condition such that

$$\arg\min_{c} \|Ac - d\|_2$$

where $c \in \mathbb{R}^{n_1+n_2}$ is the value of the unknown coefficients a_j and c_j, $d \in \mathbb{R}^m$ represents the value of boundary condition on sample point, and $A \in \mathbb{R}^{m\times(n_1+n_2)}$ represents the sampled basis functions in $f(z)$. We use the boundary condition which is $\mathrm{Im}[f(z)] = d(z)$, for an $M \times 1$ vector of sample points Z, we have

$$A = \begin{bmatrix} \mathrm{Re}\left(\frac{1}{z_1-\hat{z}_1}\right) & \cdots & \mathrm{Re}\left(\frac{1}{z_1-\hat{z}_{n_1}}\right) & \log|z_1-\bar{z}_1| & \cdots & \log|z_1-\bar{z}_{n_2}| \\ \mathrm{Re}\left(\frac{1}{z_2-\hat{z}_1}\right) & \cdots & \mathrm{Re}\left(\frac{1}{z_2-\hat{z}_{n_1}}\right) & \log|z_2-\bar{z}_1| & \cdots & \log|z_2-\bar{z}_{n_2}| \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ \mathrm{Re}\left(\frac{1}{z_{m-1}-\hat{z}_1}\right) & \cdots & \mathrm{Re}\left(\frac{1}{z_{m-1}-\hat{z}_{n_1}}\right) & \log|z_{m-1}-\bar{z}_1| & \cdots & \log|z_{m-1}-\bar{z}_{n_2}| \\ \mathrm{Re}\left(\frac{1}{z_m-\hat{z}_1}\right) & \cdots & \mathrm{Re}\left(\frac{1}{z_m-\hat{z}_{n_1}}\right) & \log|z_m-\bar{z}_1| & \cdots & \log|z_m-\bar{z}_{n_2}| \end{bmatrix}$$

$$c = \begin{bmatrix} a_1 \\ a_2 \\ a_3 \\ \vdots \\ a_{n_1-1} \\ a_{n_1} \\ c_1 \\ c_2 \\ c_3 \\ \vdots \\ c_{n_2-1} \\ c_{n_2} \end{bmatrix} \qquad d = \begin{bmatrix} \mathrm{Im}(z_1) \\ \mathrm{Im}(z_2) \\ \mathrm{Im}(z_3) \\ \vdots \\ \mathrm{Im}(z_{m-1}) \\ \mathrm{Im}(z_m) \end{bmatrix}$$

Here $\hat{z}$ is the poles of Newmann part and $\bar{z}$ is the poles of MFS.

It is important to note that the matrix A is ill-conditioned. The regularization process is performed within the least squares framework. By adding a perturbation term to the diagonal elements of the matrix $A^T A$, where A is the matrix constructed from the sample points and A^T is its transpose, we ensure the system's stability and uniqueness of the solution. This regularization aims to find a set of coefficients that provide the best fit to a given condition, denoted by the vector b. The augmented matrix and vector, A_{aug} and d_{aug}, respectively, are used to compute the coefficients that minimize the residuals between the fitted condition and the actual condition A_0.

```
function [A_augmented, d_augmented] =
    augmentLeastSquaresMatrix(A, d, perturbation)
     A_transpose_A = A' * A;
     d_transpose = A' * d;

     A_augmented = A_transpose_A + perturbation * eye(size(
    A_transpose_A, 1));
     d_augmented = d_transpose;
end
```

And now we have the flow graph which is like Figure 2.2 and this new method is good at dealing with the area around corner.

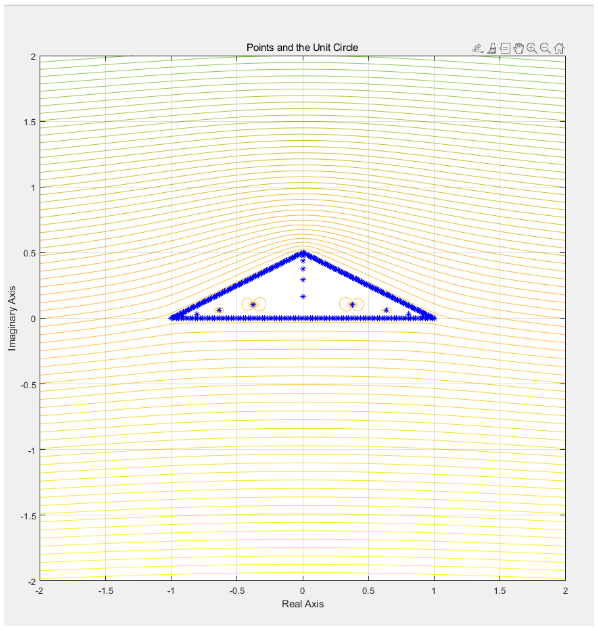


Figure 3.2: Lightning Triangle

And we could also see the details in corner show the correct graph following the fact.

Corner left:

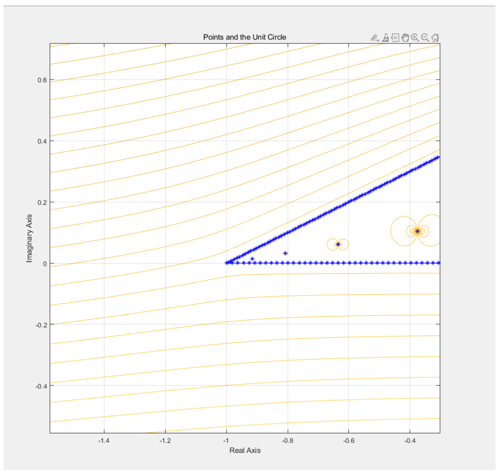


Figure 3.3: Left part

Corner mid:

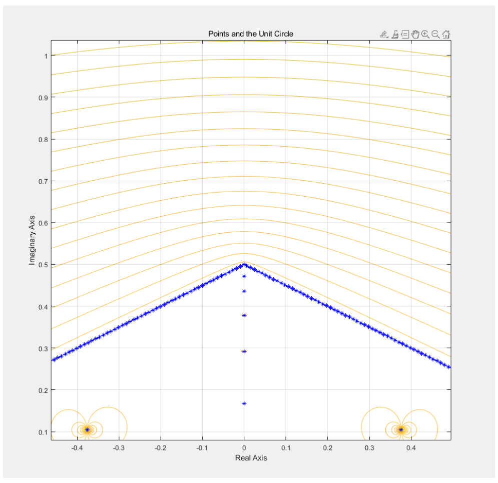


Figure 3.4: Mid part

Corner right:

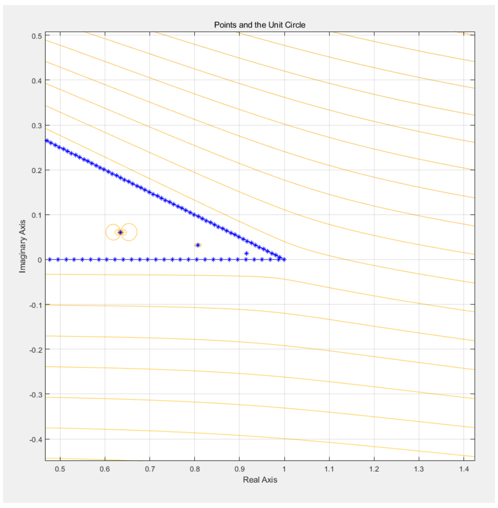


Figure 3.5: Right part

It is obviously that Lightning method plays a good role in corner and our combine of MFS and lightning make the work better than only MFS graph.

Chapter 4

Free Grid after Conformal mapping

4.1 Free Grid circle Joukowsky transform

We consider to use Joukowsky transform for circle as our next graph. Here is the mapping formula. z

$$z = \zeta + \frac{1}{\zeta}$$

where $z = x + iy$ is a complex variable in the new space and $\zeta = \chi + i\eta$ is a complex variable in the original space.

$$\begin{aligned} z = x + iy = \zeta + \frac{1}{\zeta} \\ = X + i\eta + \frac{1}{X + i\eta} \\ = X + i\eta + \frac{X - i\eta}{X^2 + \eta^2} \\ = X\left(1 + \frac{1}{X^2 + \eta^2}\right) + i\eta\left(1 - \frac{1}{X^2 + \eta^2}\right). \end{aligned}$$

So the real x and imaginary y components are:

$$x = X\left(1 + \frac{1}{X^2 + \eta^2}\right),$$

$$y = \eta\left(1 - \frac{1}{X^2 + \eta^2}\right).$$

And now we try a new module Symmetrical Joukowsky airfoils for suit a more complex graph.

