

NAVAL POSTGRADUATE SCHOOL

Monterey, California



THESIS

**AN IMPROVED ALGEBRAIC GRID GENERATOR FOR
NUMERICAL AERODYNAMIC ANALYSES OF AIRFOIL
CROSS-SECTIONS**

by

Justin M. Verville

December 2002

Thesis Advisor:

Kevin Jones

Thesis Co-Advisor:

Max Platzer

Approved for public release; distribution is unlimited

THIS PAGE INTENTIONALLY LEFT BLANK

REPORT DOCUMENTATION PAGE			<i>Form Approved OMB No. 0704-0188</i>	
Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instruction, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188) Washington DC 20503.				
1. AGENCY USE ONLY (Leave blank)		2. REPORT DATE December 2002	3. REPORT TYPE AND DATES COVERED Master's Thesis	
4. TITLE AND SUBTITLE: An Improved Algebraic Grid Generator for Numerical Aerodynamic Analyses of Airfoil Cross-Sections			5. FUNDING NUMBERS	
6. AUTHOR(S) Justin M. Verville				
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Naval Postgraduate School Monterey, CA 93943-5000			8. PERFORMING ORGANIZATION REPORT NUMBER	
9. SPONSORING /MONITORING AGENCY NAME(S) AND ADDRESS(ES) N/A			10. SPONSORING/MONITORING AGENCY REPORT NUMBER	
11. SUPPLEMENTARY NOTES The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.				
12a. DISTRIBUTION / AVAILABILITY STATEMENT Approved for public release; distribution is unlimited			12b. DISTRIBUTION CODE	
13. ABSTRACT (maximum 200 words) Although computer-processing power has increased dramatically over the past few decades, minimizing computation time is still critical when conducting numerical aerodynamic analyses. One area where this is evident is the grid generation routines used in most code for this area of research. While many more sophisticated grid generation techniques are available, algebraic grid generation is still in use today due strictly to efficiency. Computational efficiency is of particularly great concern during analyses that involve motion of the surface being analyzed, since computing a new grid at each time step is often required. Unfortunately however, few if any, algebraic grid generation routines exist that are powerful enough to produce a grid with no overlapping gridlines and minimal distortion, yet still minimize computation time. As a result, the purpose of this thesis was to design such a routine. The end result is a C-Grid generating routine with a Graphical User Interface (GUI) called Astro Grid.				
14. SUBJECT TERMS Algebraic Grid Generator, Aeronautical Engineering, Navier Stokes, Flow Solver,			15. NUMBER OF PAGES 158	
			16. PRICE CODE	
17. SECURITY CLASSIFICATION OF REPORT Unclassified	18. SECURITY CLASSIFICATION OF THIS PAGE Unclassified	19. SECURITY CLASSIFICATION OF ABSTRACT Unclassified	20. LIMITATION OF ABSTRACT UL	

THIS PAGE INTENTIONALLY LEFT BLANK

Approved for public release; distribution is unlimited

**AN IMPROVED ALGEBRAIC GRID GENERATOR FOR NUMERICAL
AERODYNAMIC ANALYSES OF AIRFOIL CROSS-SECTIONS**

Justin M. Verville
Lieutenant, United States Naval Reserve
B.S., Purdue University, 1995

Submitted in partial fulfillment of the
requirements for the degree of

AERONAUTICAL AND ASTRONAUTICAL ENGINEER

from the

**NAVAL POSTGRADUATE SCHOOL
December 2002**

Author: Justin M. Verville

Approved by: Kevin Jones
Thesis Advisor

Max Platzer
Thesis Co-Advisor

Max Platzer
Chairman
Department of Aeronautical and Astronautical Engineering

THIS PAGE INTENTIONALLY LEFT BLANK

ABSTRACT

Although computer-processing power has increased dramatically over the past few decades, minimizing computation time is still critical when conducting numerical aerodynamic analyses. One area where this is evident is the grid generation routines used in most code for this area of research. While many more sophisticated grid generation techniques are available, algebraic grid generation is still in use today due strictly to efficiency. Computational efficiency is of particularly great concern during analyses that involve motion of the surface being analyzed, since computing a new grid at each time step is often required.

Unfortunately however, few if any, algebraic grid generation routines exist that are powerful enough to produce a grid with no overlapping gridlines and minimal distortion, yet still minimize computation time. As a result, the purpose of this thesis was to design such a routine. The end result is a C-Grid generating routine with a Graphical User Interface (GUI) called Astro Grid.

THIS PAGE INTENTIONALLY LEFT BLANK

TABLE OF CONTENTS

I.	INTRODUCTION	1
A.	METHODOLOGY	1
II.	CODE DESCRIPTION	3
A.	OVERVIEW	3
B.	USER-CONTROLLED CONSTANTS.....	6
1.	Type.....	7
2.	AirfoilType.....	7
3.	xB, yB	8
4.	CHORD.....	8
5.	r.....	8
6.	m	8
7.	NACA4.....	8
8.	thick1, thick2, thick3	9
9.	SRigid.....	9
10.	sizesmooth.....	9
11.	bendslopebot, bendslopetop	9
12.	aoa.....	10
13.	xo, yo.....	10
14.	plunge	10
15.	RBPitch, LBPitch, RBPlunge, LBPlunge	10
16.	obendslopebot, obendslopetop	10
17.	NumLowerSeg, NumUpperSeg.....	11
18.	AWP, AWQ, LAP, LAQ, LRP, LRQ, LSP, LSQ, UAP, UAQ, URP, URQ	11
19.	LANUM, UANUM	12
20.	Dir, LDir	12
21.	EdgeLowerSeg, EdgeUpperSeg	12
C.	UTILITY FUNCTIONS	12
1.	AirfoilReadIn.m	12
2.	ArcLength.m.....	13
3.	ArcLengthBotomt.m, ArcLengthTop.m	13
4.	Duplicate.m.....	14
5.	Metric.m.....	14
6.	Naca.m.....	14
7.	NacaSlope.m	15
8.	RTESlope.m.....	15
9.	RectArcLengthBot.m, RectArcLengthTop.m	15
10.	RotateClockwise.m.....	15
11.	Stretch.m.....	16
12.	UpperSurfaceNormal.m	16

D.	FUNCTIONS GENERATING GRIDLINES WITH USER-CONTROLLED DISTRIBUTIONS	16
1.	AirfoilWakeInit.m, AirfoilWake.m	16
2.	LeftSurfaceInit.m, LeftSurface.m, RectLeftSurface.m	18
3.	LowerAirfoil.m, LowerAirfoilRTE.m, UpperAirfoil.m, ...	19
	UpperAirfoilRTE.m.....	19
4.	LowerRightVerticalInit.m, LowerRightVertical.m, UpperRightVerticalInit.m, UpperRightVertical.m	20
E.	RIGID SURFACE GENERATING FUNCTIONS.....	20
1.	LowerRigid.m, UpperRigid.m	20
F.	OUTER GRIDLINE GENERATING FUNCTIONS	24
1.	LowerBottom.m, RectLowerBottom.m, UpperTop.m, RectUpperTop.m.....	24
6.	LowerHorizontal.m, UpperHorizontal.m	25
7.	WakeLower.m, WakeUpper.m.....	25
G.	PITCH AND PLUNGE MOTION FUNCTION	26
1.	PitchPlungeRigid.m	26
III.	USER'S GUIDE	27
A.	INTRODUCTION.....	27
B.	INITIAL DATA	27
	APPENDIX A.....	47
A.	UTILITY FUNCTIONS	47
1.	AirfoilReadIn.m	47
2.	ArcLength.m.....	48
3.	ArcLengthBottom.m	48
4.	ArcLengthTop.m.....	49
5.	CreateOutput.m	50
6.	Duplicate.m.....	51
7.	Metric.m.....	51
8.	Naca.m.....	53
9.	NacaSlope.m	54
10.	RTEsSlope.m.....	56
11.	RectArcLengthBottom.m	57
12.	RectArcLengthTop.m	57
13.	RotateClockwise.m.....	58
14.	Stretch.m.....	58
15.	UpperSurfaceNormal.m	58
B.	FUNCTIONS GENERATING GRIDLINES WITH USER-CONTROLLED DISTRIBUTIONS	59
1.	AirfoilWakeInit.m.....	59
2.	AirfoilWake.m.....	60
3.	LeftSurfaceInit.m.....	62
4.	LeftSurface.m	63
5.	RectLeftSurface.m	65

	6.	LowerAirfoil.m.....	67
	7.	LowerAirfoilRTE.m.....	69
	8.	UpperAirfoil.m.....	71
	9.	UpperAirfoilRTE.m.....	73
	10.	LowerRightVerticalInit.m.....	76
	11.	LowerRightVertical.m.....	76
	12.	UpperRightVerticalInit.m.....	77
	13.	UpperRightVertical.m.....	77
C.		RIGID SURFACE GENERATING FUNCTIONS.....	78
	1.	LowerRigid.m.....	78
	2.	UpperRigid.m.....	86
D.		OUTER GRID GENERATING FUNCTIONS.....	94
	1.	LowerBottom.m.....	94
	2.	RectLowerBottom.m.....	97
	3.	UpperTop.m.....	99
	4.	RectUpperTop.m.....	101
	5.	LowerHorizontal.m.....	103
	6.	UpperHorizontal.m.....	106
	7.	WakeLower.m.....	108
	8.	WakeUpper.m.....	109
E.		PITCH AND PLUNGE MOTION FUNCTION.....	111
	1.	PitchPlungeRigid.m.....	111
F.		GRAPHICAL USER INTERFACE CALLBACK FILES.....	112
	1.	AdjustMotion.m.....	112
	2.	AdjustOuterGrid.m.....	112
	3.	AdjustRigidData.m.....	112
	4.	GridMotion.m.....	113
	5.	GridMotion2.m.....	114
	6.	InitialData.m.....	115
	7.	PlotEntire.m.....	116
	8.	PlotRigid.m.....	117
	9.	PQData.m.....	118
	10.	RedoInitial.m.....	119
	11.	RedoNumAirfoilSegments.m.....	120
	12.	RedoOuterGrid.m.....	120
	13.	RedoPQData.m.....	120
	14.	SelectAirfoilType.m.....	123
	15.	SelectBoundary.m.....	123
	16.	SelectxTE.m.....	124
	17.	UpdateAdjustRigid.m.....	124
	18.	UpdateOuterGrid.m.....	124
	19.	ViewEntire.m.....	125
	20.	ViewEntireGrid.m.....	127
	21.	ViewLeadingEdgeLine.m.....	128
	22.	ViewLowerAirfoil.m.....	129

23.	ViewLR.m	130
24.	ViewRigid.m	131
25.	ViewTrailingEdgeLine.m	134
26.	ViewUpperAirfoil.m	135
27.	ViewUR.m	136
LIST OF REFERENCES		139
INITIAL DISTRIBUTION LIST		141

LIST OF FIGURES

Figure 1.	“Leading and Trailing Edge Lines”	3
Figure 2.	“Circumferential Gridlines”	4
Figure 3.	“Radial Gridlines”	5
Figure 4.	Hyperbolic Tangent Curve.....	6
Figure 5.	Boundary Types.....	7
Figure 6.	Airfoil Point Distribution.....	11
Figure 7.	Initial Data Menu	27
Figure 8.	Boundary Data Entry	29
Figure 9.	Effect of Size of Smoothing Region on Rigid Surface – 0.3 Chord	31
Figure 10.	Effect of Size of Smoothing Region on Rigid Surface – 0.95 Chord	32
Figure 11.	Effect of Size of Smoothing Region on Rigid Surface – 0.7 Chord	33
Figure 12.	Rigid Surface Enforce Normal Constants – 0.8, 0.8	34
Figure 13.	Rigid Surface Enforce Normal Constants – 0.1, 0.1	35
Figure 14.	Effect of Right Plunge Constant on Trailing Edge Line – 1.0.....	37
Figure 15.	Effect of Right Pitch Constant on Trailing Edge Line – 0.0.....	38
Figure 16.	Number of Airfoil Segments Menu Should be on same page as figure.....	39
Figure 17.	P and Q Data Menu – Two Segments Per Airfoil Surface	40
Figure 18.	Adjust Rigid Surface Data Menu.....	42
Figure 19.	Outer Grid Menu.....	42
Figure 20.	Adjust Outer Grid Data Menu.....	43
Figure 21.	Adjust Motion Menu.....	44

THIS PAGE INTENTIONALLY LEFT BLANK

ACKNOWLEDGMENTS

This thesis is dedicated to my fiancé, Natalie Chouinard. Without her love, patience and support, this thesis would not have been possible.

