CFD Project #05:

Generating a Grid that Wraps Around an Airfoil

Abstract:

In this project several methods of generating grids for none rectangular geometry have been examined. This project lays the ground work for understanding how rectangular computational domains can be converted into most physical non-rectangular domains. As most engineering applications will consider complex geometry this project give the tools to generates the grids that will be used to analyze the physics on or around a body.

The two methods studied are the algebraic method and the elliptic method. The first method applies a simple transformation from the computational domain to the physical domain. The second method exploits elliptical second order partial differential equation (PDE) transformation equation that can be solved using point Guass-Seidel (PGS). Solving elliptical PDEs using PGS has be used several times using on previous projects. These two methods demonstrate the process of generating as well communicate benefits draw backs to both methods.

Written By: Michael J. Bouey Jr.

Course: MAE 561

Section Number: 13151

Report Due: 23 Mar 2011

Table of Contents:

I. Introduction:

In the previous project an analysis of the vorticity equation was done. The purpose was to determine the velocity field, streamline field and vorticity field of a lid driven cavity. That project closely resembled actual engineering problems. In this project a grid wraps around an airfoil is to be generated. As many of of the engineering problems that exists will be unable to use rectangular girds it is important to understand how to create irregular grids. The computational domain is always rectangular, but the domain that an engineer must consider is dependent on the geometry of the product considered and the area around the product that must be considered. In this project a rectangular computational grid will be transformed into a grid that wraps around an airfoil.

II. Problem Statement:

Consider and air foil whose geometry is described be Equation 1. An outer C-shape domain is specified by a half circle and two parallel lines, as shown in Figure 1. Grid point distribution on the airfoil surface is to be clustered near the leading edge and trailing edges.

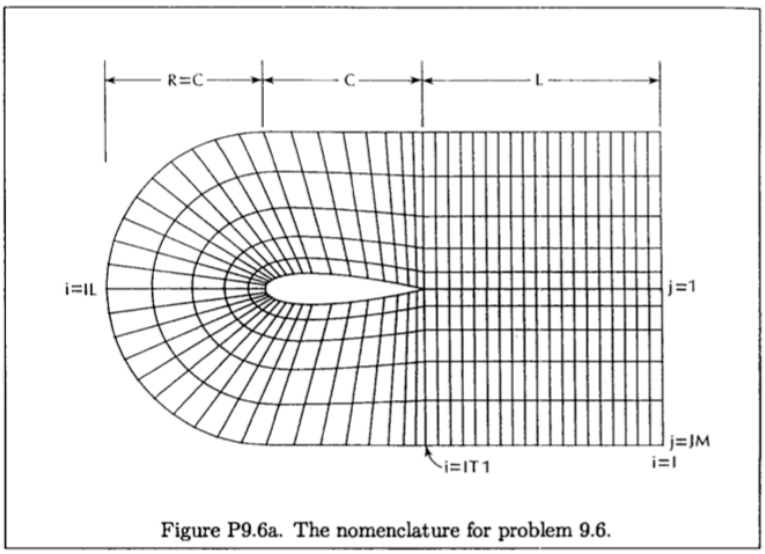

Figure P9.6a. The nomenclature for problem 9.6.

Figure 1: (Hoffmann) Figure P9-6a page 424

$$y = \frac{t}{0.2}\left(0.2969x^{1/2} - 0.126x - 0.3516x^2 + 0.2843x^3 - 0.1015x^4\right) \quad \textbf{(1)}$$

Part A: Generate a grid by an algebraic method with gird point clustering clustering near the airfoil surface and the wake region.

Part B: Generate a grid by the elliptic scheme with no grid clustering.

The following parameters were used:

$t = 0.2$, $c = 1.0$, $L = 1.5$, $IT1 = 20$, $IL = 38$, $JM = 30$

III. Derivations:

A. Algebraic Method

To be able to generate a C-type grid, a transformation from the rectangular domain to the C-type domain needs to be conducted. This can be accomplished through an algebraic transformation converting computational x and y coordinates to ξ and η coordinates of the C-type domain. A physical example of the a physical domain that must be transformed can be seen in Figure 2. The physical domain is then transformed to a rectangular computational domain as seen in Figure 3.