In 1943 Hsue-shen Tsien published a transform of a circle of radius a into a symmetrical airfoil that depends on parameter ε and angle of inclination α:

$$z = e^{i\alpha}\left(\zeta - \varepsilon + \frac{1}{\zeta - \varepsilon} + \frac{2\varepsilon^2}{a+\varepsilon}\right).$$

The parameter ε yields a flat plate when zero, and a circle when infinite; thus it corresponds to the thickness of the airfoil. Furthermore, the radius of the cylinder $a = 1 + \varepsilon$.

Both of a and ε are constant, so we remove the third part

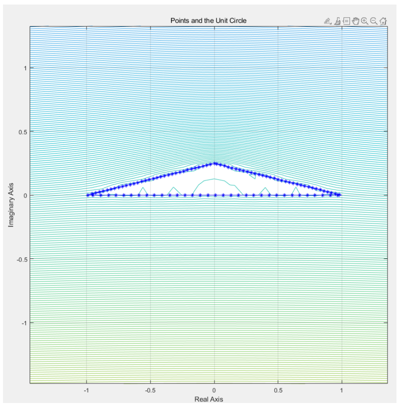
$$z = e^{i\alpha}\left(\zeta - \varepsilon + \frac{1}{\zeta - \varepsilon}\right)$$

Now we try to change the variable and see what influence would be in our graph.(Basic on (Sample) $n = 200, \epsilon = 0.1, factor = \frac{4}{5}$). And factor is the reduction factor from sample point.(xPole = real(samplepoint) * factor, yPole = imag(samplepoint) * factor)

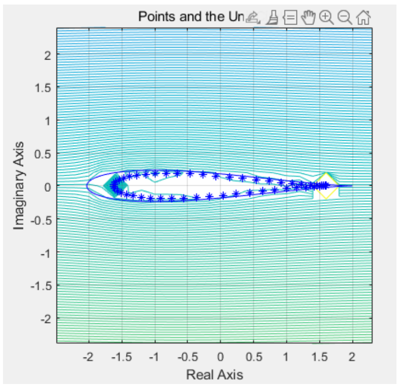
Change number of Sample point(n)

Standard variable :

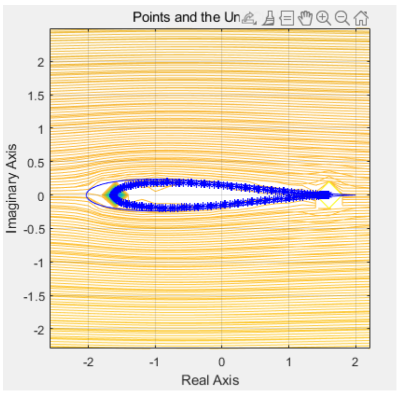
sample point = 200 , $\varepsilon = 0.1$, factor = 0.8



sample point = 100 , $\varepsilon = 0.1$, factor = 0.8



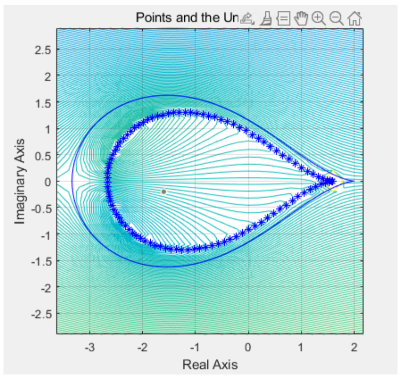
sample point = 300 , $\varepsilon = 0.1$, factor = 0.8



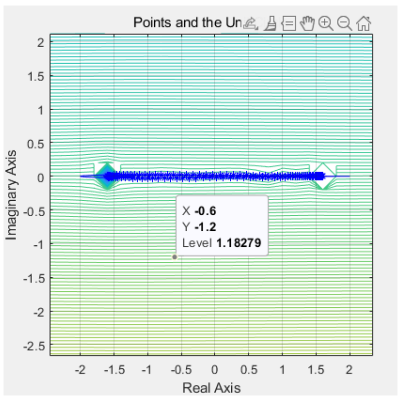
Conclusion: The difference is small for big enough sample point

Change ε

sample point $= 200$, $\varepsilon = 1$, factor $= 0.8$



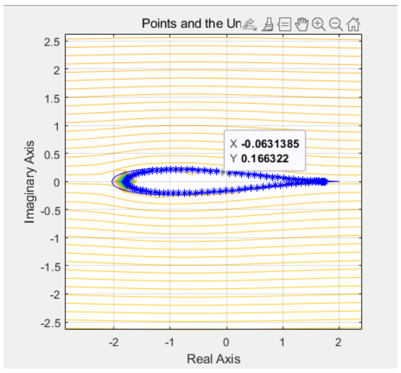
sample point $= 200$, $\varepsilon = 0.01$, factor $= 0.8$ (The mapping is too small and looks like a line)



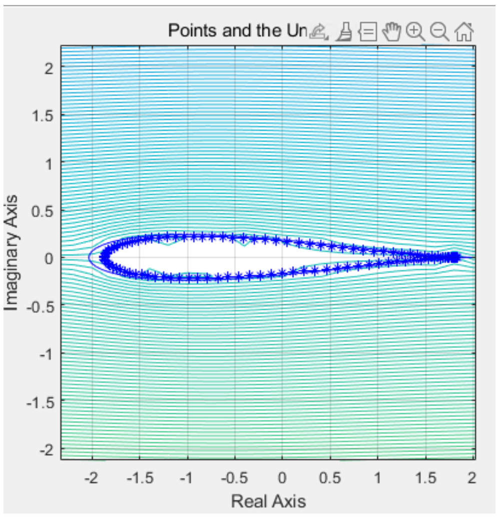
Conclusion: It is better for bigger ε (depends on shape we need)

Change factor

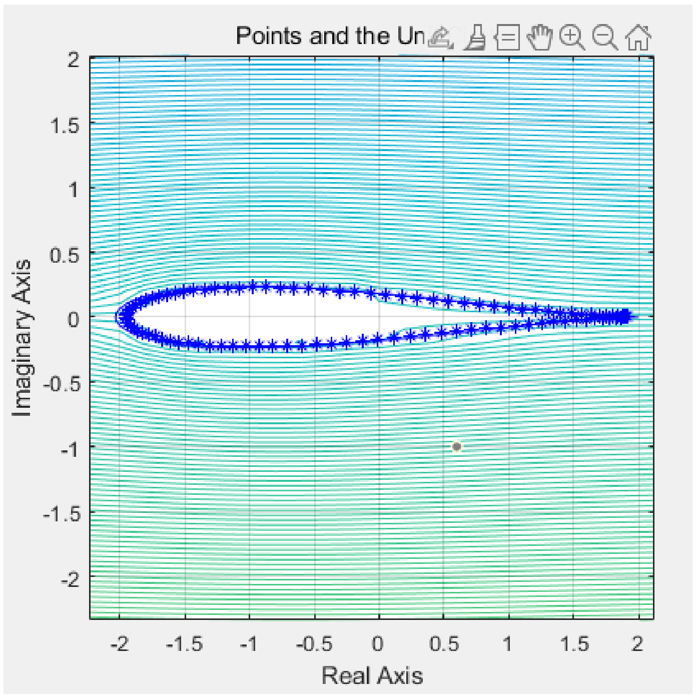
sample point $= 200$, $\varepsilon = 0.1$, factor $= \frac{22}{25}$



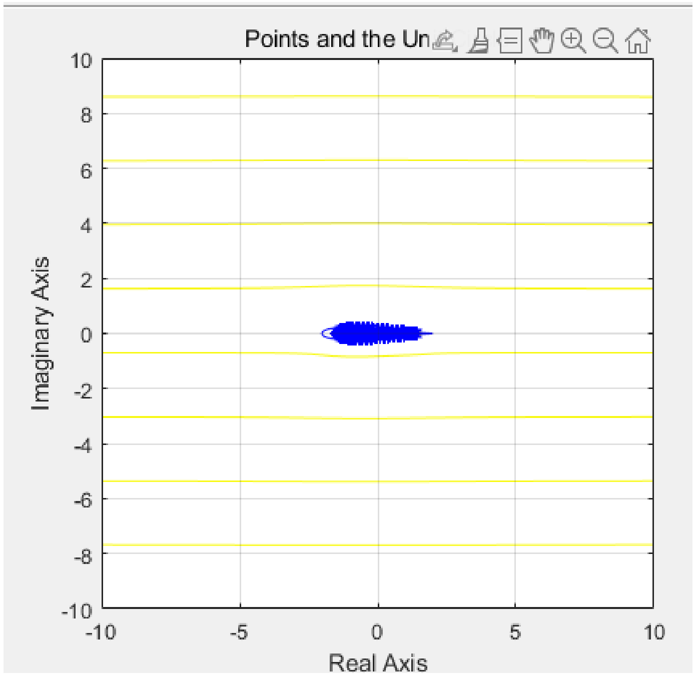
sample point $= 200$, $\varepsilon = 0.1$, factor $= \frac{23}{25}$



sample point $= 200$, $\varepsilon = 0.1$, factor $= \frac{24}{25}$



sample point $= 200$, $\varepsilon = 0.1$, factor $= \frac{18}{25}$



We observe that for different values of n, ε, and *factor*, once n exceeds a certain threshold, further increases do not significantly enhance the image. For ε, the closer it is to 1, the more precise the representation, or in other words, the less it is affected by singularities. Similarly, for *factor*, values approaching 1 yield more accurate boundary treatments, yet the influence of singular points remains apparent.