THIS PAGE INTENTIONALLY LEFT BLANK

I. INTRODUCTION

A. METHODOLOGY

In order to produce the best possible grid without increasing processing time, Astro Grid was designed to be an interactive algebraic grid generation routine. Due to MATLAB's ease of manipulating arrays and displaying data graphically, it was used to generate the source code for Astro Grid. Although the majority of aerodynamic flow solvers are written in Fortran, in the end, the gains realized from using MATLAB far outweighed the cost of having to translate the final product into Fortran or any other computer language.

In the MATLAB version of Astro Grid, the user is given control over several constants, which affect the appearance of the grid and are discussed later. The user is able to interactively change the constants, view the effects, and make additional changes if needed. While an interactive routine is acceptable for the initial grid generation, most flow solvers, which often generate a new grid at each time step, must contain a fully automated grid generator. Therefore, once the user is satisfied with the appearance of the grid in MATLAB, the constants are saved to a file, which is then read into the flow solver source code. The version of Astro Grid embedded in the flow solver source code has the interactive portion removed and simply generates the grid in the same manner as that viewed in MATLAB. The result is a grid free from distortion and overlap without the added computation time of a more complicated and computationally expensive grid generator.

THIS PAGE INTENTIONALLY LEFT BLANK

II. CODE DESCRIPTION

A. OVERVIEW

Astro Grid is implemented by creating two arrays. The first array contains the grid's x-coordinates, and is given the name X . The second array contains the grid's y-coordinates and is given the name Y . The size of both of these three-dimensional arrays is [the number of points in the horizontal direction $\times$ the number of points in the vertical direction $\times$ two]. The coordinates for the upper half of the grid are stored in the portion of these arrays with the third dimension's index equaling one. The coordinates for the lower half of the grid are stored in the portion of these arrays with the index of the third dimension equaling two. The horizontal lines that connect the airfoil's leading and trailing edges with the left and right boundaries separate the upper and lower halves of the grid. Throughout this report, these lines will be referred to as the "*Leading Edge Line*" and "*Trailing Edge Line*" respectively and are shown in Figure 1.

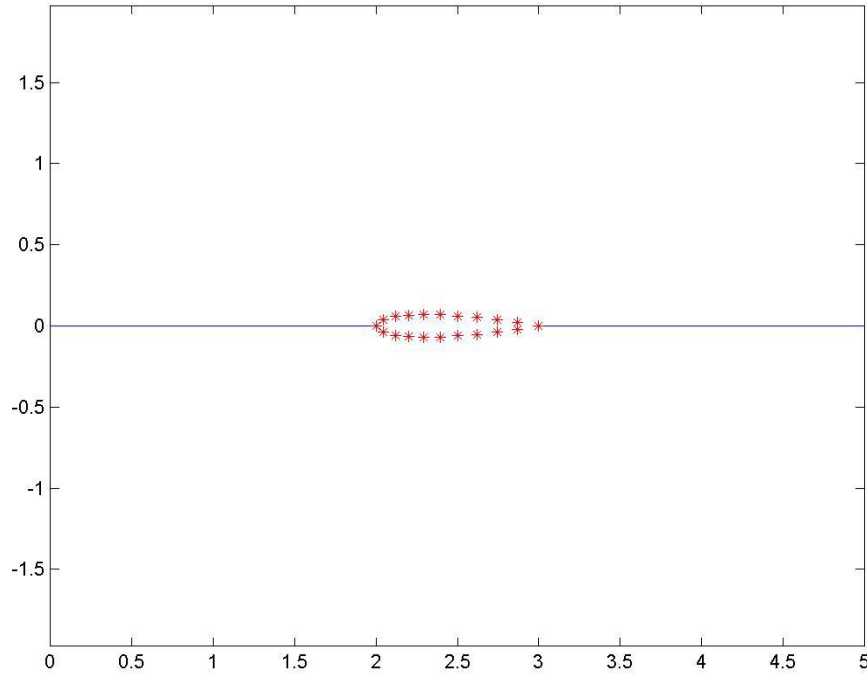


Figure 1. "Leading and Trailing Edge Lines"

For the both the upper and lower halves of the grid, the point with the arrays' first dimension's index equaling one is on the *Leading Edge Line*. Plotting the points attained by increasing the arrays' first dimension index and holding the other dimensions constant describes lines moving initially up then aft parallel to the airfoil surface until the right boundary is reached. For the purposes of this report, these gridlines are referred to as "circumferential gridlines" and are shown in Figure 2. As one can see in Figure 2, Astro Grid produces a C-Grid.

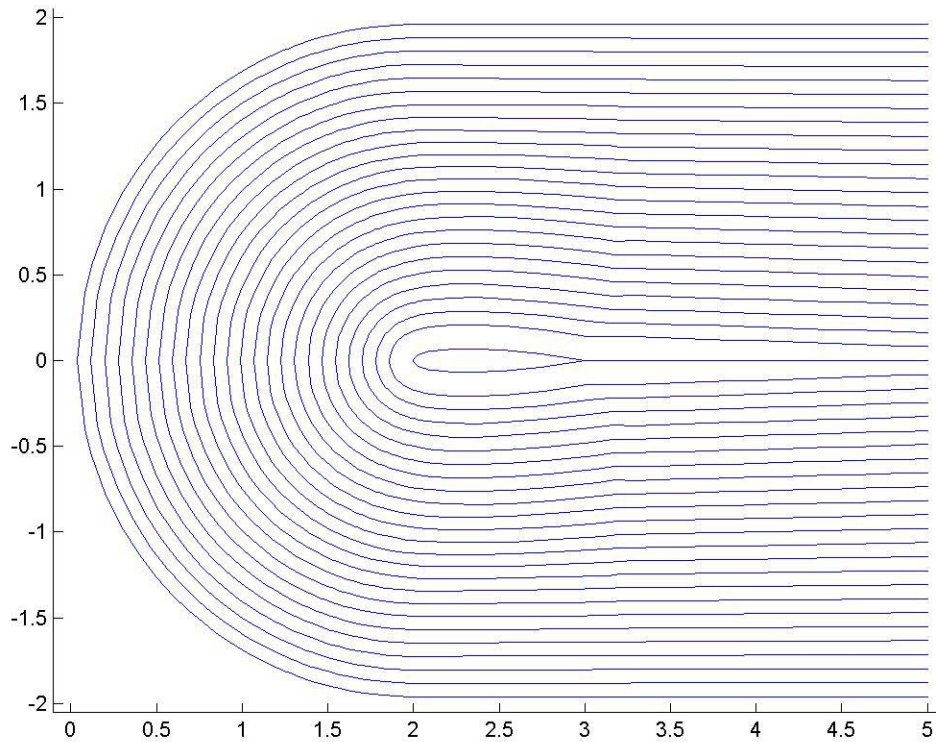


Figure 2. "Circumferential Gridlines"

For the upper half, the points in the arrays with the second dimension index equaling one is along the upper airfoil surface and the *Trailing Edge Line*, while for the lower half, the points are along the lower boundary. Increasing the second dimension index on the upper half of the grid, and holding the other indices constant describes

points moving up from the upper airfoil surface to the upper boundary. For the lower half, increasing the second dimension index describes points moving up from the lower boundary to the lower airfoil surface and the *Trailing Edge Line*. For the purposes of this report, these lines are referred to as “radial gridlines,” and are shown in Figure 3.

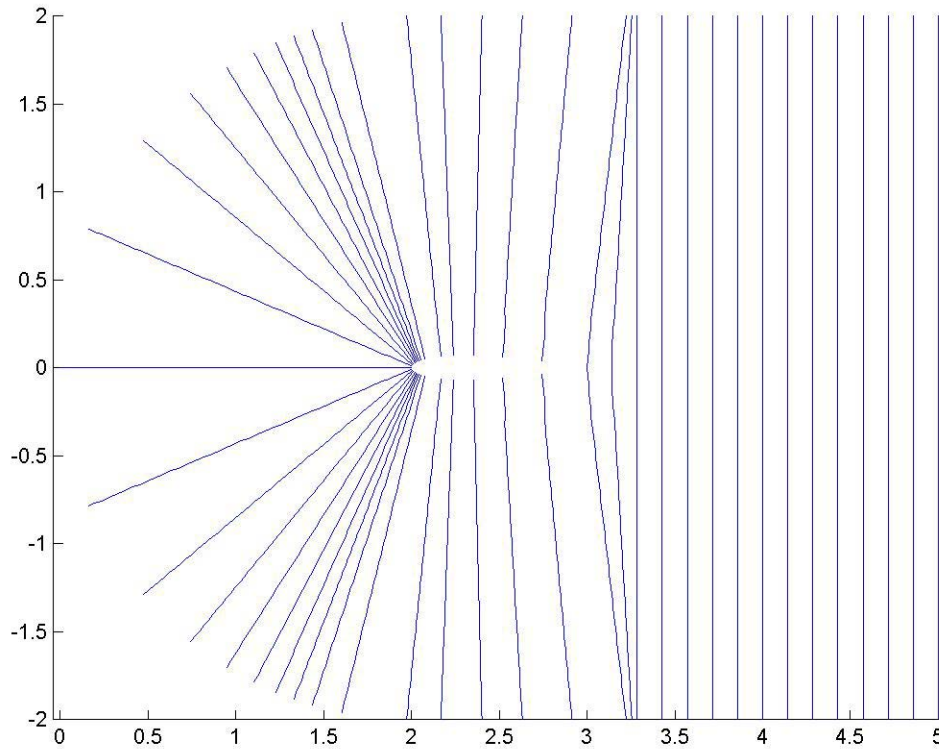


Figure 3. “Radial Gridlines”

Because most algebraic grid generators have problems with the discontinuity in the first derivative at the trailing edge, Astro Grid places a fixed “*Rigid Surface*” around the airfoil. Smoothing is done within the *Rigid Surface*, the size of which is specified by the user. The *Rigid Surface* then simply rotates and plunges with the airfoil, and as a result, the smoothing effects are unchanged. Not only is grid distortion reduced and overlap prevented, computation time is minimized since the *Rigid Surface* is a one-time calculation.