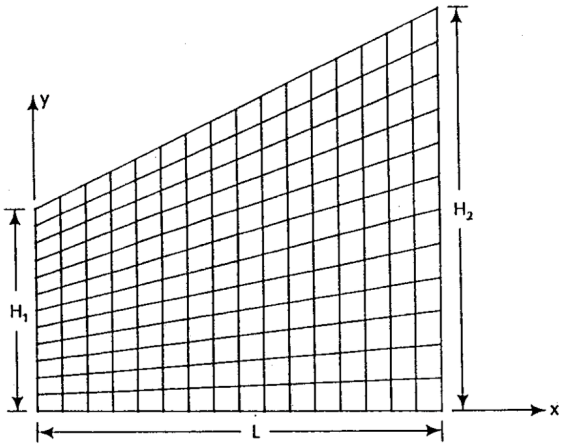

Figure 2: (Hoffmann) Figure 9-4 page 365

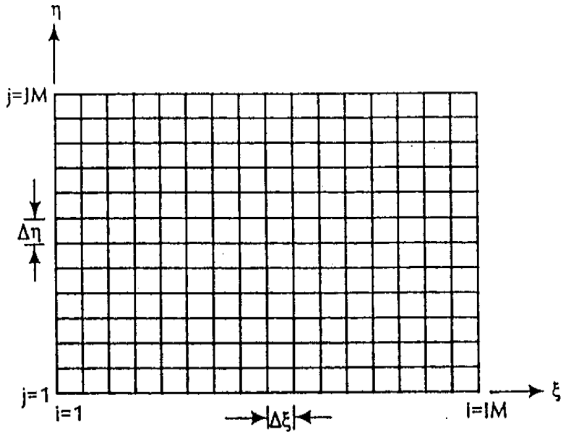

Figure 3: (Hoffmann) Figure 9-5 page 367

The relationship that exists between the physical and computational domains can seen in Equations 2 (9-19) and 3 (9-19). The physical domain can be converted to the computational through the use of Equations 4 (9-31) and 5 (9-32).

$$\xi = x \quad \textbf{(2)} \qquad \eta = \frac{y}{y_t} \quad \textbf{(3)} \qquad \xi = x \quad \textbf{(4)} \qquad \eta = \alpha + (1-\alpha)\frac{\ln\left[\dfrac{\beta + \dfrac{(2\alpha+1)y}{H} - 2\alpha}{\beta - \dfrac{(2\alpha+1)y}{H} + 2\alpha}\right]}{\ln\left(\dfrac{\beta+1}{\beta+1}\right)} \quad \textbf{(5)}$$

Where y_t is the upper limit of y and β is the clustering term. This effectively gives η as a ratio of y to the upper limit. β is given is a clustering parameter whose range is 1 to infinity. As the value of β approaches 1,the grid points cluster near the surface, where y = 0. This is

particularly useful when tight grids with concentrated in specific regions such as the airfoil of a wing are desired. The increase in grid points at the area of interest will further increase accuracy of any engineering analysis being conducted on this system. It is common to desire tighter grids at very specific regions while other location can be sparse, which saves on computational costs.

B. Elliptic Method

The elliptic method system starts with a of elliptic partial differential equations (PDE). The two elliptic equations that the system is derived from are Equations 6 (9-61) and 7 (9-62). Through the deviation that can be seen in Appendix F of Hoffmann's Computational Fluid Dynamics: Volume 1 Equations 6 and 7 were transformed into Equations 8 (9-63) and 9 (9-64).

$$\xi_{xx} + \xi_{yy} = 0 \quad \textbf{(6)} \qquad a x_{\xi\xi} - 2b x_{\xi\eta} + c x_{\eta\eta} = 0 \quad \textbf{(8)}$$

$$\eta_{xx} + \eta_{yy} = 0 \quad \textbf{(7)} \qquad a y_{\xi\xi} - 2b y_{\xi\eta} + c y_{\eta\eta} = 0 \quad \textbf{(9)}$$

Where: $a = x_\eta^2 + y_\eta^2$ $b = x_\xi x_\eta + y_\xi y_\eta$ $c = x_\xi^2 + y_\xi^2$

Equation 8 is a PDE that needs to be discretized and turned into a finite difference equation (FDE) which can be seen in Equation 10. Similarly Equation 9 is converted into equation 11.

$$a\left[\frac{x_{i+1,j} - 2x_{i,j} + x_{i-1,j}}{(\Delta\xi)^2}\right] - 2b\left[\frac{x_{i+1,j+1} - x_{i+1,j-1} + x_{i-1,j-1} - x_{i-1,j+1}}{4\Delta\xi\Delta\eta}\right] + c\left[\frac{x_{i,j+1} - 2x_{i,j} + x_{i,j-1}}{(\Delta\eta)^2}\right] = 0 \quad \textbf{(10)}$$

$$a\left[\frac{y_{i+1,j} - 2y_{i,j} + y_{i-1,j}}{(\Delta\xi)^2}\right] - 2b\left[\frac{y_{i+1,j+1} - y_{i+1,j-1} + y_{i-1,j-1} - y_{i-1,j+1}}{4\Delta\xi\Delta\eta}\right] + c\left[\frac{y_{i,j+1} - 2y_{i,j} + y_{i,j-1}}{(\Delta\eta)^2}\right] = 0 \quad \textbf{(11)}$$

Using a point Gauss-Seidel (PGS) iterative method the FDE equation can isolate the $x_{i,j}$ and $y_{i,j}$ terms and then solved for. The resulting equations, Equation 12 and 13, after some algebraic manipulation can be seen below.