In all the graphs produced, it is evident that there are significant deviations at the far left and right edges, suggesting that our standard MFS cannot fully address the flow lines at singular points. Therefore, we propose a composite method that employs the MFS and Lightning method previously applied to triangles to address the issues with achieving a satisfactory fit for symmetrical airfoils.

4.2 Lightning solver on mapping

In the application of the lightning solver method, it is imperative to incorporate the pole points of Neumann part. Specifically, we address the singularities situated at the extremities of the domain: the rightmost and leftmost sample points. So we introduce

$$A = [\max(Real(\text{samplepoint})), 0]$$
$$B = [\min(Real(\text{samplepoint})), 0]$$

as the points of interest. Subsequently, we compute the centroid of all sample points to serve as the central node, given by

$$\text{center} = [Real(Mean(\text{samplepoint}), Imag(Mean(\text{samplepoint})],$$

where Mean(samplepoint) denotes the mean of the sample points.

For these graph, we apply n = 600 , $\varepsilon = 0.1$, Newmann pole points = 30 , MFS pole = 200, factor = 0.98. So total pole points = 200 + 30 = 230

$$A \in \mathbb{R}^{600\times 230}, \quad c \in \mathbb{R}^{600\times 1} \quad \text{and} \quad d \in \mathbb{R}^{230\times 1}$$

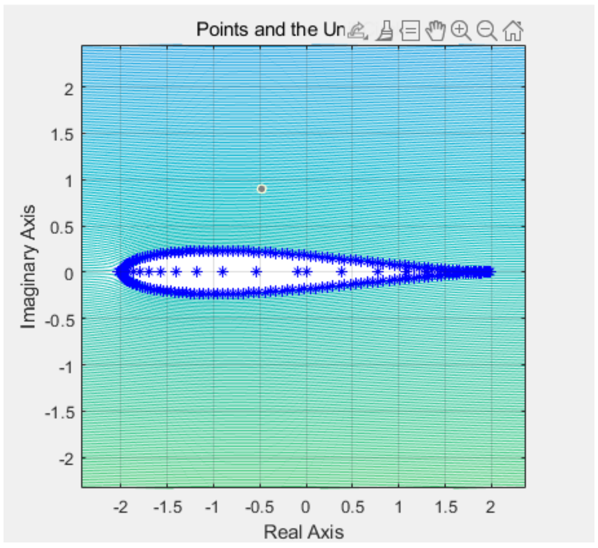


Figure 4.1: Overview Graph

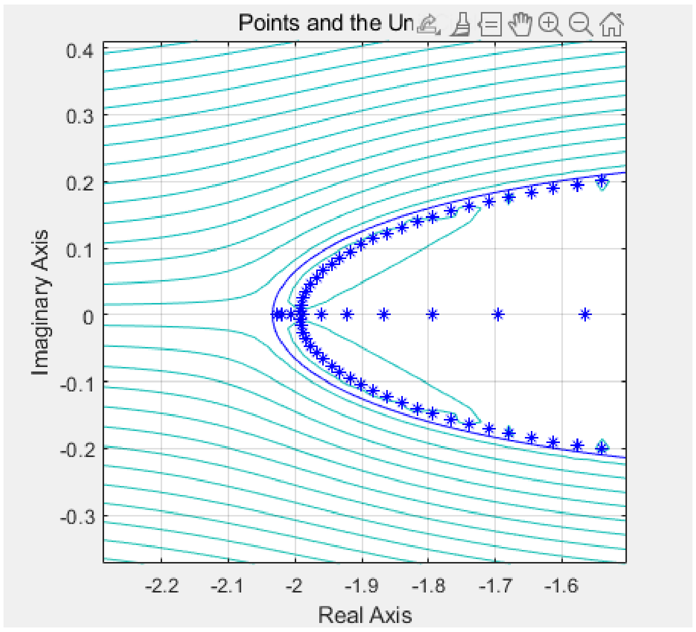


Figure 4.2: Left part

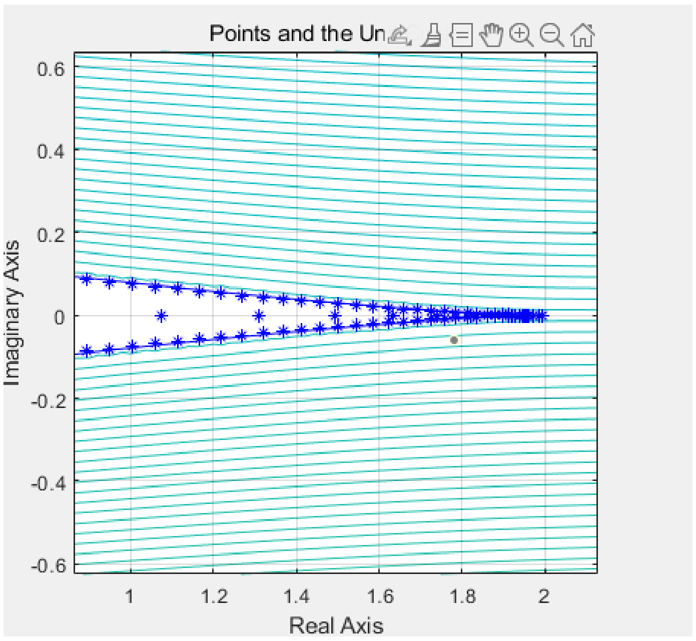


Figure 4.3: Right part

From the analysis of the three figures, it is evident that within the vicinity of the singular points on the left and right, no streamline crosses the boundary. This observation indicates that our method appropriately and effectively fits the coefficients a_j and c_j at the singularities. However, it is also crucial to recognize that fluid simulation cannot rely solely on the x-direction. Our approach should be adaptable to fluids approaching from various angles. The following discussion will elucidate the implementation process for fluids moving in different directions.

4.3 Angle change for Joukowsky transform

Now let us get back to the definition of stream function ψ and potential function ϕ.

$$\psi = y + \phi$$

$$\nabla^2\psi = 0, \quad \nabla^2\phi = 0$$

With $\psi = 0$ at the boundary, we have $\phi = -y$.

$$\begin{aligned}
u + iv &= Ue^{i\alpha} \\
u - iv &= Ue^{-i\alpha} \\
w &= Ue^{-i\alpha} \\
w &= U\left[\cos\alpha - i\sin\alpha\right]\left[x + iy\right] \\
&= U\left[\cos\alpha x + y\sin\alpha\right] \\
-iU\left[x\sin\alpha - y\cos\alpha\right] &\leftarrow \psi
\end{aligned}$$

Now we could get the formula of ψ for angle α

$$\psi = y\cos\alpha - x\sin\alpha$$

$$\begin{aligned}
\psi &= y && \text{for } \alpha = 0, \\
\psi &= \frac{y}{\sqrt{2}} - \frac{x}{\sqrt{2}} && \text{for } \alpha = \frac{\pi}{4}.
\end{aligned}$$

We need to apply this on our conditon d and on contour matrix we build.