In addition to the *Rigid Surface*, Astro Grid also often makes use of the features of a hyperbolic tangent distribution. As one can see in Figure 4, the hyperbolic tangent curve is flat on each end, and sloped in the middle. When applied to a grid to distribute the deltas between grid boundaries, the function works well since the slope of lines connecting the boundaries do not change immediately. This is due to the flatness of the ends of the hyperbolic tangent curve.

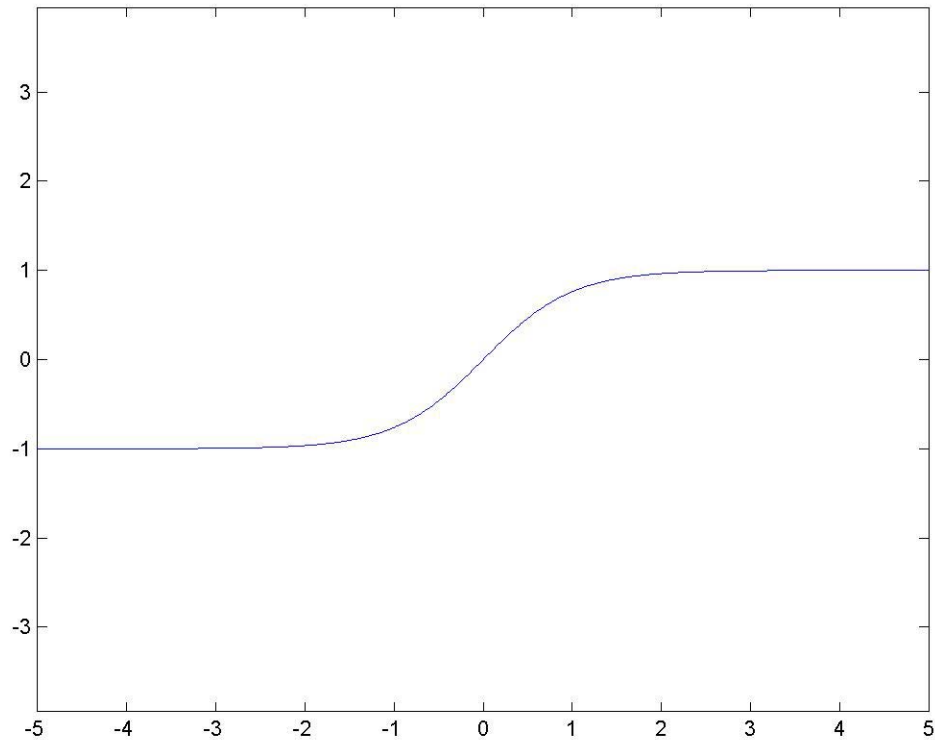


Figure 4. Hyperbolic Tangent Curve

B. USER-CONTROLLED CONSTANTS

As mentioned in the introduction, Astro Grid provides the user with control over forty-four constants, each of which affect the shape of the grid. The user sets the constants in MATLAB in a Graphical User Interface format.

1. Type

Type is the variable that determines whether the boundary is rectangular or semi-circular/rectangular. *Type* equal to one selects the semi-circular/rectangular boundary. Any other value selects the rectangular boundary. Each of these boundaries is shown in Figure 5.

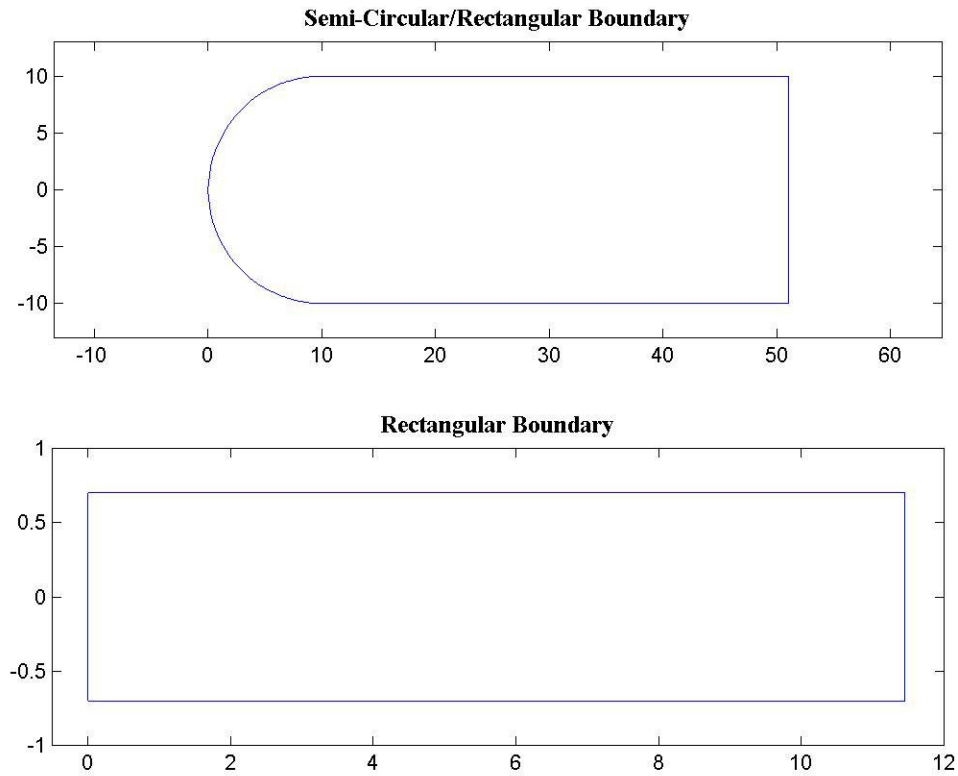


Figure 5. Boundary Types

2. AirfoilType

AirfoilType is the variable that determines which type of airfoil is used. *AirfoilType* equal to one selects the four digit NACA airfoil series. The user is able to enter the four digit NACA airfoil desired. *AirfoilType* equal to two selects the rounded trailing edge flat plate. In this case, the user is able to select the percent thickness of the flat plate. Finally, *AirfoilType* equal to three allows the user to specify a file from which

Astro Grid reads-in the airfoils coordinates and the values of the normal slope at each point

3. **xB, yB**

xB and yB are vectors that determine the overall size of the boundary. xB is the boundary's x-coordinates, while yB is the boundary's y-coordinates. For both vectors, the first entry is the airfoil's leading edge, and the second entry is the point on the right boundary on the *Trailing Edge Line*. The third and final entry is the top point on the right boundary. The point on the left boundary on the *Leading Edge Line* is the grid's origin. The boundaries of the upper and lower halves of the grid are assumed to be symmetrical.

4. **CHORD**

CHORD is the airfoil's chord length. The airfoil is initially placed at zero angle of attack. Therefore, the y-coordinate of the trailing edge is equal to the y-coordinate of the leading edge. The x-coordinate of the trailing edge is the x-coordinate of the leading edge plus the airfoil's chord length.

5. **r**

r is the variable that determines the semi-circle's radius if *Type* is equal to one, or the semi-height of the rectangular boundary otherwise.

6. **m**

m is a vector that contains the number of points in the grid. The first entry is the number of radial gridlines in the upper half of the grid, while the second entry is the number of circumferential gridlines in the upper half. The third and fourth entries contain the lower half's number of radial and circumferential gridlines respectively.

7. **NACA4**

NACA4 is the variable that contains the four-digit NACA airfoil code if *AirfoilType* is equal to one. The first digit is the maximum camber in hundredths of chord. The second digit is the position of maximum camber in tenths of chord. The third and fourth digits are the airfoil thickness in hundredths of chord. If *AirfoilType* is equal to two, *NACA4* contains the percent thickness of the flat plate. The function InitialData.m decodes *NACA4* to properly separate the amount and position of maximum

camber and percent thickness in either case. These values are stored as the variables *MaxCamber*, *PosMaxCamber*, and *Thickness*. If *AirfoilType* is equal to three, *NACA4* stores the location of the file contain the coordinates and normal slopes of the airfoil to be read-in.

8. *thick1, thick2, thick3*

thick1, *thick2*, and *thick3* are the variables that specify the cell heights along the row adjacent to the airfoil surface at the leading edge, airfoil midpoint, and trailing edge respectively on both the upper and lower surfaces of the airfoil. The cell height for the entire row is then linearly varied along the airfoil surface. The specific technique is described later when the function *UpperRigid.m* is discussed. These variables are needed since most viscous flow solvers need the exact cell height in order to perform their computations.

9. *SRigid*

SRigid is the approximate size of the *Rigid Surface* in percent chord. *SRigid* is used in *AirfoilWakeInit.m*, *LeftSurfaceInit.m*, and *RectLeftSurfaceInit.m*.

10. *sizesmooth*

Because of the discontinuity in the first derivative at the trailing edge, smoothing within the *Rigid Surface* is required to prevent gridline overlap. For the remainder of this report, the area where smoothing is performed is referred to as the “*Smoothing Region*”. *sizesmooth* controls the size of this *Smoothing Region* in percent chord. The size of the *Smoothing Region* is measured from the airfoil’s trailing edge forward. As long as the second derivative of the airfoil’s slope within the *Smoothing Region* is approximately zero, the grid will be well behaved. If the *Smoothing Region* is too large, and the second derivative of the airfoil’s slope is non-zero within the *Smoothing Region*, assumptions break down and the grid will become distorted.

11. *bendslopebot, bendslopetop*

These constants, which may vary from zero to one, control the degree with which the normal is enforced within the *Smoothing Region* both on the surface of the airfoil, and

the top of the *Rigid Surface*. The lower the number, the more quickly the slope will change. Conversely, the larger the number, the longer the normal is enforced.

12. **aoa**

aoa is the airfoil angle of attack in degrees.

13. **xo, yo**

xo and *yo* are the variables that determine the airfoil's center of rotation. The coordinates are given relative to the grid's origin, which is the point on the left boundary on the *Leading Edge Line*.

14. **plunge**

plunge is the constant that determines the amount the airfoil plunges. The constant represents an absolute distance since the y-coordinates of the airfoil and *Rigid Surface* are simply incremented or decremented by the size of the variable *plunge*.

15. **RBPitch, LBPitch, RBPlunge, LBPlunge**

These constants control the degree with which the right and left boundaries are affected by airfoil pitch and plunge motion. Specifically, the constants represent the amount the points on the left and right boundaries on the *Trailing* and *Leading Edge Lines* respectively move up or down in percent of natural motion. The natural motion is the motion that would result if the horizontal lines connected to the airfoil's chord line were rigid. For example, if *RBPitch* is set to zero and the airfoil pitches, the *Trailing Edge Line* would remain fixed to the center of the right boundary. Similarly, if *LBPlunge* is set to one and the airfoil plunges upward by 0.5, the point on the left boundary on the *Leading Edge Line* would also be raised upward by 0.5.

16. **obendslopebot, obendslopetop**

obendslopebot and *obendslopetop* are similar to the variables *bendslopebot* and *bendslopetop* with the exception that they control the degree with which normals are enforced at the outer edge of the *Rigid Surface* and the edge of the boundary respectively.

17. NumLowerSeg, NumUpperSeg

These constants determine how many segments the lower and upper airfoil surfaces should be divided into. The ability break the airfoil into segments provides the user with excellent control over the distribution of points. For example, if a transonic case were being examined, the user may wish to pack extra points near the position on the airfoil where he expects a shock. This code gives the user that ability. An example of this is shown in Figure 6.