$$x_{i,j} = \frac{\left\{\frac{a}{(\Delta\xi)^2}\left[x_{i+1,j} + x_{i-1,j}\right] - \frac{b}{2\Delta\xi\Delta\eta}\left[x_{i+1,j+1} - x_{i+1,j-1} + x_{i-1,j-1} - x_{i-1,j+1}\right] + \frac{c}{(\Delta\eta)^2}\left[x_{i,j+1} + x_{i,j-1}\right]\right\}}{2\left[\frac{a}{(\Delta\xi)^2} + \frac{c}{(\Delta\eta)^2}\right]} \quad \textbf{(12)}$$

$$y_{i,j} = \frac{\left\{ \frac{a}{(\Delta\xi)^2}\left[y_{i+1,j} + y_{i-1,j} \right] - \frac{b}{2\Delta\xi\Delta\eta}\left[y_{i+1,j+1} - y_{i+1,j-1} + y_{i-1,j-1} - y_{i-1,j+1} \right] + \frac{c}{(\Delta\eta)^2}\left[y_{i,j+1} + y_{i,j-1} \right] \right\}}{2\left[\frac{a}{(\Delta\xi)^2} + \frac{c}{(\Delta\eta)^2} \right]} \quad \textbf{(13)}$$

IV. Development of Coding Algorithms:

A. Algebraic Method

The coding algorithm will be illustrate with a block diagram. The code can be seen directly following the block diagram. The block diagram illustrates a short synopsis of how the code is employed to solve the vorticity equation. The code is also commented to ensure understanding.

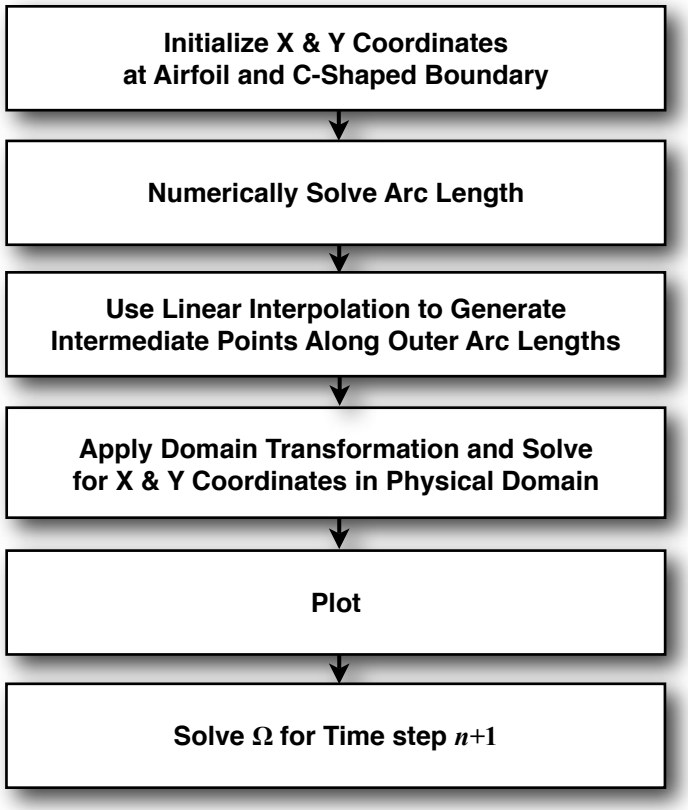

Figure 4: Algebraic Coding Block Diagram:

Algebraic Code:

```
close all;
clear all;

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

```
% Define all the parameters
%------------------------------------------------------------------------%

Na = 50;
M = 30;
A = 10;          %Clustering points near leading edge, inf = no clustering
B = 2;              %Clustering points near the surface, 1 = no clustering
t = .12;
c = 1;
L = 1;
N = floor(2*c/(2*c+L)*Na);
x1 = c*ones(1,N+1);
y1 = zeros(1,N+1);
x2 = x1;
y2 = y1;
y2(1) = c;
s1 = y1;
s2 = s1;
N2 = 250;

%------------------------------------------------------------------------%
% Solve Algebraic Method
%------------------------------------------------------------------------%
for i = 2:N2+1
    x1(i) = c*(1-(i-1)/N2);
    y1(i) = max([t/.2*(.2969*x1(i)^.5-.126*x1(i)-.3516*x1(i)^2+.2843*x1(i)^3-.
1015*x1(i)^4) 0]);
    s1(i) = s1(i-1)+sqrt((x1(i)-x1(i-1))^2+(y1(i)-y1(i-1))^2);
    x2(i) = c*(1-2*(i-1)/N2);
    if x2(i) >= 0;
        y2(i) = c;
    else
        y2(i) = sqrt(c^2-(x2(i))^2);
    end
    s2(i) = s2(i-1)+sqrt((x2(i)-x2(i-1))^2+(y2(i)-y2(i-1))^2);
end
S1 = s1(end)*(1-exp(-(0:N)/A))/(1-exp(-(N)/A));
S1(end)/s1(end)
S2 = (0:N)*s2(end)/N;
for i = 1:N+1
    X1(i) = interp1(s1,x1,S1(i));
    Y1(i) = interp1(s1,y1,S1(i));
    X2(i) = interp1(s2,x2,S2(i));
    Y2(i) = interp1(s2,y2,S2(i));
end
r = exp(((1:(M+1))-M+1)/((M+1)/B));
r = r-r(1);
r = r/max(r);
for i = 1:N+1
    for j = 1:M+1
        x(i,j) = X1(i) + r(j)*(X2(i)-X1(i));
        y(i,j) = Y1(i) + r(j)*(Y2(i)-Y1(i));
    end
end
N = Na-N;
x = [((L+c):-L/N:(c+L/N))'*ones(1,M+1); x];
y = [(c*r'*ones(1,N))'; y];
x = [x; x(end-1:-1:1,:)];
y = [y ;-y(end-1:-1:1,:)];
[n,m] = size(x);
```