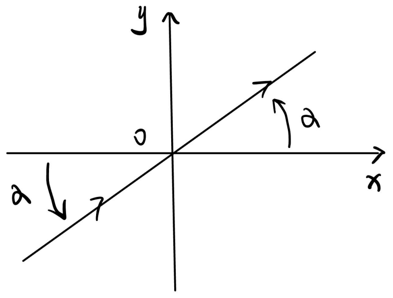


Figure 4.4: Angle change α

$\alpha = \frac{\pi}{4}$: (the parameter is same as before)

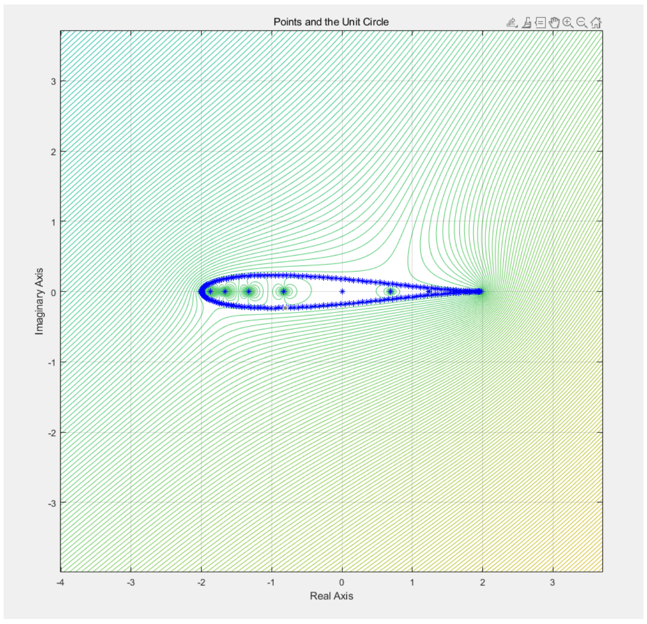


Figure 4.5: $\alpha = \frac{\pi}{4}$ Overview graph

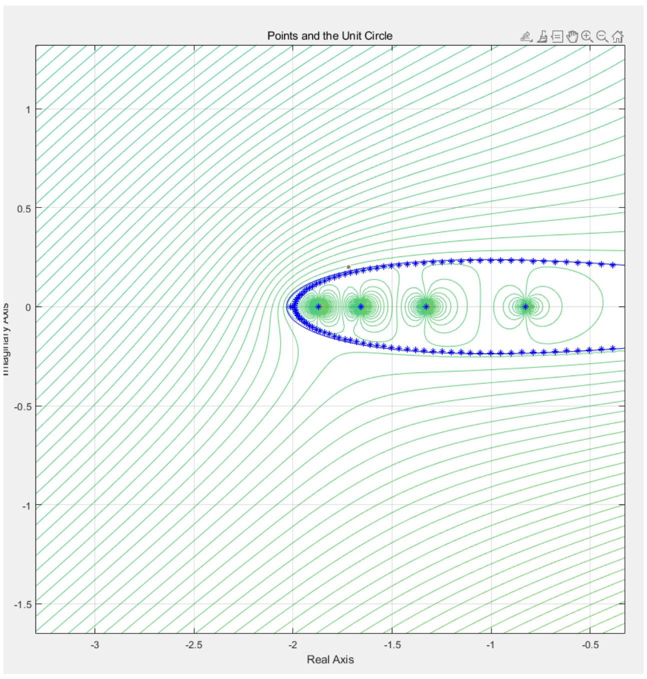


Figure 4.6: Left part

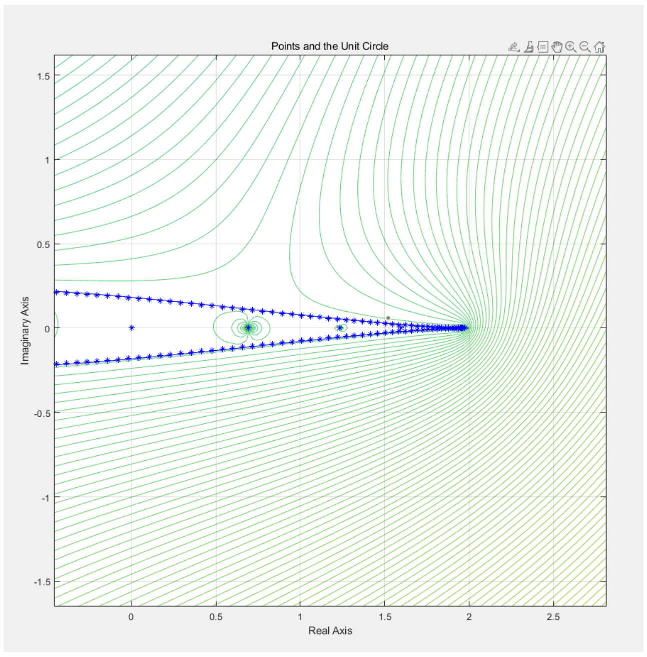


Figure 4.7: Right part

$\alpha = \frac{\pi}{6}$:

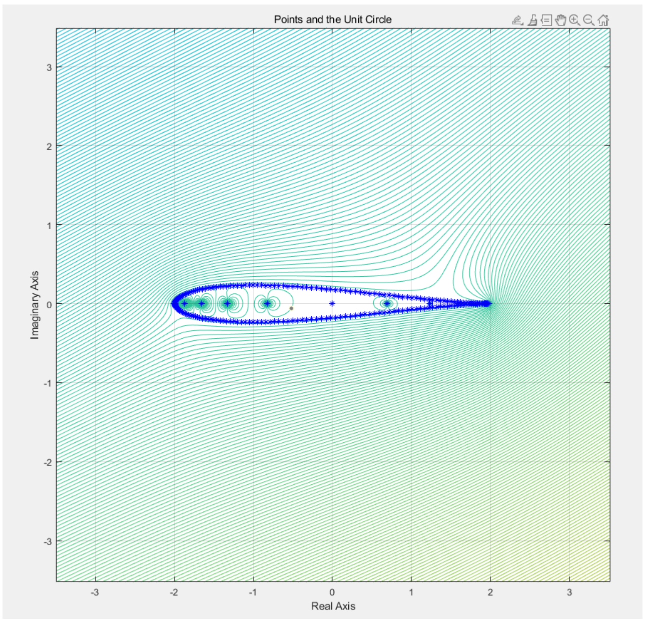


Figure 4.8: $\alpha = \frac{\pi}{6}$ Overview graph

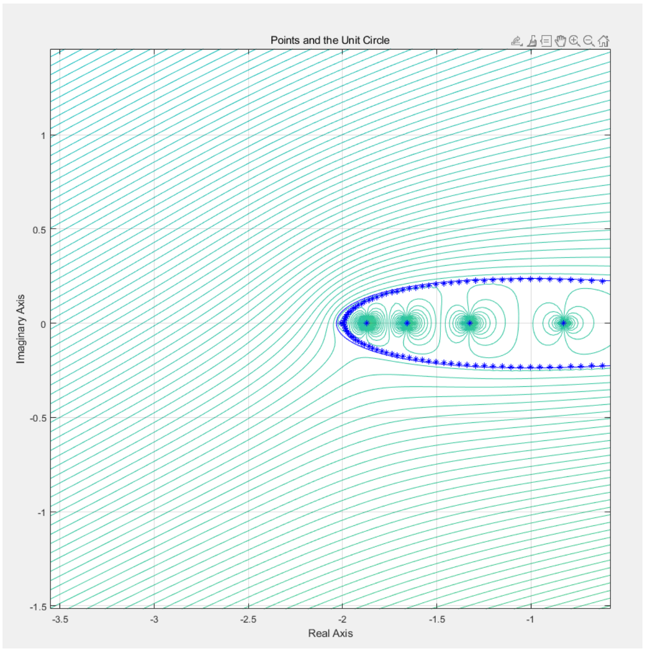


Figure 4.9: Left part

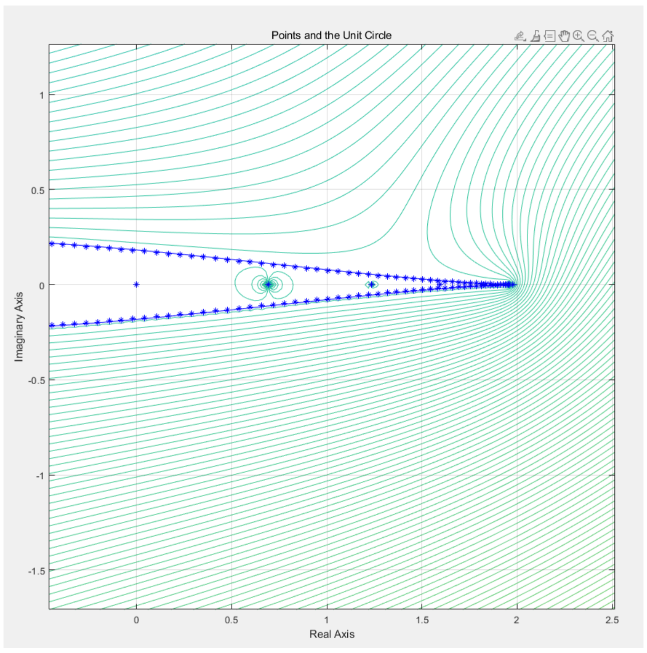


Figure 4.10: Right part

After we apply the angle on our condition. We find that the graph shows proper accuracy on the singularities parts.(for $\alpha \in (0, 2\pi])$