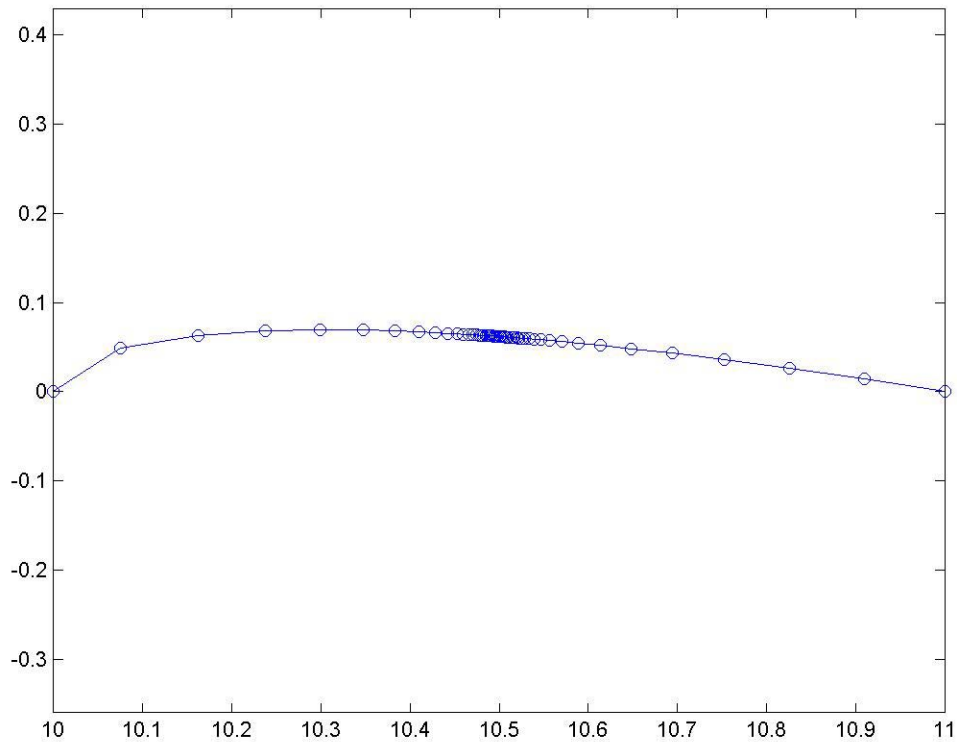


Figure 6. Airfoil Point Distribution

18. AWP, AWQ, LAP, LAQ, LRP, LRQ, LSP, LSQ, UAP, UAQ, URP, URQ

These constants control the distribution of points along each of the boundary surfaces in a manner described later in this chapter. The user sets the constants in an

interactive format. *AWP*, and *AWQ* control the distribution of points along the *Trailing Edge Line*. *LAP*, *LAQ*, *UAP*, and *UAQ* are vectors that control the distribution of points on the lower and upper airfoil surfaces respectively. The vectors have as many rows as there are segments on the lower/upper airfoil. *LRP*, *LRQ*, *URP*, and *URQ* distribute points on the lower and upper right vertical boundaries respectively. Finally, *LSP* and *LSQ* distribute points *Leading Edge Line*.

19. LANUM, UANUM

These variables are vectors that have as many rows as there are segments on the lower and upper airfoil surfaces respectively. The values contained within the vectors determine how many points are allotted for each individual segment on the airfoil surfaces.

20. Dir, LDir

These constants are vectors that have as many rows as there are segments on the upper and lower airfoil surfaces respectively. The values determine the direction of tight grid spacing on each of the segments on the airfoil surfaces. *Dir* is a vector that controls the grid spacing for each segment on the upper airfoil surface, while *LDir* controls the grid spacing on the lower airfoil surface. Values of two place the tightly spaced points to the right, while any other value space the points tightly to the left.

21. EdgeLowerSeg, EdgeUpperSeg

These constants are vectors that have as many rows as there are segments on the upper and lower airfoil surfaces respectively. Their values determine the airfoil location in percent chord where each segment on the airfoil surface ends. If the length of the vector is one, the value defaults to 1.0.

C. UTILITY FUNCTIONS

1. AirfoilReadIn.m

AirfoilReadIn.m is called when the coordinates of an airfoil are read from a file. The first line of the file containing the coordinates should contain the total number of points specified on the upper and lower airfoil surfaces. The next and remaining rows should contain two columns. The first column should be an x-coordinate of the airfoil, while the second column should be the y-coordinate. The first of these two column rows

should contain the coordinates of the trailing edge point. The next row should contain the x and y-coordinates of the point on the lower surface forward and adjacent to the trailing edge. Each successive row should then contain the next point moving forward on the lower surface of the airfoil, wrapping around to the upper surface, and continuing aft until the trailing edge point is reached again. The last point should be the trailing edge point repeated. The next rows, which also should have two columns, should contain the normalized x and y-coordinates of the vector describing the normal slope at each point on the airfoil. The order of the normal slopes should be the same as the coordinates of the points. AirfoilReadIn.m reads these points in to for subsequent grid generation.

2. ArcLength.m

ArcLength.m is called by the functions UpperHorizontal.m, UpperHorizontalLin.m, LowerHorizontal.m, and LowerHorizontalLin.m. ArcLength.m approximates the arc length along both the *Leading Edge Line*, and the portion of the vertical right boundary on the upper or lower half of the grid.

ArcLength.m calculates the arc length with a loop that is repeated once for each point on the gridline of interest. The first entry in the arc length vector, S , is zero. The next entry is the linear approximation of arc length between the first and second points. The third entry is the linear approximation of arc length between the first and third points and so on.

3. ArcLengthBottom.m, ArcLengthTop.m

ArcLengthBottom.m and ArcLengthTop.m are called by the functions LowerBot.m and UpperTop.m and compute the exact arc length along the upper boundary. ArcLengthBottom.m and ArcLengthTop.m are only used with the semi-circular/rectangular boundary.

The functions are simply a loop that is repeated once for each point along the bottom or top boundary. The first entry to the arc length vector, $ALBot$ or $ALTop$, is zero. The next entry is the exact arc length between the first and second points computing from left to right. The third entry is the exact arc length between the first and third points and so on.

The functions make the arc length calculations by first testing if the x-coordinate of the current point and the previous point are greater than the radius of the semi-circle. If so, the arc length is simply the difference between the x-coordinates. If the current point is less than the semi-circle radius, the arc length is calculated with the following equation.

$$s = R\alpha \quad \text{where} \quad \begin{array}{l} s = \text{arc length} \\ R = \text{radius of circle} \\ \alpha = \text{arc angle between current and previous point} \end{array}$$

$$s = R \left(\tan^{-1} \left(\frac{y(i)}{R - x(i)} \right) - \tan^{-1} \left(\frac{y(i-1)}{R - x(i-1)} \right) \right) \quad \text{where} \quad \begin{array}{l} i = \text{current point} \\ i - 1 = \text{previous point} \end{array}$$

Equation 1. Arc Length Calculation

If the current point is greater and the previous point is less than the semi-circle radius, a combination of the above calculations is used.

4. Duplicate.m

Since both the *Leading* and *Trailing Edge Line* are included in both the upper and lower half of the *X* and *Y* arrays, Duplicate.m is used to ensure all the points along these gridlines are coincident in both sectors. The function simply copies the points calculated for the upper half to the lower half.

5. Metric.m

The function Metric.m was taken from the Navier-Stokes flow solver source code. The function is used to check the quality of the grid when it is complete. The function takes the difference of the x and y-coordinates, performs some other manipulations of the x and y-coordinates and takes their jacobian. The result is that if there are no negative values within the jacobian, the grid should be well behaved.

6. Naca.m

The function Naca.m is called by the functions UpperAirfoil.m, and LowerAirfoil.m. Naca.m computes the shape of a NACA four-digit series airfoil. Using the user specified NACA four-digit series airfoil, the equation for the NACA airfoil is applied. Conditional statements are used to ensure the correct equation is being used

depending on the location of the maximum camber and whether the upper or lower surface is being computed.

7. NacaSlope.m

NacaSlope.m is called by the functions WakeUpper.m, and WakeLower.m. NacaSlope.m computes the slope of the line perpendicular to the airfoil surface at each grid point along the airfoil surface. The exact derivative of the equation for the NACA four-digit airfoil with respect to the x-coordinate was derived analytically and is applied to x-coordinate of interest. The normal slope is then calculated by taking the negative reciprocal of the exact slope.

Once again, conditional statements are used to ensure the correct equation is being used depending on the location of the maximum camber and whether the normal slopes on the upper or lower half are being computed. Additionally, a check to prevent a divide by zero error is used since both the derivative equation and the conversion from slope to normal slope involves a division.

8. RTESlope.m

RTESlope.m calculates the slope of the airfoil if *AirfoilType* is equal to two. The function is called by the same functions as NacaSlope.m and operates in the same manner with the only difference being the shape of the airfoil.

9. RectArcLengthBot.m, RectArcLengthTop.m

The functions RectLowerBot.m and RectUpperTop.m call the functions RectArcLengthBot.m and RectArcLengthTop.m respectively. RectArcLengthBot.m and RectArcLengthTop.m compute the exact arc length along the lower and upper boundaries. These functions are only used when the rectangular boundary is selected. The calculation of arc length is a straightforward application of the Pythagorean Theorem since the entire boundary is linear.

10. RotateClockwise.m

The RotateClockwise.m function is called by the function PitchPlungeRigid.m, and is used to rotate the entire *Rigid Surface* by the desired angle of attack. A rotation matrix is computed, and the resulting, rotated *Rigid Surface* results from matrix

multiplication of the original *Rigid Surface* with the rotation matrix. The user has control of the variables that determine the center of rotation.

11. Stretch.m

Stretch.m is a one-dimensional stretching function developed by Roberts (1971) and modified by Eiseman (1979)¹. It accepts two variables as inputs, P and Q . The user is given control of these values, which effectively provides control of the grid point spacing along each of the boundary surfaces. P controls the spacing of the points on the tightly spaced side, while Q controls the spacing on the other side.

12. UpperSurfaceNormal.m

For the case of a semi-circular/rectangular boundary, UpperSurfaceNormal.m is called by LeftSurface.m. UpperSurfaceNormal.m calculates the normal slope of the upper boundary at a selected point. The equation used is simply the derivative of the equation of a circle with respect to the x-coordinate if the x-coordinate is less than the radius of the circle. Otherwise, an infinite slope, which is the normal slope of a horizontal line, is approximated by a value of one million.

D. FUNCTIONS GENERATING GRIDLINES WITH USER-CONTROLLED DISTRIBUTIONS

1. AirfoilWakeInit.m, AirfoilWake.m

The functions AirfoilWakeInit.m and AirfoilWake.m distribute points along the *Trailing Edge Line*. AirfoilWakeInit.m is called only once and sets the initial distribution with the function Stretch.m. Once the distribution is determined, the function scales the distribution to fit on the *Trailing Edge Line*. The results are the x-coordinates for the *Trailing Edge Line*.

Next, the location of the right edge of the *Rigid Surface* is determined with the constant $SRigid$. The value of the first dimension index in the X and Y arrays that corresponds to the right edge of the *Rigid Surface* is named $EDGER$. The y-coordinates are then set equal to the y-coordinate of the airfoil's trailing edge since, initially, pitch and plunge are zero.

¹ Fletcher, C.A.J., *Computational Techniques for Fluid Dynamics*, Volume II (Berlin: Springer-Verlag, 1991), 105.