```
%-------------------------------------------------------------------------%
% Plot Algebraic Method
%-------------------------------------------------------------------------%
mesh(x,y,zeros(n,m))
view(0,90)
colormap([0 0 0])
axis('equal','tight');
xlabel('Length [unitless]','FontSize',14);
ylabel('Length [unitless]','FontSize',14);
title('Algebraic Method ','FontSize',18);
```

B. Elliptic Method

The coding algorithm be will illustrate with a block diagram. The code can be seen directly following the block diagram. The block diagram illustrates a short synopsis of how the code is employed to solve the vorticity equation. The code is also commented to ensure understanding.

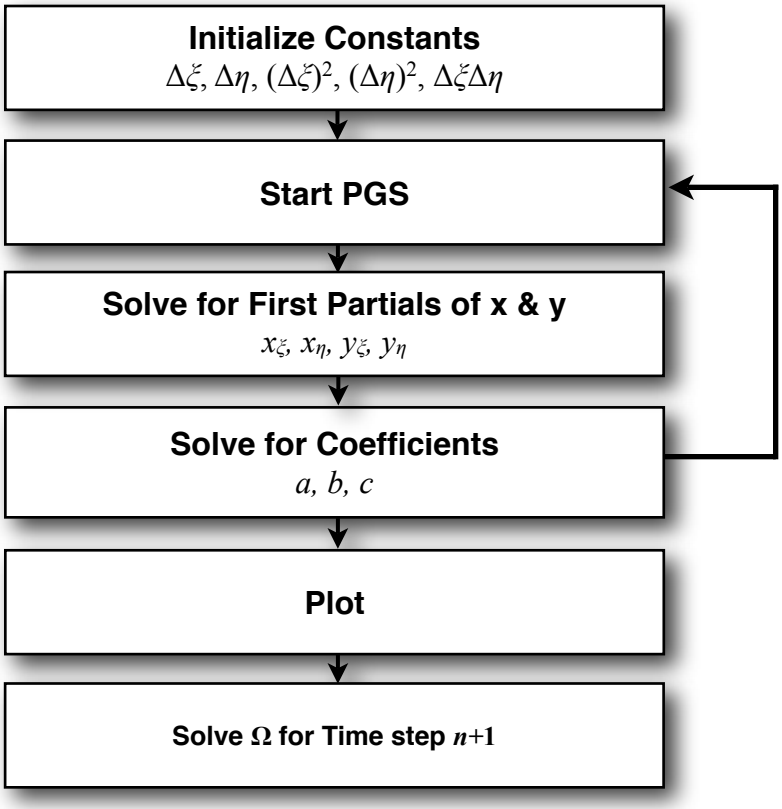

Figure 5: Elliptic Coding Block Diagram**:**

Elliptic Code:

```
%-------------------------------------------------------------------------%
% Predetermined Constants
%-------------------------------------------------------------------------%
```

```
de = 1/M;
de2 = de^2;
dn = 1/Na;
dn2 = dn^2;
dedn = 2/(M*Na);
P = 100;            %number of iterations for the elliptic solver.

%-------------------------------------------------------------------------%
% Solve Elliptic Method
%-------------------------------------------------------------------------%

for k = 1:P
    for i = 2:2*Na-1
        for j = 2:M-1
            xn = (x(i+1,j)-x(i-1,j))/(2*dn);
            yn = (y(i+1,j)-y(i-1,j))/(2*dn);
            xe = (x(i,j+1)-x(i,j-1))/(2*de);
            ye = (y(i,j+1)-y(i,j-1))/(2*de);
            a(i,j) = xn^2+yn^2;
            b(i,j) = xe*xn+ye*yn;
            c(i,j) = xe^2+ye^2;
        end
    end
    for i = 2:2*Na-1
        for j = 2:M-1
            x(i,j) = (a(i,j)/de2*(x(i+1,j)+x(i-1,j))+c(i,j)/dn2*(x(i,j+1)+x(i,j-1))...
                -b(i,j)/dedn*(x(i+1,j+1)-x(i+1,j-1)+x(i-1,j-1)-x(i-1,j+1)))...
                /(2*(a(i,j)/de2+c(i,j)/dn2));
            y(i,j) = (a(i,j)/de2*(y(i+1,j)+y(i-1,j))+c(i,j)/dn2*(y(i,j+1)+y(i,j-1))...
                -b(i,j)/dedn*(y(i+1,j+1)-y(i+1,j-1)+y(i-1,j-1)-y(i-1,j+1)))...
                /(2*(a(i,j)/de2+c(i,j)/dn2));
        end
    end
end
xn = ones(Na,M)/0;
yn = xn;
xe = xn;
ye = xn;
for i = 2:Na-1
    for j = 2:M-1
        xn(i,j) = (x(i+1,j)-x(i-1,j))/(2*dn);
        yn(i,j) = (y(i+1,j)-y(i-1,j))/(2*dn);
        xe(i,j) = (x(i,j+1)-x(i,j-1))/(2*de);
        ye(i,j) = (y(i,j+1)-y(i,j-1))/(2*de);
        J(i,j) = xn(i,j)*ye(i,j)-yn(i,j)*xe(i,j); % Jacobian
    end
end
figure
mesh(x,y,zeros(n,m))
xlabel('Length [unitless]','FontSize',14);
ylabel('Length [unitless]','FontSize',14);
title('Elliptic Method ','FontSize',18);
axis('equal','tight');