4.4 The effect of pole number chosen

That looks correct in Samplepoint = 600 , pole = 24. Now we try Sam-

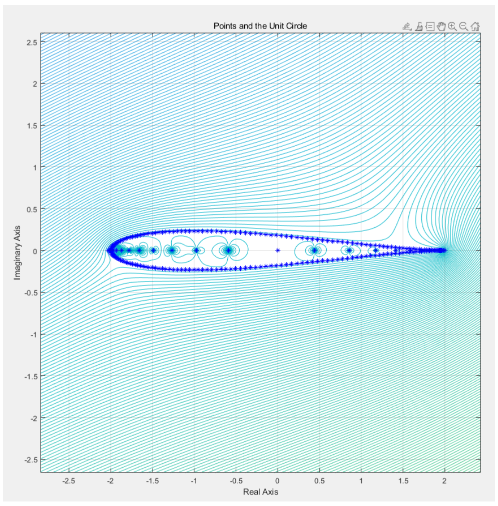


Figure 4.11: Samplepoint = 600 , pole = 24

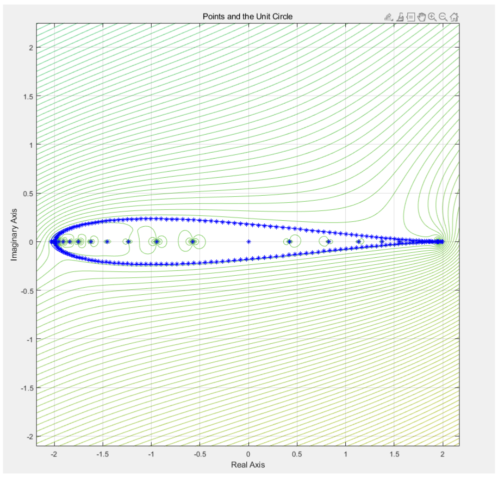


Figure 4.12: Samplepoint = 600 , pole = 26

plepoint = 600 , pole = 26. The problem seems happen on the tails of the shape. But it seems still fit. Next we try Samplepoint = 600 , pole = 28 It is obviously that a lot of streamline print in the shape and it is

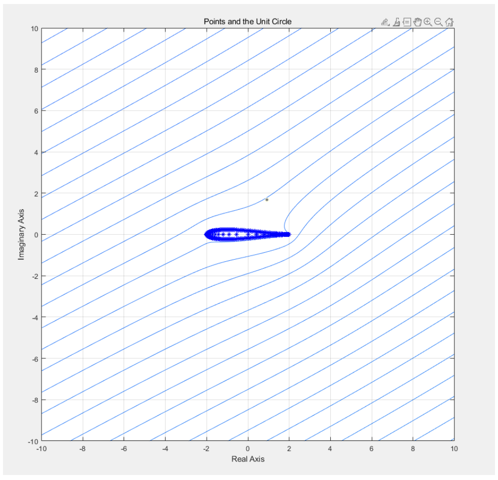


Figure 4.13: Samplepoint = 600 , pole = 28

also mean that our a_j and c_j do not have a nice fit on boundary.

So we find that it would be better for fitting in Lightning method when

$$\frac{numpole}{numsamplepoint} \leq \frac{24}{600} = 0.04$$

Basic on number of pole in MFS is one third of sample point.

Chapter 5

Conclusion

Within the context of the Finite Strip Method (MFS), there is no requirement for a grid on the graphical representation, enhancing efficiency, particularly in regions where the solution exhibits smooth characteristics. The MFS also yields an analytically differentiable expression. However, a notable limitation of this method is its reduced precision at corners where is not smooth.

So we need one more method (Lightning solver) to assist us on these area which is using extra poles (Newman points) exponentially approach to these corner. Moreover, we replace Runge part in lightning solver by MFS which arrive a better approximation and more efficient in smooth part. The matrix determine the coefficient is VanderMonde known to give ill-condition system but with Arnoldi algorithm they can be compensated good. (Baboo[6])

Finally, we get the airfoil shape after conformal mapping from circle and try to solve this with Kutta condition

Bibliography

1. M. Katsurada and U. Okamoto. *The Collocation Points of the Fundamental Solution Method for the Potential Problem.* Computers Math. Applic. Vol. 31, No. 1, pp. 123–137, 1996

2. R. L. Johnston and G. Fairweather. *The method of fundamental solutions for problems in potential flow.* Appl. Math. Modelling, 1984, Vol. 8, August

3. A. Gopal a and L. N. Trefethen a,1. *New Laplace and Helmholtz solvers.* Edited by David L. Donoho, Stanford University, Stanford, CA, and approved April 18, 2019 (received for review March 11, 2019)

4. A. H. Barnett and T. Betcke. *Stability and convergence of the method of fundamental solutions for Helmholtz problems on analytic domains.* Received 26 August 2007; received in revised form 21 March 2008; accepted 3 April 2008 Available online 20 April 2008

5. L. N. Trefethen. *Series Solution of Laplace Problems.* Received 3 March, 2018; accepted 10 April, 2018; first published online 6 July 2018

6. P. J. Baddoo. *Lightning Solvers for Potential Flows.* Received: 9 October 2020; Accepted: 20 November 2020; Published: 30 November 2020

7. H - S Tsien. *Symmetrical Joukowski airfoils in shear flow.* D.Reidel. Received Dec. 27, 1942.

8. R. Mathon and R. L. Johnston. *The approximate solution of elliptic boundary-value problems by fundamental solutions.* SIAM J. Numer. Anal., Vol. 14, No. 4, September 1977.

Appendix A

1.Circle before mapping

*

```
num_points = 200;
n = 2;
factor = 0.8;

angles = linspace(0, 2*pi, num_points);

samplepoint = exp(1i * angles);

x = real(samplepoint);
y = imag(samplepoint);

pole_x = zeros(num_points / 2, 1);
pole_y = zeros(num_points / 2, 1);

pole_number = length(pole_x);

pole_x1 = x * factor;
pole_y1 = y * factor;

x = -real(samplepoint);
y = imag(samplepoint);

for i = 1:(num_points/2)
    pole_x(i) = pole_x1(2*i-1);
end
for i = 1:(num_points/2)
    pole_y(i) = pole_y1(2*i-1);
end
```

```

pole = pole_x +  pole_y*1i ;

pole_x = real(pole);
pole_y = imag(pole);

number_pole = length(pole_x);

condition = y.';


A = buildLeastSquaresMatrix(samplepoint,pole,
    number_pole);

coefficients = A\condition ;

z = Findfz((0+2*i),coefficients,pole,pole_number)
    ;
disp(z)

x_inter = linspace(-5, 5, 501);
y_inter = linspace(5, -5, 501);
linspace_matrix = zeros(length(x_inter),length(
    y_inter));

[X, Y] = meshgrid(x_inter, y_inter);

complex_grid = X + 1i * Y;


for i = 1:length(y_inter)
    for j = 1:length(x_inter)
        linspace_matrix(i,j) = Findfz(
    complex_grid(i,j),coefficients,pole,pole_number
    );
    end
end

y_r = repmat(y_inter',1,length(x_inter));
psi = linspace_matrix;

contour(x_inter, y_inter, -y_r+psi, 200);
```

```

title('flow graph');
xlabel('x');
ylabel('y');
axis equal;
hold on

plot(real(pole), imag(pole), 'b*');

hold on

plot(real(samplepoint), imag(samplepoint), 'b-');

axis equal;
grid on;
title('Points and the Unit Circle');
xlabel('Real Axis');
ylabel('Imaginary Axis');

figure;


[minValue, minIndex] = min(linspace_matrix(:));


[maxValue, maxIndex] = max(linspace_matrix(:));


function A = buildLeastSquaresMatrix(
   sample_points, poles, n1)
    num_samples = length(sample_points);
    A = zeros(num_samples, n1);

    for i = 1:num_samples
        z = sample_points(i);
        for j = 1:n1
            A(i, j) = log(abs(z - poles(j)));
        end
    end
end

function B = Findfz(point,coeffients,pole,n)
```

```
    C = zeros(1,n);
    for i = 1:n
        C(1,i) = log(abs(point-pole(i)));
    end
    B = C * coeffients;
end
```