AirfoilWake.m is called for each subsequent grid generation, which may involve pitching and/or plunging. First, the function determines the new intersection of the *Trailing Edge Line* with the right vertical boundary. AirfoilWake.m makes the assumption that the right boundary is a vertical line. As a result, the x-coordinate of the right boundary intersection does not change, and the normal slope is zero. The y-coordinate is the zero motion y-coordinate plus the percentage of motion due to pitch and plunge – which the user controls with the pitch and plunge constants. The resulting y-coordinate is then checked to ensure the intersection is still within the grid boundary. If not, the user is instructed to reduce the pitch and plunge constants.

Next, the function sets the x-coordinates by scaling the portion of the distribution used for the zero motion case not in the *Rigid Surface* on the *Trailing Edge Line*. Then, the function enforces the normal at the right boundary by equating the y-coordinate of the point to the left of the right boundary to the y-coordinate of the point on the right boundary. Once this is complete, a hyperbolic tangent distribution is used to both distribute the y-coordinate delta between the trailing edge and the right boundary, and to smoothly transition from the slope at the right edge of the *Rigid Surface* to the slope at the right boundary.

Specifically, the hyperbolic tangent distributions are implemented as follows. First, the slope at the right edge of the *Rigid Surface* is calculated. The slope is then allowed to vary in a hyperbolic tangent distribution to zero between the right edge of the *Rigid Surface* in approximately 10% of the length of the remaining gridline. For example, the left most point on the *Trailing Edge Line* not in the *Rigid Surface* has a slope equal to the point to the left of it. The next point to the right has a slope slightly less than the previous, and by the time the point 10% of the length of the *Smoothing Region* to the right is reached, the slope is zero.

Line segments, each with the slope just calculated, are then drawn from the right edge of the *Rigid Surface*, with one line segment for each point on the *Trailing Edge Line*. By the time the point where the slope is zero is reached, there is some delta in the y-coordinate from the y-coordinate on the right edge of the *Rigid Surface*. This delta is calculated and added to the actual y-coordinate delta between the two edges of the

Smoothing Region. This new delta is then spread out in a hyperbolic tangent fashion. The end result is a smoothly varying region with matching slopes at each side.

2. **LeftSurfaceInit.m, LeftSurface.m, RectLeftSurface.m**

These functions are very similar to *AirfoilWakeInit.m* and *AirfoilWake.m*. The main difference is that they distribute points on the *Leading Edge Line*. *LeftSurface.m* is called when *Type* is equal to one, and *RectLeftSurface.m* is called otherwise. *LeftSurfaceInit.m* is called regardless to perform one-time calculations.

Like *AirfoilWakeInit.m*, *LeftSurfaceInit.m* assumes an airfoil at zero angle of attack and zero plunge. As a result, the function simply distributes the x-coordinates with the function, *Stretch.m*, and sets the y-coordinates equal to the y-coordinate of the leading edge. For the semi-circular/rectangular boundary, the leading edge must be at a y-coordinate of zero to prevent a boundary error. In addition, the function also calculates the position of the left edge of the *Rigid Surface* and saves the corresponding index of the second dimension of the *X* and *Y* arrays in the variable named *EDGEL*.

RectLeftSurface.m is very similar to *AirfoilWake.m*. *LeftSurface.m*, however, is slightly more complicated because of the presence of the semi-circular/rectangular boundary. First, *LeftSurface.m* computes the intersection of the *Leading Edge Line* and the left boundary. This is accomplished by calculating the y-coordinate based on the values of the pitch and plunge constants. Next, using the equation of the circle, the x-coordinate is determined analytically. This x-coordinate is then passed to the function *UpperSurfaceNormal.m*, which provides the exact slope normal to the circular boundary at the intersection point. The x-coordinates are then redistributed using the same distribution determined in *LeftSurfaceInit.m*, only scaled between the new x-coordinates at the left boundary and the left edge of the *Rigid Surface*.

Next, the normal is enforced at the left boundary by placing the point to the right of the left boundary perpendicular to the semi-circle. The slope determined by *UpperSurfaceNormal.m* is used for this calculation. Finally, both the slopes at each end of the circumferential gridline and the y-coordinate delta between the two sides are

tapered using a hyperbolic tangent distribution. The technique used is identical to that used in `AirfoilWake.m`.

3. **LowerAirfoil.m, LowerAirfoilRTE.m, UpperAirfoil.m, UpperAirfoilRTE.m**

`LowerAirfoil.m` and `UpperAirfoil.m` or `LowerAirfoilRTE.m` and `UpperAirfoilRTE.m`, depending on whether *AirfoilType* is equal one or two respectively, are called only once when embedded in the source code. These functions distribute points along the upper and lower surfaces of the airfoil. To provide adequate control of the point distribution, both the upper and lower surfaces of the airfoil can be broken into three segments. Points can then be distributed in each segment with the `Stretch.m` function. To further improve control, the user has control of the direction the points will be packed. The user simply specifies the number of segments, the percent chord location to separate each of the segments, and the number of points desired within each segment. If *Dir* or *LDir*, which are the constants that determine the spacing direction, is set to two, the points are tightly spaced at the right end of the segment. Otherwise, the points are tightly spaced at the left end of the segment.

Unlike the other functions that call the `Stretch.m` function however, the slope of the airfoil varies from infinity to zero. As a result, to ensure an even distribution of points, the spacing is applied as an arc length instead of as an x or y-coordinate.

Unfortunately in the case where *AirfoilType* is equal to one, because of the complexity of the NACA four-digit series airfoil equation, an analytical solution for the x and y-coordinates as a function of arc length could not be achieved. As a result, a spline was created. First, the function `Naca.m` is called with a vector of x-coordinates with endpoints coincident with the airfoil segment of interest and a spacing of 0.0001. Next, the slope of the line connecting each of the resulting points is calculated. Finally, the arc length spacing determined by the `Stretch.m` function is set by finding the appropriate x and y-coordinates along the spline. To account for the slight error as a result of the spline approximation, the resulting x-coordinate is evaluated with the NACA airfoil function to produce the exact y-coordinate which lies on the airfoil surface.

4. **LowerRightVerticalInit.m, LowerRightVertical.m, UpperRightVerticalInit.m, UpperRightVertical.m**

These functions distribute points on the right vertical boundary. LowerRightVerticalInit.m and UpperRightVerticalInit.m first call the function Stretch.m. The user has the ability to place more tightly spaced points toward the top of the lower right vertical boundary segment, and toward the bottom of the upper right vertical boundary segment by adjusting the appropriate P and Q values. LowerRightVertical.m and UpperRightVertical.m are called after the airfoil and *Rigid Surface* are put in motion. When these functions are called, AirfoilWake.m has already calculated the intersection of the *Trailing Edge Line* and the right boundary. Therefore, LowerRightVertical.m and UpperRightVertical.m simply use the distribution determined in LowerRightVerticalInit.m and UpperRightVerticalInit.m and rescale them to fit the new boundary.

E. **RIGID SURFACE GENERATING FUNCTIONS**

1. **LowerRigid.m, UpperRigid.m**

The functions LowerRigid.m and UpperRigid.m are called only once when embedded in the source code, and place points within the *Rigid Surface*. The functions first set the height of the cells adjacent to the lower or upper airfoil surface by linearly varying the height between *thick1*, *thick2*, and *thick3* along the airfoil surface. Before that can be done however, the functions must find the index of the first dimension of the X and Y arrays that corresponds approximately to the midpoint of the airfoil. This is accomplished by taking the first x-coordinate value that is greater than or equal to the 0.5 chord position when incrementing the first dimension indexes of the X array, with the second dimension index being the value that corresponds to the airfoil surface for the lower or upper half of the grid.

The actual cell height determination is accomplished with a loop that is repeated once for each grid point along the lower or upper airfoil surface. If the grid point for a given iteration of the loop is less than the airfoil midpoint approximation, it is given a cell height that varies linearly from the cell height specified at the leading edge and that

specified at the airfoil midpoint. If not, the cell height is then the linear variation between the height specified at the airfoil midpoint and that specified at the trailing edge.

Once the cell height along the upper airfoil surface is set, the next task *UpperRigid.m* accomplishes is to acquire the slope of the line perpendicular to the upper airfoil surface at each grid point. If *AirfoilType* is equal to one, the function *NacaSlope.m* is used to calculate the normal slope. If *AirfoilType* is equal to two, the function *RTESlope.m* is used. Otherwise, when *AirfoilType* is equal to three, the normal slopes were already acquired during execution of *AirfoilReadIn.m*.

With this information, the functions place the first row of cells along the lower or upper airfoil surface by converting the perpendicular slope to an angle and using simple trigonometry. The end result is cells of a specified height where the normal condition is exactly enforced, with the exception of the trailing edge. To minimize the ill effects of the discontinuity in surfaces at the trailing edge, when *AirfoilType* is equal to one, the code assigns the trailing edge the average between the actual perpendicular slope and the perpendicular slope of the adjacent wake. When *AirfoilType* is equal to two or three, the trailing edge slope is assigned a value of one half that of the point forward of the trailing edge.

Next, the functions place the first row of cells adjacent to the surface of the airfoil wake from the trailing edge, to the edge of the *Rigid Surface*. These cells are relatively easy to place since the airfoil is originally at zero pitch and zero plunge. As a result, the airfoil wake is a straight, perfectly horizontal line aligned with the airfoil's chord at zero angle of attack. The cell height is simply the cell height specified at the trailing edge.

Now that the entire row adjacent to the upper airfoil and wake surface in the upper *Rigid Surface* have been determined, the second row is placed. The cell height for the second and remaining rows in the *Rigid Surface* are initially set by the spacing dictated by the *Leading Edge Line*. Once these cell heights have been determined, the difference in cell heights between the cells adjacent to the airfoil surface and the cell heights just determined is spread out with a hyperbolic tangent distribution. This is done so there is no discontinuity in cell height near the airfoil surface, and at the same time, the

Rigid Surface still maintains a flat shape on top and bottom. The flat shape on top and bottom is necessary to guarantee a well behaved grid outside the *Rigid Surface*.

Using the same perpendicular slope calculated for the first row, with the new cell height just determined, the second row is placed. However, the entire row is not placed. Because of the discontinuity at the trailing edge, the *Rigid Surface* requires smoothing to prevent gridline overlap. The size of the *Smoothing Region* is dictated by the user controlled constant, *sizesmooth*. Therefore, the second row is placed from the leading edge to the left edge of the *Smoothing Region*. The functions then place the same portion of the remaining rows in the *Rigid Surface* in same manner.

The functions next place the points on the bottom or top of the *Smoothing Region*, which is coincident with the bottom or top of the *Rigid Surface*. The functions again use a hyperbolic tangent distribution to smoothly vary from the y-coordinate on the bottom or top of the *Rigid Surface* at the left edge of the *Smoothing Region* and the y-coordinate on the bottom or top of the *Rigid Surface* at the right edge of the *Rigid Surface*. To further improve the shape of the bottom or top of the *Rigid Surface*, the slope along the bottom or top of the *Rigid Surface* at the left side of the *Smoothing Region* is matched to the portion of the gridline immediately to the left. The slope is then varied away from this value with, again, a hyperbolic tangent distribution.