%-------------------------------------------------------------------------%
% Plot Elliptic Method
%-------------------------------------------------------------------------%

view(0,90)
colormap([0 0 0])
```

```
%-------------------------------------------------------------------------%
% Plot Jacobian
%-------------------------------------------------------------------------%

figure
x=2:30; y=2:50; [X,Y]=meshgrid(x,y);
surf(X,Y,J);
view(150,30)
title('Jacobian ','FontSize',18);
```

V. Discussion:

In this section the numerical results will be illustrated with the plots that were generated while running the code displaced in Section IV. The results will be analyzed and discussed.

Results:

The algebraic code is solved several times. The algebraic grid generated is a grid without clustering and can be seen in Figure 6. This grid was generated for to have results to compare against. An algebraic was grid generated with clustering and can be seen in Figure 7.

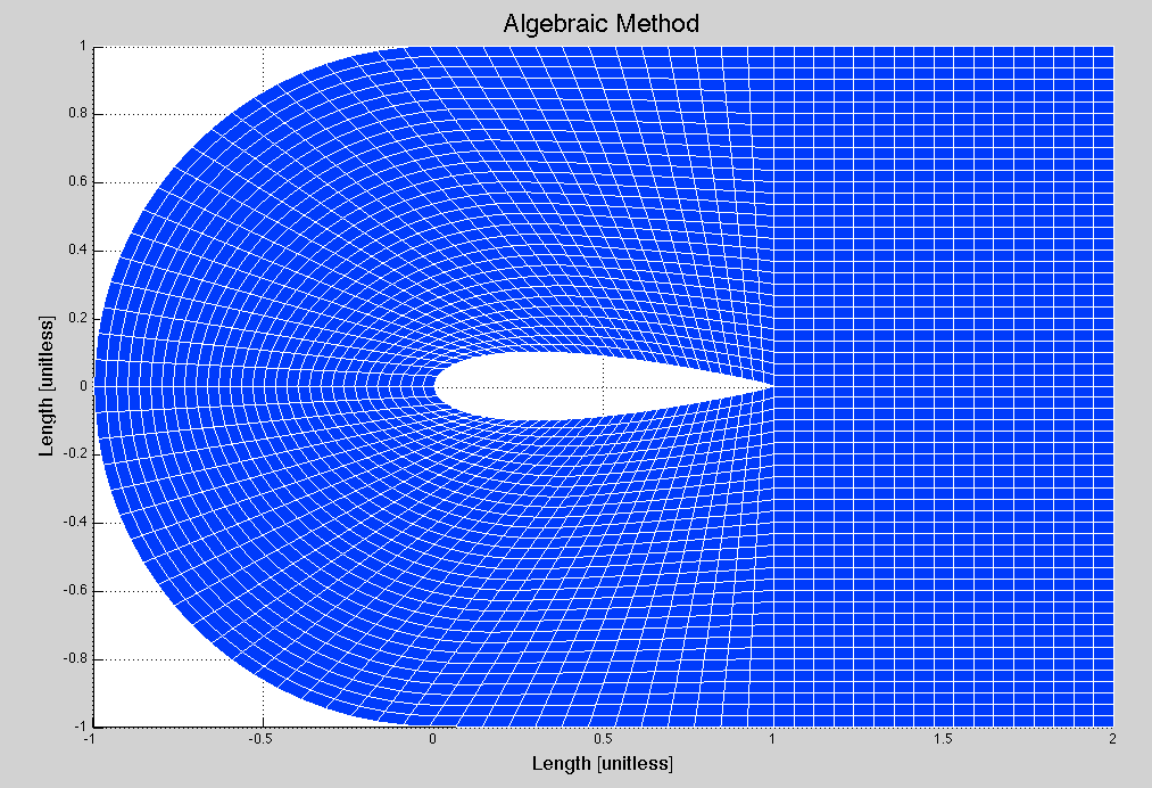

Figure 6: Algebraic Method Grid without Clustering

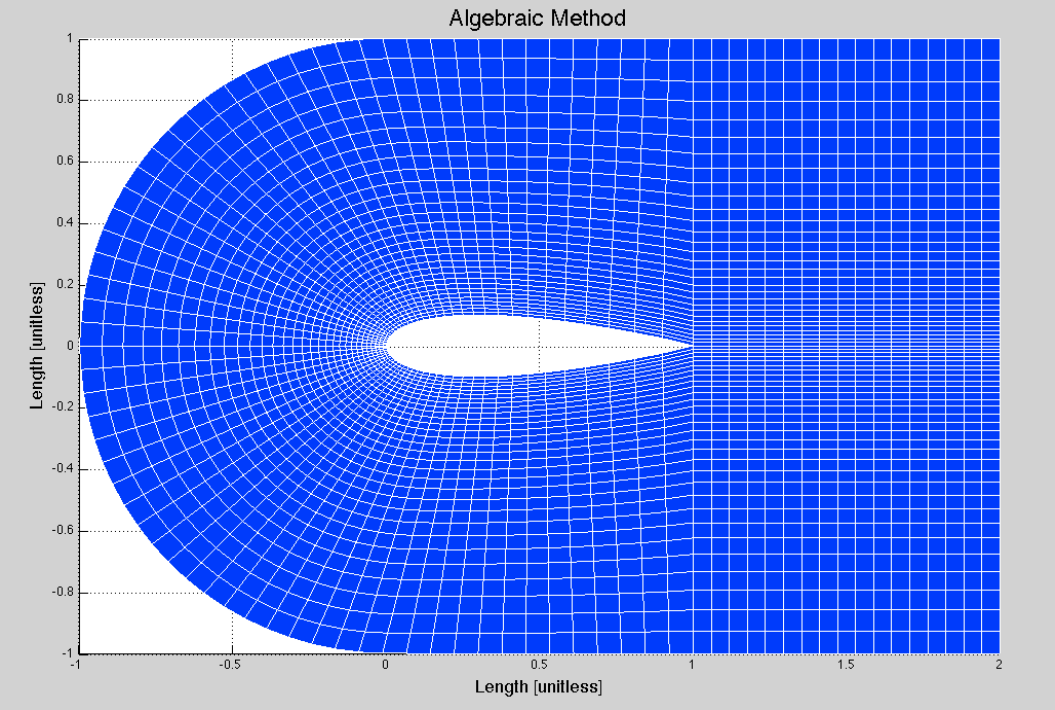

Figure 7: Algebraic Method Grid with Clustering

It is easy to see that the clustering can easy changed. These results can be compared using the Jacobian plots which can be in Figures 8 and 9, where Figure 8 corresponds to the grid without clustering and Figure 9 corresponds to the grid with clustering. It is important to note that the Jacobian plots are continuous. As long as the plots are generally continuous and do not experience significant changes the results can be considered to be