Listing A.1: Matlab Code

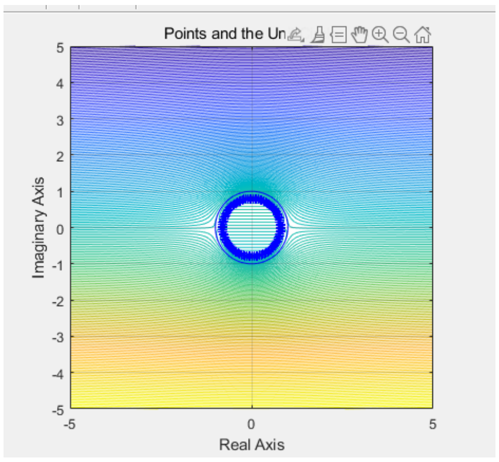


Figure A.1: Circle before mapping

Appendix B

2.Triangle MFS

*

```
num_points = 300;

num_points_per_side = num_points / 3;

A = 1 + 0i;
B = -1 + 0i;
C = 0 - 1/4i;
center = (A+B+C)/3;

x_AB = linspace(real(A), real(B),
   num_points_per_side);
y_AB = linspace(imag(A), imag(B),
   num_points_per_side);
x_BC = linspace(real(B), real(C),
   num_points_per_side);
y_BC = linspace(imag(B), imag(C),
   num_points_per_side);
x_CA = linspace(real(C), real(A),
   num_points_per_side);
y_CA = linspace(imag(C), imag(A),
   num_points_per_side);

x = [x_AB, x_BC(1:end), x_CA(1:end)];
y = [y_AB, y_BC(1:end), y_CA(1:end)];

samplepoint = x + 1i*y;

pole_x = zeros(num_points, 1);
```

```
pole_y = zeros(num_points, 1);

for i = 1:(num_points)
    scaled_point = center + (samplepoint(i) - center) * 0.99;
    pole_x(i) = real(scaled_point);
    pole_y(i) = imag(scaled_point);
end

pole = pole_x + 1i * pole_y;

x = real(samplepoint);
y = imag(samplepoint);
pole_x1 = zeros(num_points/3, 1);
pole_y1 = zeros(num_points/3, 1);

for i = 1:(num_points/3)
    pole_x1(i) = pole_x(3*i-1);
end
for i = 1:(num_points/3)
    pole_y1(i) = pole_y(3*i-1);
end

pole =  pole_y1*1i + pole_x1;

pole_x = real(pole);
pole_y = imag(pole);

number_pole = length(pole_x1);

condition = y.';


A = buildLeastSquaresMatrix(samplepoint,pole,number_pole);

coefficients = A\condition ;

z = Findfz((0+2*i),coefficients,pole,number_pole);

disp(z)
```

```

x_inter = linspace(-4, 4, 101);
y_inter = linspace(4, -4, 101);
linspace_matrix = zeros(length(x_inter),length(
   y_inter));


[X, Y] = meshgrid(x_inter, y_inter);

complex_grid = X + 1i * Y;


for i = 1:length(y_inter)
    for j = 1:length(x_inter)
        linspace_matrix(i,j) = Findfz(
   complex_grid(i,j),coefficients,pole,number_pole
   );
    end
end

y_r = repmat(y_inter',1,length(x_inter));
psi = linspace_matrix;

contour(x_inter, y_inter, -y_r+psi, 500);

title('flow graph');
xlabel('x');
ylabel('y');
axis equal;
hold on

plot(real(pole), imag(pole), 'b*');

hold on

plot(real(samplepoint), imag(samplepoint), 'b-');

axis equal;
grid on;
title('Points and the Unit Circle');
xlabel('Real Axis');
ylabel('Imaginary Axis');
```

```

figure;



function A = buildLeastSquaresMatrix(sample_points, poles, n1)
    num_samples = length(sample_points);
    A = zeros(num_samples, n1);

    for i = 1:num_samples
        z = sample_points(i);
        for j = 1:n1
            A(i, j) = log(abs(z - poles(j)));
        end
    end
end

function B = Findfz(point,coeffients,pole,n)
    C = zeros(1,n);
    for i = 1:n
        C(1,i) = log(abs(point-pole(i)));
    end
    B = C * coeffients;
end
```

Listing B.1: Matlab Code

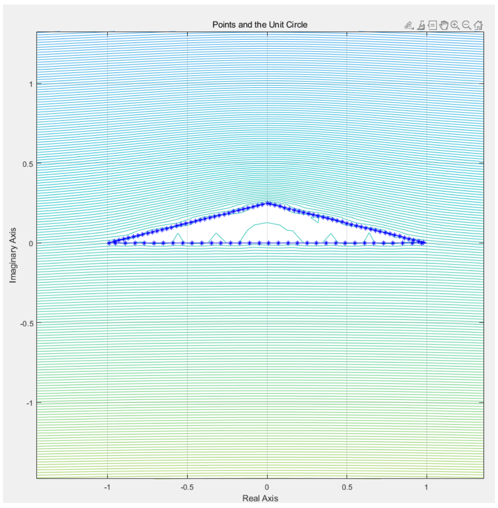


Figure B.1: Triangle MFS

Appendix C

Triangle MFS and Lightning

*

```matlab
A = [1, 0];
B = [-1, 0];
C = [0, 1/2];
center = (A+B+C)/3;            % Newman part
num_points1 = 15;
factor = 0.999;
sigma = 2;
num_points_per_side = num_points1/3;

distances = zeros(1, num_points_per_side);

for j = num_points_per_side:-1:1
    distances(num_points_per_side - j + 1) = exp
   (-sigma * (sqrt(num_points_per_side) - sqrt(j))
   );
end

pole1 = zeros(num_points_per_side, 2);
pole2 = zeros(num_points_per_side, 2);
pole3 = zeros(num_points_per_side, 2);
for i = 1:num_points_per_side
    pole1(i, :) = center + (A - center) * (1-
   distances(i));
    pole2(i, :) = center + (B - center) * (1-
   distances(i));
    pole3(i, :) = center + (C - center) * (1-
   distances(i));
end
```

```
list_all = [pole1; pole2; pole3];
x = list_all(:, 1);
y = list_all(:, 2);

pole11 = x + y*1i;



num_points2 = 990;       % MFS part, number of this part is num_point/3   must =0 (mod9)

num_points_per_side = num_points2 / 3;

A = 1 + 0i;
B = -1 + 0i;
C = 0 - 1/2i;
center = (A+B+C)/3;

x_AB = linspace(real(A), real(B), num_points_per_side);
y_AB = linspace(imag(A), imag(B), num_points_per_side);
x_BC = linspace(real(B), real(C), num_points_per_side);
y_BC = linspace(imag(B), imag(C), num_points_per_side);
x_CA = linspace(real(C), real(A), num_points_per_side);
y_CA = linspace(imag(C), imag(A), num_points_per_side);

x = [x_AB, x_BC(1:end), x_CA(1:end)];
y = [y_AB, y_BC(1:end), y_CA(1:end)];

samplepoint = x + 1i*y;

pole_x = zeros(num_points2, 1);
pole_y = zeros(num_points2, 1);

for i = 1:(num_points2)
    scaled_point = center + (samplepoint(i) - center) * factor;
    pole_x(i) = real(scaled_point);
    pole_y(i) = imag(scaled_point);
end

pole = pole_x + 1i * pole_y;

x = real(samplepoint);
y = imag(samplepoint);
pole_x1 = zeros(num_points2/3, 1);
pole_y1 = zeros(num_points2/3, 1);

for i = 1:(num_points2/3)
    pole_x1(i) = pole_x(3*i);
end
for i = 1:(num_points2/3)
    pole_y1(i) = pole_y(3*i);
end

pole =  pole_y1*1i + pole_x1;

polenew = [pole11 ; pole];
pole = polenew;

pole_x = real(pole);
pole_y = imag(pole);

number_pole = length(pole);
number_poles2 = num_points2/3;

condition = y.';

A = buildLeastSquaresMatrix(samplepoint,pole,
   num_points1,number_poles2);

perturbation = 1e-8;
[A_augmented, b_augmented] =
   augmentLeastSquaresMatrix(A, condition,
   perturbation);
coefficients = A_augmented \ b_augmented;

z = Findfz((0+2*i),coefficients,pole,num_points1,
   number_poles2);
```

```

disp(z)

x_inter = linspace(-2, 2, 1001);
y_inter = linspace(2, -2, 1001);
linspace_matrix = zeros(length(x_inter),length(
   y_inter));


[X, Y] = meshgrid(x_inter, y_inter);

complex_grid = X + 1i * Y;


for i = 1:length(y_inter)
    for j = 1:length(x_inter)
        linspace_matrix(i,j) = Findfz(
   complex_grid(i,j),coefficients,pole,num_points1
   ,number_poles2);
    end
end

y_r = repmat(y_inter',1,length(x_inter));
psi = linspace_matrix;

contour(x_inter, y_inter, -y_r+psi, 200);

title('flow graph');
xlabel('x');
ylabel('y');
axis equal;
hold on

plot(real(pole), imag(pole), 'b*');

hold on

plot(real(samplepoint), imag(samplepoint), 'b-');

axis equal;
grid on;
title('Points and the Unit Circle');
```

```matlab
xlabel('Real Axis');
ylabel('Imaginary Axis');

figure;



function A = buildLeastSquaresMatrix(sample_points, poles, n1, n2)
    num_samples = length(sample_points);
    A = zeros(num_samples, n1 + n2);

    for i = 1:num_samples
        z = sample_points(i);
        for j = 1:n1
            A(i, j) = real(1 / (z - poles(j)));
        end
        for k = (n1+1):(n1+n2)
            A(i, k) = log(abs(z - poles(k)));
        end
    end
end

function [A_augmented, d_augmented] = augmentLeastSquaresMatrix(A, d, perturbation)
    A_transpose_A = A' * A;
    d_transpose = A' * d;

    A_augmented = A_transpose_A + perturbation * eye(size(A_transpose_A, 1));
    d_augmented = d_transpose;
end

function B = Findfz(point,coeffients,pole,n1,n2)
    C = zeros(1,n1+n2);
    for i = 1:n1
        C(1,i) = real(1/(point-pole(i)));
    end
    for j = n1+1:n1+n2
        C(1,j) = log(abs(point-pole(j)));
    end
    B = C * coeffients;
```

```
end
```