Specifically, the hyperbolic tangent distribution is implemented as follows. First, the slope on the bottom or top of the *Rigid Surface* immediately to the left of the *Smoothing Region* is calculated. The slope is then allowed to vary in a hyperbolic tangent distribution to zero between the left edge of the *Smoothing Region* to 80% of the length of the *Smoothing Region*. For example, the left most point in the *Smoothing Region* on the top of the *Rigid Surface* has a slope equal to the point to the left of it. The next point to the right has a slope slightly less than the previous, and by the time the point 80% of the length of the *Smoothing Region* to the right is reached, the slope is zero.

Line segments, each with the slope just calculated, are then drawn from the left edge of the *Smoothing Region* on the bottom or top of the *Rigid Surface*, with one line segment for each point on the gridline on the bottom or top of the *Smoothing Region*. By

the time the point where the slope equals zero is reached, there is some delta in the y-coordinate from the y-coordinate on the left edge of the *Smoothing Region*. This delta is calculated and added to the actual y-coordinate delta between the two edges of the *Smoothing Region*. This new delta is then spread out in a hyperbolic tangent fashion. The end result is a smoothly varying region with matching slopes at each side. The consequence of this is the elimination of the first order discontinuity at the trailing edge by the edge of the *Rigid Surface*.

Now that the entire gridline along the bottom or top of the *Rigid Surface* has been drawn, the slope along the line is approximated. The functions do this through a linear approximation by dividing the y-coordinate delta by the x-coordinate delta of the points immediately to the right and left of the point of interest.

Finally, the code draws the vertical lines from the lower or upper airfoil and wake surfaces to the bottom or top of the *Rigid Surface*. Here, the x-coordinate delta is varied in a hyperbolic tangent distribution. As before, the slopes are matched at both the lower or upper airfoil and wake surfaces and at the bottom or top of the *Rigid Surface*. A similar method as that described above is used.

Specifically, the y-coordinate spacing for each vertical line in the *Smoothing Region* is chosen to be the same as that of the vertical line at the left edge of the *Smoothing Region*. This method is accurate as long as the second derivative of the airfoil surface in the *Smoothing Region* is approximately zero. If the *Smoothing Region* is chosen too large so this is not the case, the gridlines in the *Rigid Surface* will become distorted.

The main difference between the placement of these vertical lines and the placement of the line on top of the *Smoothing Region* is that the speed with which the slopes at the top and bottom of the vertical lines change is controlled by two user specified constants – *bendslopetop*, and *bendslopebot*. For the horizontal line on the top of the *Smoothing Region*, the slope was given 80% of the length of the *Smoothing Region* to change to zero. Here, the slope is given however long the user wishes. For larger discontinuities at the trailing edge and tighter grid spacing, the normal cannot be enforced

very long. As a result, the user must choose a smaller region for the slope to change. If the constants are chosen so small that there aren't any points with which to change the slope to zero, the constants are transparently changed to approximately their smallest allowable value.

F. OUTER GRIDLINE GENERATING FUNCTIONS

1. LowerBottom.m, RectLowerBottom.m, UpperTop.m, RectUpperTop.m

These functions are called each time a grid is drawn. The purpose of the functions is to distribute the points on the upper and lower boundaries by finding the intersection of the outer boundary gridlines and perpendicular lines emitting from the *Rigid Surface* and its wake extension at each grid point. This technique provides a well-behaved grid outside of the *Rigid Surface*. LowerBottom.m and UpperTop.m are called when the boundary *Type* is equal to one, and RectLowerBottom.m and RectUpperTop.m are called otherwise.

First, these functions use the normal slope of the circumferential gridline along the top or bottom of the *Rigid Surface*. This slope is calculated in the functions WakeLower.m and WakeUpper.m, which will be discussed later. Next, the code analytically determines the intersection of the radial line with the outer boundary. For the semi-circular/rectangular boundary, the equation of the radial line and the equation of the lower or upper boundary are equated. The resulting equation for the x-coordinate at the intersection is shown below as Equation 2.

$$x_2 = \frac{(2R + 2m^2x_1 - 2my_1) \pm \sqrt{R^2 + m(2y_1(x_1 - R) + mx_1(2R - x_1)) - y_1^2}}{(m^2 + 1)}$$

R = boundary circle radius

where m = normal slope at outer edge of rigid surface

(x_1, y_1) = coordinates at outer edge of rigid surface

Equation 2. Upper and Lower Points for Semi-Circular/Rectangular Boundary

A series of conditional statements are then used to determine whether the intersection occurs on the circular portion of the boundary and, if so, whether the larger or smaller solution of the quadratic equation should be used.

Once the points on the lower or upper surface have been placed, they are then checked for the presence of an overlap. When the circumferential gridline along the top or bottom of the *Rigid Surface* has a non-zero second derivative, grid overlap on the lower or upper boundary is likely. To perform this check, the code simply ensures the x-coordinates continually increases along the outer boundary when starting from the leftmost point. If an overlap is detected, the code determines the size of the overlap region, and then linearly spaces the effected points throughout the region.

6. LowerHorizontal.m, UpperHorizontal.m

These functions draw the gridlines from the circumferential gridline along the bottom or top of the *Rigid Surface* to the outer boundary. LowerHorizontal.m and UpperHorizontal.m begin by first distributing the outer gridlines in a linear fashion. The deltas dictated by the outer boundary are also varied linearly to prevent any cell discontinuity with the outer boundary.

Next, the differences in cell height between the cells on each side of the outer edge of the *Rigid Surface* are spread out in a hyperbolic tangent distribution in a manner similar to that used in UpperRigid.m. Finally, the slopes at the outer edge of the *Rigid Surface* are also tapered in a hyperbolic tangent fashion.

7. WakeLower.m, WakeUpper.m

WakeLower.m and WakeUpper.m are called each time a grid is generated. These functions place the points in the wake of the *Rigid Surface*. The functions accomplish this by having WakeUpper.m, which is always called first, calculate the normal slope of the *Trailing Edge Line*.

Next, lines perpendicular to the *Trailing Edge Line* are drawn at each point along the gridline outside of the *Rigid Surface*. The spacing along the perpendicular lines are set by linearly varying the cell height from the right edge of the *Rigid Surface* to the right vertical boundary.

Finally, the slope of the circumferential gridlines on the top and bottom of this wake region is calculated with a linear approximation. The result is then saved for later use in other functions.

G. PITCH AND PLUNGE MOTION FUNCTION

1. PitchPlungeRigid.m

PitchPlungeRigid.m is the function responsible for rotating and translating the *Rigid Surface*. The rotating is accomplished by calling the function RotateClockwise.m, which is described in Chapter II, Section C of this thesis, while the translating is done by simply adding the variable plunge to the y-coordinates of the *Rigid Surface*.

III. USER'S GUIDE

A. INTRODUCTION

This chapter is not intended to be a self-contained user's guide. Chapter I, and Chapter II, Section A of this thesis should be read before continuing. These portions of the thesis contain definitions of terms that are used later in this chapter.

B. INITIAL DATA

Astro Grid is executed by typing Astro Grid at the command line in MATLAB. When this is done, the Initial Data Menu is displayed and is shown in Figure 7.

Astro Grid

Boundary Data		Airfoil Data		Motion Data	
Circular Boundary		NACA 4-Digit Series Airfoil		Degrees Pitch	
Airfoil Leading Edge	X 10.0 Y 0.0	4-Digit NACA Airfoil Designation	0014	Center of Rotation	X 10.25 Y 0.0
Airfoil Chord Length	1.0	Cell Height at Leading Edge	0.00001	Amount of Plunge	1.0
Right Center	X 21.0	Cell Height at Airfoil Midpoint	0.00001	Pitch Constants	Right 1.0 Left 1.0
Boundary Circle Radius	10.0	Cell Height at Airfoil Trailing Edge	0.00001	Plunge Constants	Right 1.0 Left 1.0
Upper Half of Grid		Rigid Surface Data		Outer Boundary Data	
Radial Gridlines	161	Size of Rigid Surface (Percent Chord)	7.0	Enforce Normal Constant at Edge of Rigid Surface	0.2
Circumferential Gridlines	91	Size of Upper Smoothing Region (Percent Chord)	0.7	Enforce Normal Constant at Outer Boundary	0.2
Lower Half of		Size of Lower Smoothing Region (Percent Chord)	0.7		
Radial Gridlines	161	Enforce Normal Constant at Airfoil Surface	0.01		
Circumferential Gridlines	91	Enforce Normal Constant at Outer Edge of Rigid Surface	0.8		

Continue

Figure 7. Initial Data Menu

First, the user inputs the boundary data. The default boundary of Astro Grid is the semi-circular/rectangular boundary. However, the user may select the rectangular boundary through the use of the pull-down menu, which also contains a selection for a rectangular boundary.

Entering the x and y-coordinates of the airfoil's leading edge, with the boundary's origin being the leftmost point on the *Leading Edge Line*, is the first step in setting the boundary data. Next, the airfoil's chord length is entered, noting that the airfoil is initially placed at a zero angle of attack. Then, the x-coordinate of the rightmost point on the *Trailing Edge Line* is entered, while the code assumes the y-coordinate is zero. Finally, the radius of the semi-circle is entered for the semi-circular/rectangular boundary. For the rectangular boundary, the semi-height of the rectangle is entered instead of radius. The boundary resulting from the settings in the Initial Data Menu shown in Figure 7 is shown in Figure 8.

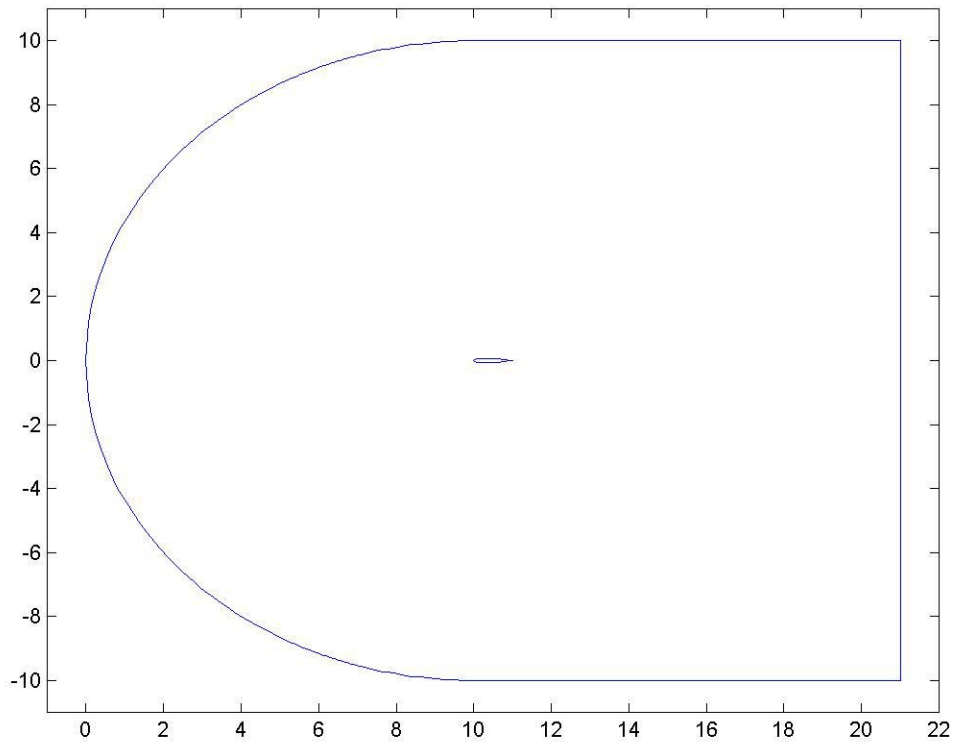


Figure 8. Boundary Data Entry

Next, the numbers of Vertical and circumferential gridlines are entered for both the upper and lower halves of the grid. There is no requirement for the upper and lower halves of the grid to contain the same number of gridlines – as long as both the *Leading* and *Trailing Edge Lines* are allotted the identical number of points for both halves.