fairly accurate. To demonstrate this point a poor grid using algebraic method was intensionally generated to illustrate this point.

Figures 10 and 11 illustrate a poor physical domain to

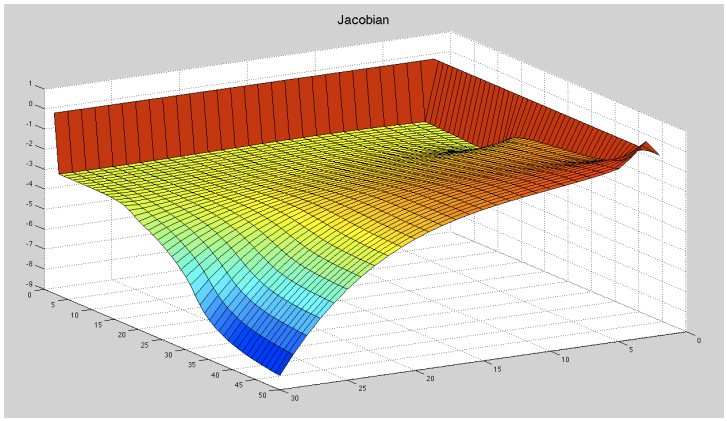

Figure 8: Jacobian, Algebraic Method Grid without Clustering

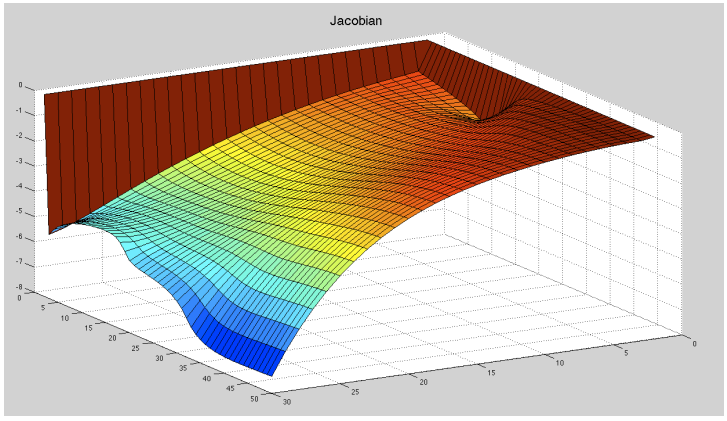

Figure 9: Jacobian, Algebraic Method Grid with Clustering

computational domain transformation. Figure 10 is the intensionally created “bad” grid. Figure 11 is the Jacobian that illustrates a none continuous Jacobian plot.

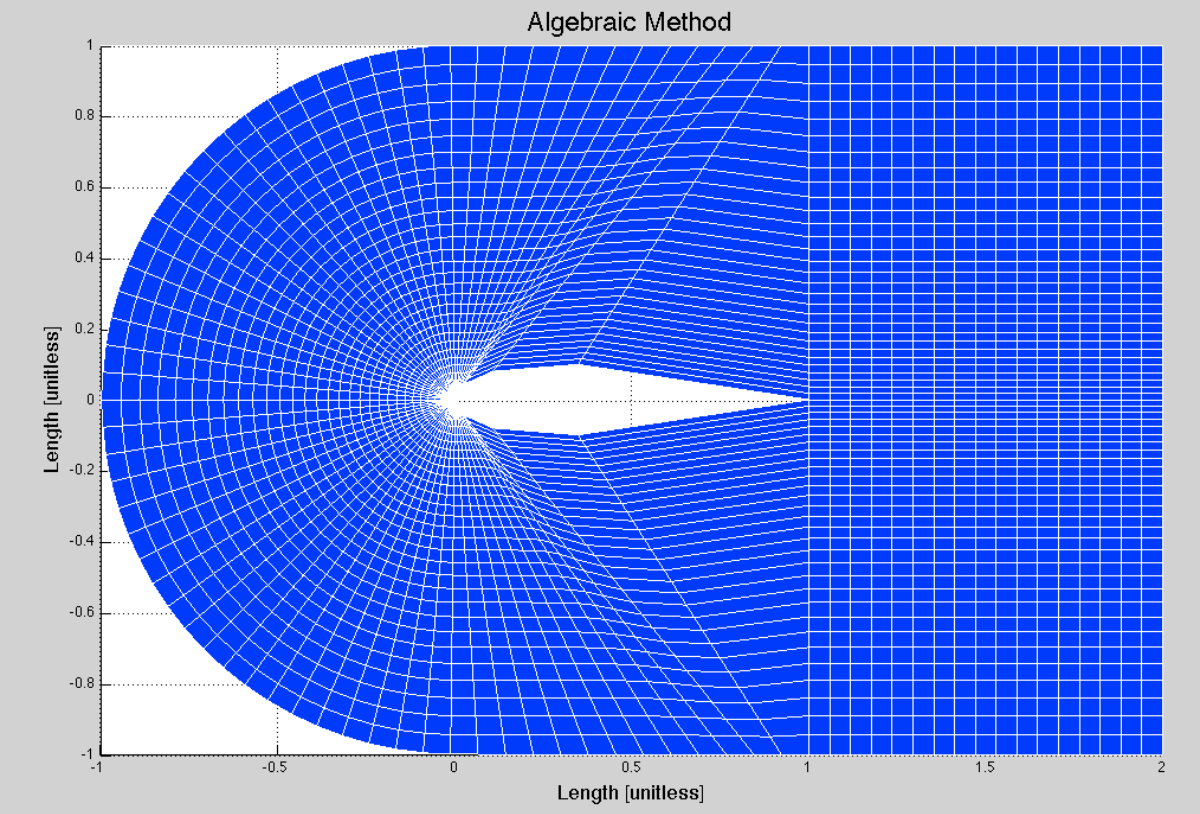

Figure 10: Algebraic Method Grid with a “bad” grid

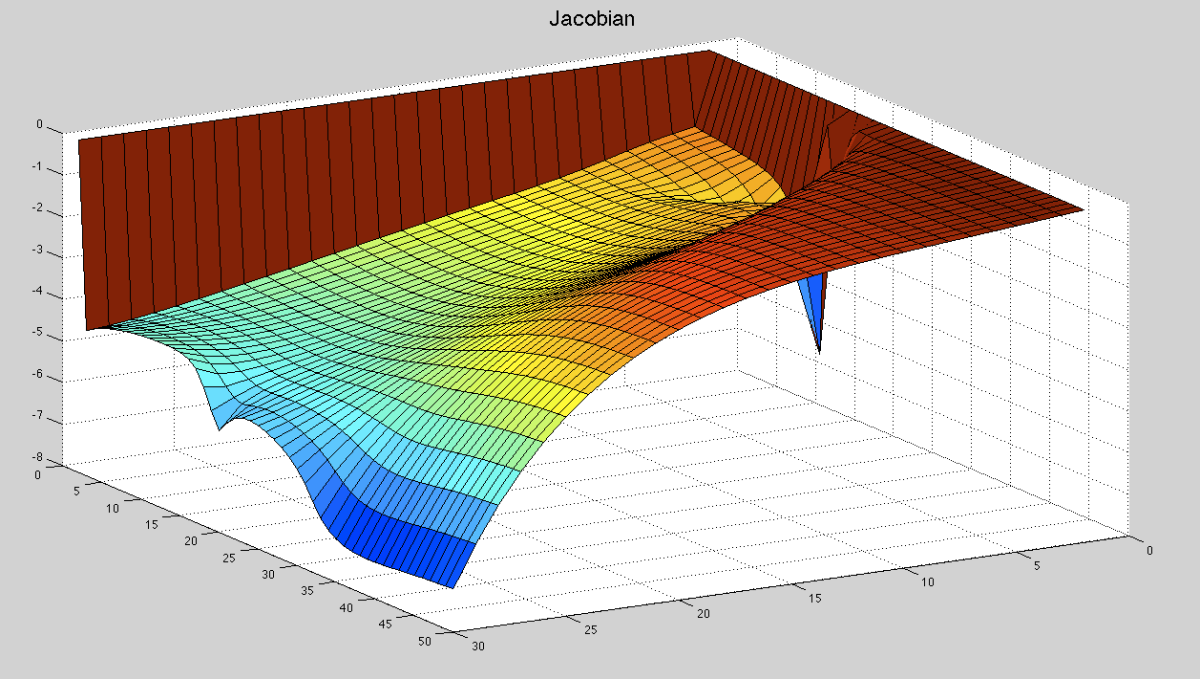

Figure 11: Jacobian, Algebraic Method Grid with a “bad” grid