Listing C.1: Matlab Code

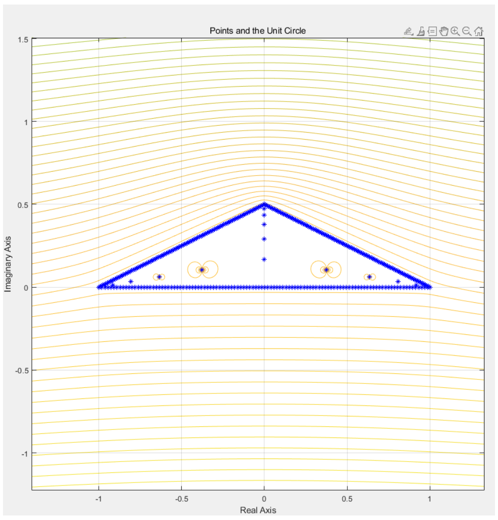


Figure C.1: Triangle MFS and Lightning

Appendix D

Circle After mapping in MFS

*

```
% Finish the shape after transform


num_points = 600;

n = 2;
b = 1.1;
elsi = 0.1;
factor = 49/50;
angles = linspace(0, 2*pi, num_points);

samplepoint =  (1+elsi)*exp(1i * angles);


pole_x = zeros(num_points / 2, 1);
pole_y = zeros(num_points / 2, 1);

pole_number = length(pole_x);

newsamplepoint = (samplepoint - elsi)  + 1 ./(
   samplepoint - elsi) ;
samplepoint = newsamplepoint;

x = real(samplepoint);
y = imag(samplepoint);

pole_x1 = x * factor;
pole_y1 = y * factor;
```

```
for i = 1:(num_points/2)
    pole_x(i) = pole_x1(2*i-1);
end
for i = 1:(num_points/2)
    pole_y(i) = pole_y1(2*i-1);
end

pole =  pole_y*1i + pole_x;

pole_x = real(pole);
pole_y = imag(pole);

number_pole = length(pole_x);

condition = y.';


A = buildLeastSquaresMatrix(samplepoint,pole,
   number_pole);

coefficients = A\condition ;

z = Findfz((0+2*i),coefficients,pole,pole_number)
   ;
disp(z)


x_inter = linspace(-10, 10, 101);
y_inter = linspace(10, -10, 101);
linspace_matrix = zeros(length(x_inter),length(
   y_inter));


[X, Y] = meshgrid(x_inter, y_inter);

complex_grid = X + 1i * Y;


for i = 1:length(y_inter)
    for j = 1:length(x_inter)
```

```
        linspace_matrix(i,j) = Findfz(
    complex_grid(i,j),coefficients,pole,pole_number
    );
    end
end

y_r = repmat(y_inter',1,length(x_inter));
psi = linspace_matrix;

contour(x_inter, y_inter, -y_r+psi, 500);

title('flow graph');
xlabel('x');
ylabel('y');
axis equal;
hold on

plot(real(pole), imag(pole), 'b*');

hold on

plot(real(samplepoint), imag(samplepoint), 'b-');

axis equal;
grid on;
title('Points and the Unit Circle');
xlabel('Real Axis');
ylabel('Imaginary Axis');

figure;



function A = buildLeastSquaresMatrix(
    sample_points, poles, n1)
    num_samples = length(sample_points);
    A = zeros(num_samples, n1);

    for i = 1:num_samples
        z = sample_points(i);
        for j = 1:n1
            A(i, j) = log(abs(z - poles(j)));
        end
    end
end

function B = Findfz(point,coeffients,pole,n)
    C = zeros(1,n);
    for i = 1:n
        C(1,i) = log(abs(point-pole(i)));
    end
    B = C * coeffients;
end
```

Listing D.1: Matlab Code

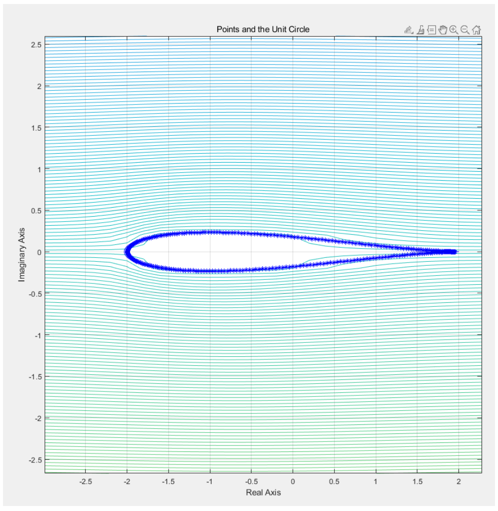


Figure D.1: Circle After mapping in MFS

Appendix E

Airfoil in MFS and Lightning

*

```
num_points1 = 30;
sigma = 2;

num_points_per_side = num_points1/2;

distances = zeros(1, num_points_per_side);

for j = num_points_per_side:-1:1
    distances(num_points_per_side - j + 1) = exp
   (-sigma * (sqrt(num_points_per_side) - sqrt(j))
   );
end

pole1 = zeros(num_points_per_side, 2);
pole2 = zeros(num_points_per_side, 2);

num_points = 600;

n = 2;
b = 1.1;
elsi = 0.1;
factor = 49/50;
angles = linspace(0, 2*pi, num_points);

samplepoint =  (1+elsi)*exp(1i * angles);

newsamplepoint = (samplepoint - elsi)  + 1 ./(
   samplepoint - elsi) ;
samplepoint = newsamplepoint;

A = [max(real(samplepoint)),0];
B = [min(real(samplepoint)),0];

x = real(samplepoint);
y = imag(samplepoint);

pole_x1 = x * factor;
pole_y1 = y * factor;

meansample = mean(samplepoint);

center = [real(meansample),imag(meansample)];

for i = 1:num_points_per_side
    pole1(i, :) = center + (A - center) * (1-
   distances(i));
    pole2(i, :) = center + (B - center) * (1-
   distances(i));
end

list_all = [pole1; pole2];
x1 = list_all(:, 1);
y1 = list_all(:, 2);

pole11 = x1 + y1*1i;

pole_x = zeros(num_points / 3, 1);
pole_y = zeros(num_points / 3, 1);

for i = 1:(num_points/3)
    pole_x(i) = pole_x1(3*i-1);
end

for i = 1:(num_points/3)
    pole_y(i) = pole_y1(3*i-1);
end


pole =  pole_y*1i + pole_x;