The second column of the Initial Data Menu begins with the Airfoil Type pull-down menu. The menu also contains selections for a round leading and trailing edge flat plate, as well as an option to read an arbitrary airfoil from a file. The NACA four-digit airfoil series is the default choice. In this case, the first digit is the maximum camber in hundredths of chord, the second digit is the position of the maximum camber in tenths of chord, while the last two digits are the airfoil's thickness in hundredths of chord. If the flat plate choice is made, the user simply enters the percent thickness. Finally, if the user chooses to use an arbitrary airfoil, he must specify the location of the file containing the airfoil's coordinates and normal slopes.

Next, the user can specify the grid height adjacent to the airfoil's surface at three points along the airfoil – the leading edge, the approximate airfoil midpoint, and the trailing edge. These heights are used for both the top and bottom surfaces of the airfoil. The grid heights of the first row of cells along the upper and lower surfaces are then a linear variation between these specified heights.

Next, the initial *Rigid Surface* data is entered. First, the size of the *Rigid Surface* is entered in chord lengths. The ideal value for the size of the *Rigid Surface* is sensitive to the size of the boundary. For a large boundary, it is often advantageous to select a larger *Rigid Surface* since all points within the *Rigid Surface* are calculated only once. A large *Rigid Surface* thus reduces overall computation time if a grid is calculated at each time step. Often, the ideal size of the *Rigid Surface* is found through trial and error.

The size of the *Smoothing Region* is then chosen. The *Smoothing Region* is where Astro Grid tapers the ill effects of the first order discontinuity between the trailing edge and the *Trailing Edge Line*. Without smoothing, gridline crossing is likely in the vicinity of the trailing edge. The size of the *Smoothing Region* is entered in percent chord and works best for most NACA airfoils at 0.7. This distance is measured from the airfoil's trailing edge forward.

As long as the second derivative of the airfoil's surface is approximately zero within the *Smoothing Region*, the grid will be well behaved. Examples of the effect of the size of the *Smoothing Region* on the appearance of the *Rigid Surface* are shown in Figures 9-11. In these Figures, values of 0.3, 0.95, and 0.7 are shown with all other user controlled constants equal. Figure 9 demonstrates if the size of the *Smoothing Region* is chosen too small, there may be a distortion in the *Rigid Surface* at the edge of the *Smoothing Region*. This effect is clear when one looks along the radial gridline 30% of the chord length in front of the trailing edge.

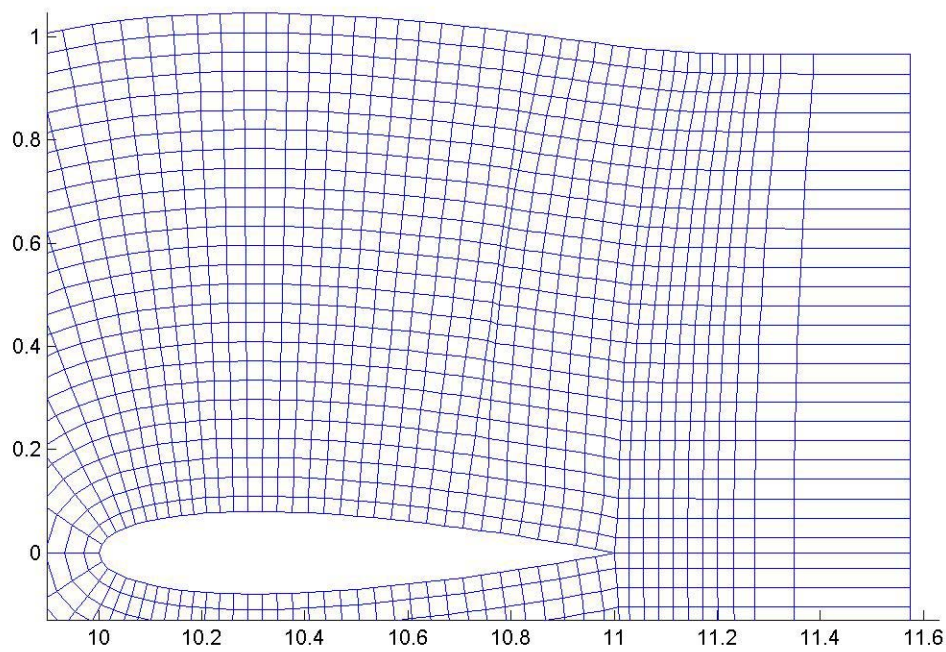


Figure 9. Effect of Size of Smoothing Region on Rigid Surface – 0.3 Chord

If the *Smoothing Region* extends into the portion of the airfoil near the leading edge where the second derivative of the airfoil surface is not approximately zero, severe distortion may result. This is evident in Figure 10, where the *Smoothing Region* extends 95% of the length of the airfoil.

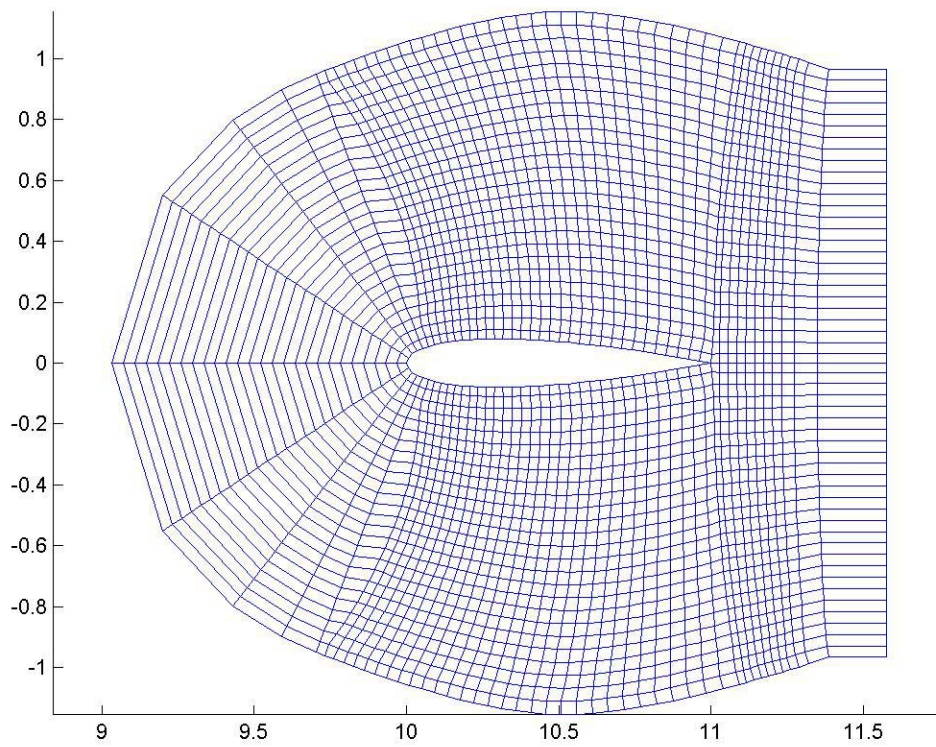


Figure 10. Effect of Size of Smoothing Region on Rigid Surface – 0.95 Chord

Finally, Figure 11 shows the result when the size of the *Smoothing Region* is correctly specified.

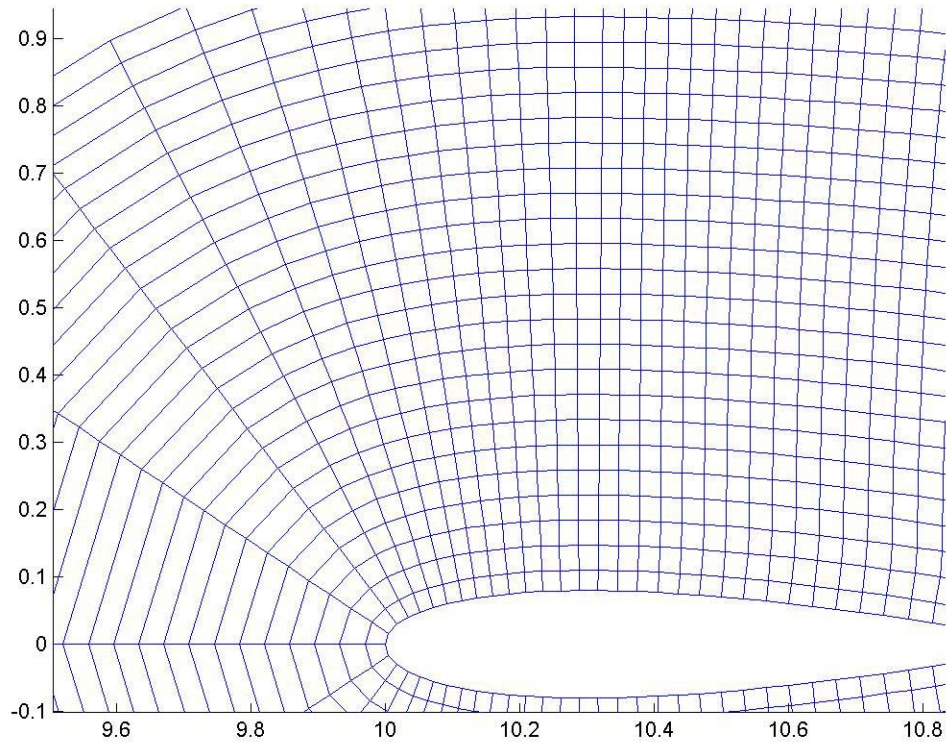


Figure 11. Effect of Size of Smoothing Region on Rigid Surface – 0.7 Chord

After the size of the *Smoothing Region* is chosen, the constants that determine how long the normal is enforced for the top and bottom of the radial gridlines within the *Smoothing Region* are specified. These variables can vary from zero to one, with a lower number allowing the gridlines to “bend” more quickly. Because of the discontinuity at the trailing edge, for tight grid spacing, these constants work best fairly small, but a trial and error approach is often required before the ideal values are found. Examples of the effect of these constants on the appearance of the *Rigid Surface* are shown in Figures 12-13. Figure 12 demonstrates the adverse effects of setting the constants too large. Clearly, gridline overlap has occurred.