The final plot present is Figure 12 which resulted from the elliptic method. This plot shows a very evenly distributed grid; however, has to be solved using an iterative method, so will require more computational cost.

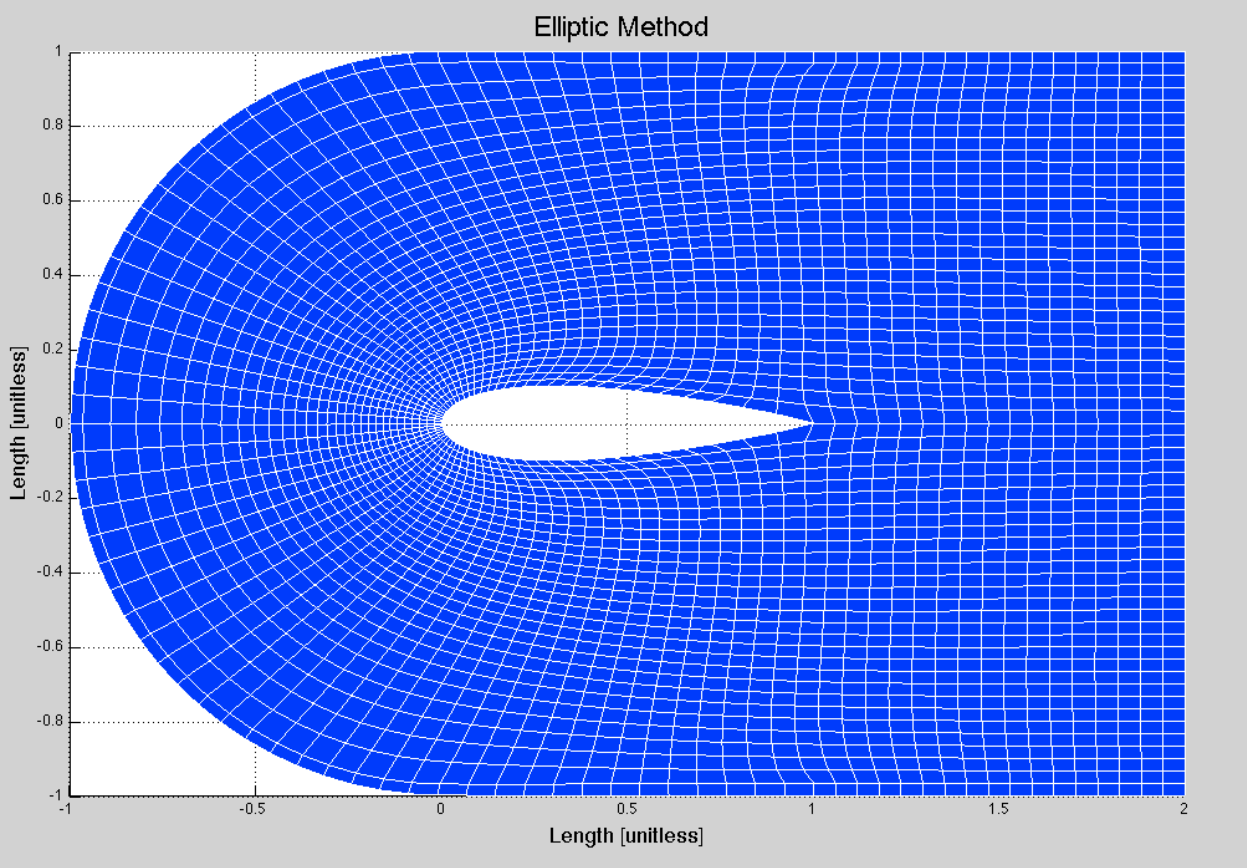

Figure 12: Elliptic Method Grid without Clustering

VI. Conclusion:

The results from this project demonstrate varying things about the different methods. For instance, the algebraic method is easy to derive and fairly easy to generate; however can have very specific inaccuracies due to the loss of information through the domain transformation. The elliptic method yields a grid that is easy to generate evenly space grids but clustering is not very difficult if not impossible. There is also the loss of computational cost associate with solving for a grid utilizing an iterative method.