```

```
number_pole = length(pole_x);


pole = [pole11;pole];

pole_x = real(pole);
pole_y = imag(pole);


condition = y.';


A = buildLeastSquaresMatrix(samplepoint,pole,
    num_points1,number_pole);

perturbation = 1e-8;
[A_augmented, b_augmented] =
    augmentLeastSquaresMatrix(A, condition,
    perturbation);
coefficients = A_augmented \ b_augmented;

z = Findfz((0+2*i),coefficients,pole,num_points1,
    number_pole);

disp(z)


x_inter = linspace(-10, 10, 1001);
y_inter = linspace(10, -10, 1001);

linspace_matrix = zeros(length(x_inter),length(
    y_inter));


[X, Y] = meshgrid(x_inter, y_inter);

complex_grid = X + 1i * Y;


for i = 1:length(y_inter)
    for j = 1:length(x_inter)
```

```
        linspace_matrix(i,j) = Findfz(
   complex_grid(i,j),coefficients,pole,num_points1
   ,number_pole);
    end
end

y_r = repmat(y_inter',1,length(x_inter));
psi = linspace_matrix;

contour(x_inter, y_inter, -y_r+psi, 800);

title('flow graph');
xlabel('x');
ylabel('y');
axis equal;
hold on

plot(real(pole), imag(pole), 'b*');
plot(real(samplepoint), imag(samplepoint), 'b-');

axis equal;
grid on;
title('Points and the Unit Circle');
xlabel('Real Axis');
ylabel('Imaginary Axis');

hold on
plot(real(pole1),imag(pole1),'b*')
hold on
plot(real(pole2),imag(pole2),'b*')




function A = buildLeastSquaresMatrix(
   sample_points, poles, n1, n2)
    num_samples = length(sample_points);
    A = zeros(num_samples, n1 + n2);

    for i = 1:num_samples
        z = sample_points(i);
        for j = 1:n1
```

```matlab
            A(i, j) = real(1 / (z - poles(j)));
        end
        for k = (n1+1):(n1+n2)
            A(i, k) = log(abs(z - poles(k)));
        end
    end
end

function [A_augmented, d_augmented] = augmentLeastSquaresMatrix(A, d, perturbation)
    A_transpose_A = A' * A;
    d_transpose = A' * d;

    A_augmented = A_transpose_A + perturbation * eye(size(A_transpose_A, 1));
    d_augmented = d_transpose;
end

function B = Findfz(point,coeffients,pole,n1,n2)
    C = zeros(1,n1+n2);
    for i = 1:n1
        C(1,i) = real(1/(point-pole(i)));
    end
    for j = (n1+1):(n1+n2)
        C(1,j) = log(abs(point-pole(j)));
    end
    B = C * coeffients;
end
```

Listing E.1: Matlab Code

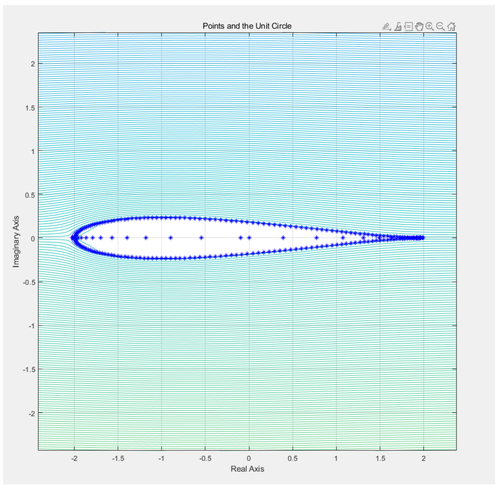


Figure E.1: Airfoil in MFS and Lightning

Appendix F

Airfoil in different direction

*

```
num_points1 = 10;
sigma = 2;
num_points_per_side = num_points1/2;

distances = zeros(1, num_points_per_side);

angle = pi/6 ;

for j = num_points_per_side:-1:1
    distances(num_points_per_side - j + 1) = exp
    (-sigma * (sqrt(num_points_per_side) - sqrt(j
    -1)));
end

pole1 = zeros(num_points_per_side, 2);
pole2 = zeros(num_points_per_side, 2);

num_points = 600;

n = 2;
b = 1.1;
elsi = 0.1;
factor = 49/50;
angles = linspace(0, 2*pi, num_points);

samplepoint =  (1+elsi)*exp(1i * angles);

```

```
newsamplepoint = (samplepoint - elsi)  + 1 ./(samplepoint - elsi) ;
samplepoint = newsamplepoint;

A = [max(real(samplepoint)),0];
B = [min(real(samplepoint)),0];

x = real(samplepoint);
y = imag(samplepoint);

pole_x1 = x * factor;
pole_y1 = y * factor;

meansample = mean(samplepoint);

center = [real(meansample),imag(meansample)];

for i = 1:num_points_per_side
    pole1(i, :) = center + (A - center) * (1-distances(i));
    pole2(i, :) = center + (B - center) * (1-distances(i));
end

list_all = [pole1; pole2];
x1 = list_all(:, 1);
y1 = list_all(:, 2);

pole11 = x1 + y1*1i;

pole_x = zeros(num_points / 3, 1);
pole_y = zeros(num_points / 3, 1);

for i = 1:(num_points/3)
    pole_x(i) = pole_x1(3*i-1);
end

for i = 1:(num_points/3)
    pole_y(i) = pole_y1(3*i-1);
end


```

```
pole =  pole_y*1i + pole_x;

number_pole = length(pole_x);


pole = [pole11;pole];

pole_x = real(pole);
pole_y = imag(pole);


condition = y.'*cos(angle)-x.'*sin(angle);


A = buildLeastSquaresMatrix(samplepoint,pole,
    num_points1,number_pole);

perturbation = 1e-8;
[A_augmented, b_augmented] =
    augmentLeastSquaresMatrix(A, condition,
    perturbation);
coefficients = A_augmented \ b_augmented;

z = Findfz((0+2*i),coefficients,pole,num_points1,
    number_pole);

disp(z)


x_inter = linspace(-10, 10, 1001);
y_inter = linspace(10, -10, 1001);

linspace_matrix = zeros(length(x_inter),length(
    y_inter));


[X, Y] = meshgrid(x_inter, y_inter);

complex_grid = X + 1i * Y;


for i = 1:length(y_inter)
```

```
    for j = 1:length(x_inter)
        linspace_matrix(i,j) = Findfz(
   complex_grid(i,j),coefficients,pole,num_points1
   ,number_pole);
    end
end

y_r = repmat(y_inter',1,length(x_inter));
x_r1 = repmat(x_inter',1,length(x_inter));
x_r = x_r1.';
totoal_r = cos(angle)*y_r-sin(angle)*x_r;

psi = linspace_matrix;

contour(x_inter, y_inter, -totoal_r+psi, 800);

title('flow graph');
xlabel('x');
ylabel('y');
axis equal;
hold on

plot(real(pole), imag(pole), 'b*');
plot(real(samplepoint), imag(samplepoint), 'b-');

axis equal;
grid on;
title('Points and the Unit Circle');
xlabel('Real Axis');
ylabel('Imaginary Axis');

hold on
plot(real(pole1),imag(pole1),'b*')
hold on
plot(real(pole2),imag(pole2),'b*')




function A = buildLeastSquaresMatrix(
   sample_points, poles, n1, n2)
    num_samples = length(sample_points);
    A = zeros(num_samples, n1 + n2);

    for i = 1:num_samples
        z = sample_points(i);
        for j = 1:n1
            A(i, j) = real(1 / (z - poles(j)));
        end
        for k = (n1+1):(n1+n2)
            A(i, k) = log(abs(z - poles(k)));
        end
    end
end

function [A_augmented, d_augmented] =
   augmentLeastSquaresMatrix(A, d, perturbation)
    A_transpose_A = A' * A;
    d_transpose = A' * d;

    A_augmented = A_transpose_A + perturbation *
   eye(size(A_transpose_A, 1));
    d_augmented = d_transpose;
end

function B = Findfz(point,coeffients,pole,n1,n2)
    C = zeros(1,n1+n2);
    for i = 1:n1
        C(1,i) = real(1/(point-pole(i)));
    end
    for j = (n1+1):(n1+n2)
        C(1,j) = log(abs(point-pole(j)));
    end
    B = C * coeffients;
end
```

Listing F.1: Matlab Code

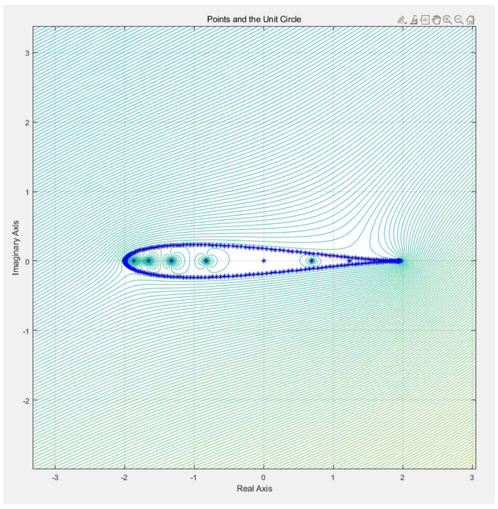


Figure F.1: Airfoil in different direction