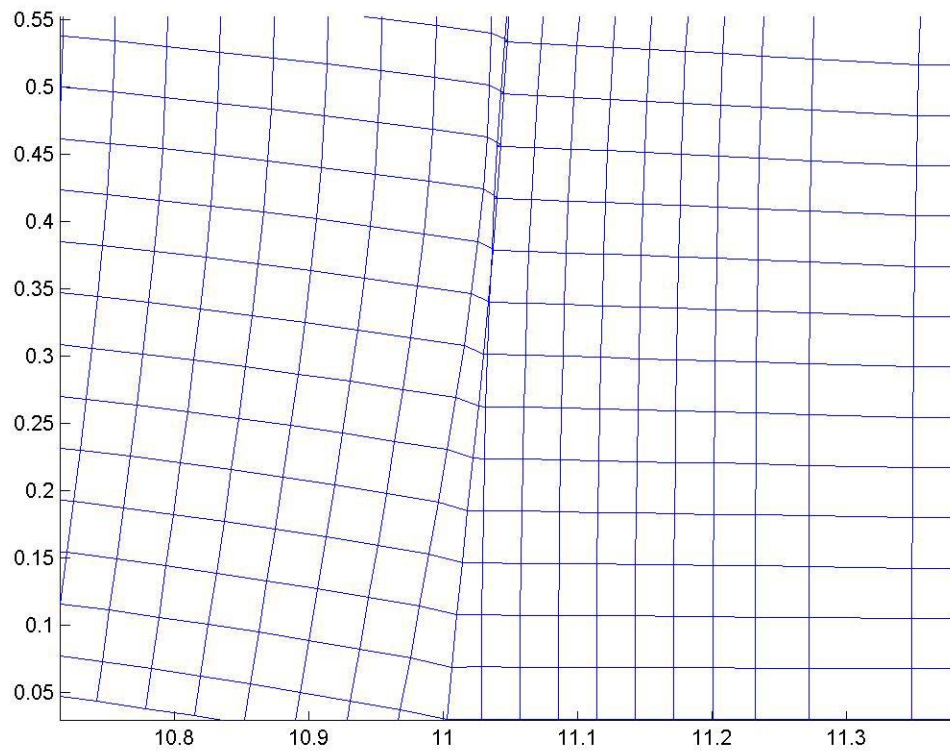


Figure 12. Rigid Surface Enforce Normal Constants – 0.8, 0.8

Figure 13 illustrates the results of correctly specified constants.

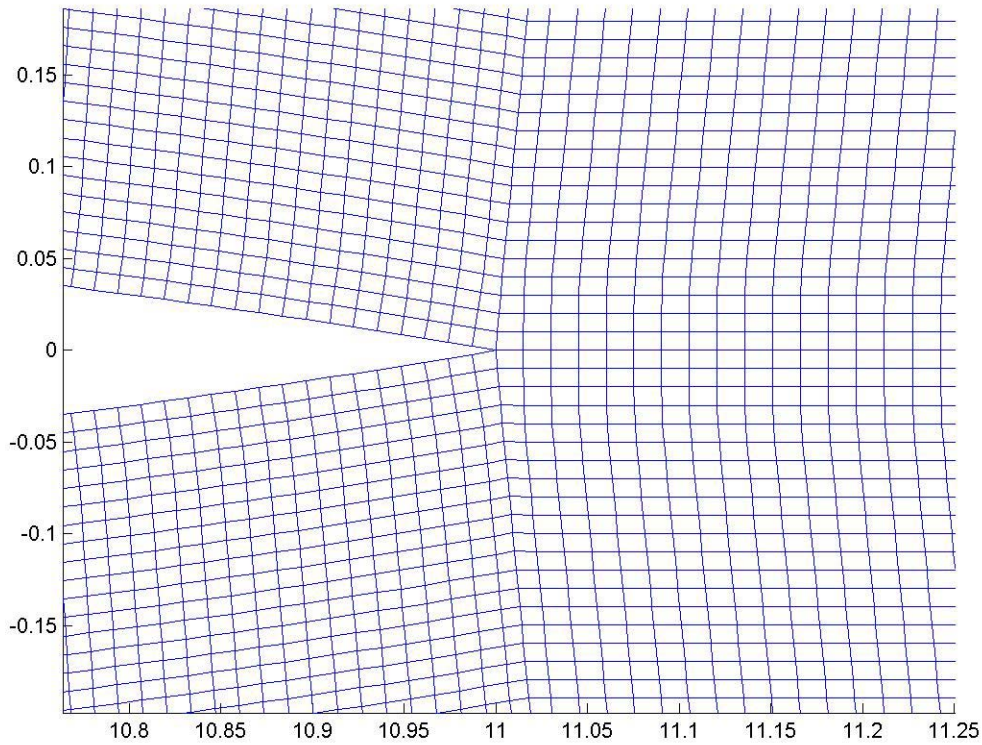


Figure 13. Rigid Surface Enforce Normal Constants – 0.1, 0.1

It is noted that in a separate menu that is discussed later, each of the four constants that control the shape of the grid within the *Rigid Surface* can be changed as many times as necessary after the initial *Rigid Surface* is viewed.

The third column of the Initial Data Menu contains the data that controls how the airfoil is set in motion. Because Astro Grid was designed to draw a grid with reduced computation time, there is nothing done within Astro Grid to prevent or detect gridline overlap or distortion. Astro Grid simply provides the user with the ability to choose several constants that will allow the grid to be drawn to be free from these ill effects. Therefore, it is the user's responsibility to run the MATLAB version of Astro Grid several times to cover the airfoil's entire range of motion to ensure the constants have been chosen correctly and the grid behaves properly. The Airfoil Motion Constants allow

the user to control how the airfoil is set in motion. The user has the ability to vary these constants after viewing the grid resulting from the initial motion. This process is discussed later.

The first constant that is entered is the airfoil's angle of attack in degrees. Positive and negative numbers are accepted. Next, the airfoil's center of rotation is specified. The center of rotation can be any point in the grid and is specified with respect to the grid's origin, which is the leftmost point on the *Leading Edge Line*. After the rotation is specified, airfoil plunge is considered. The maximum plunge can be positive or negative and is actual displacement in the y-coordinate direction. Care must be taken to ensure the plunge amplitude does not push part of the *Rigid Surface* outside the grid boundaries.

The constants that are specified next are the Pitch and Plunge Constants for the left and right boundaries. These constants affect the amount the left and rightmost points of the *Leading Edge Line* and *Trailing Edge Line* respectively, move due to pitch and plunge. They represent the percent of the natural motion of these points along the left and right boundaries. Therefore, a value of one represents natural motion. Figure 14 shows natural motion of the *Trailing Edge Line* due to plunge for a plunge only case.

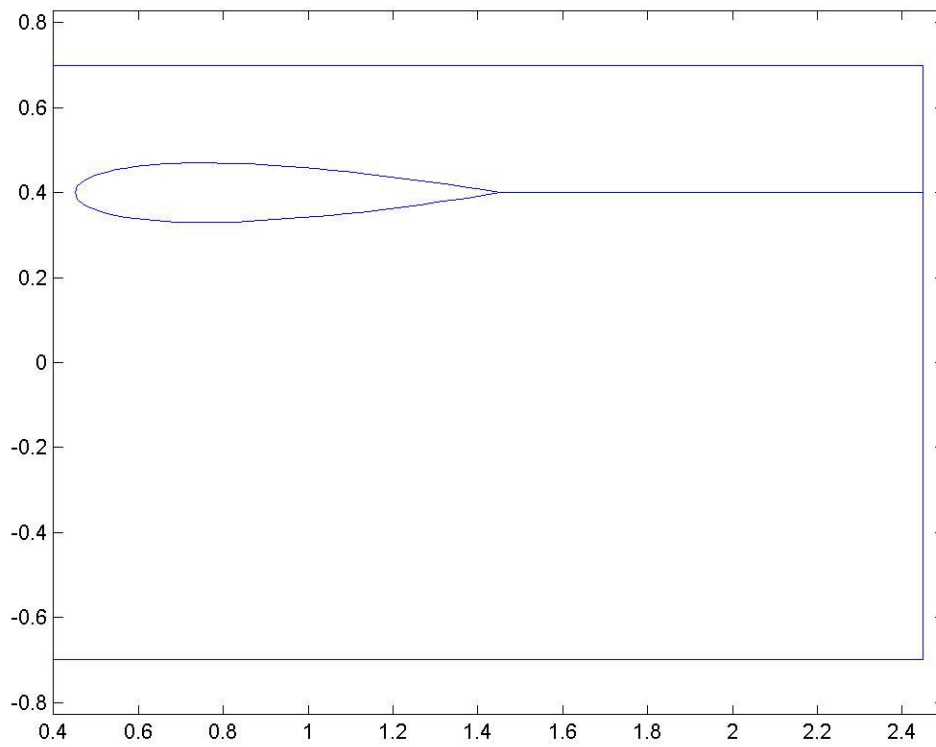


Figure 14. Effect of Right Plunge Constant on Trailing Edge Line – 1.0

A Pitch or Plunge constant of zero would result in no motion along the vertical boundary, as can be seen in Figure 15 for the pitch only case with the center of rotation at the airfoil's quarter chord location.

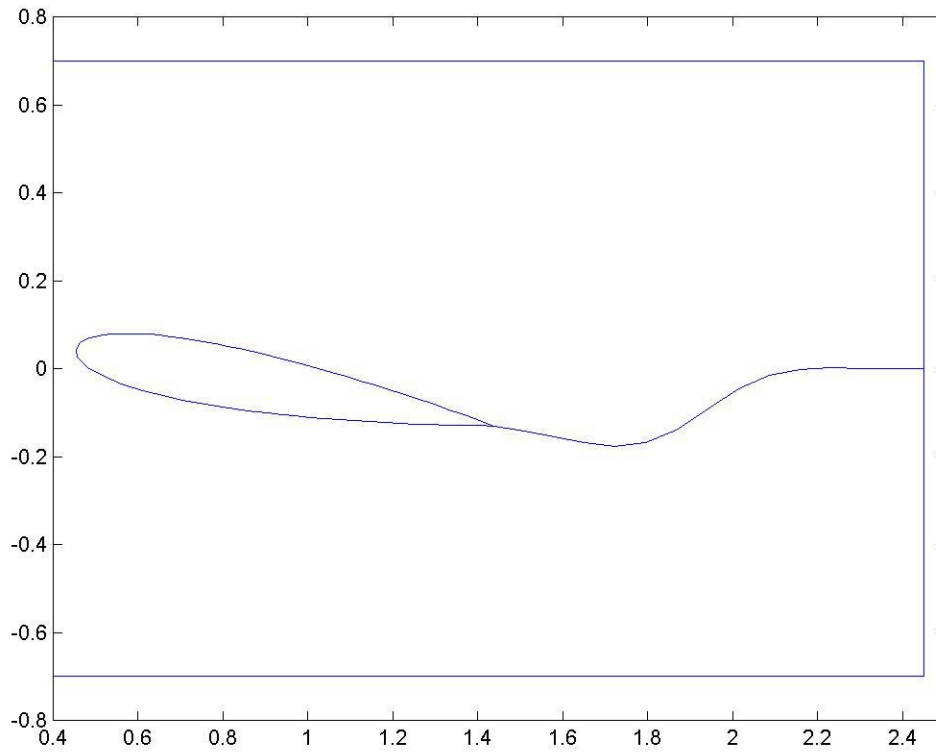


Figure 15. Effect of Right Pitch Constant on Trailing Edge Line – 0.0

The Pitch and Plunge constants chosen are specific to the size of the boundary and maximum amplitudes of airfoil motion. The best value must be chosen on a trial and error basis.

Finally, the last constant on the Initial Data Menu controls the shape of the gridlines connecting the *Rigid Surface* to the outer boundary. The Enforce Normal Constant at the Edge of the Rigid Surface is similar to the Enforce Normal Constants for the *Rigid Surface*, however, this constant controls how quickly the slope is tapered at the edge of the *Rigid Surface*. Once again, the final value of this constant is best chosen through a trial and error process. As with some of the other user-controlled constants, in later menus, the user may alter the value of this constant as many times as necessary.

When all the constants on the Initial Data Menu have been specified, the user should click on the Continue pushbutton. This brings up the menu shown in Figure 16.

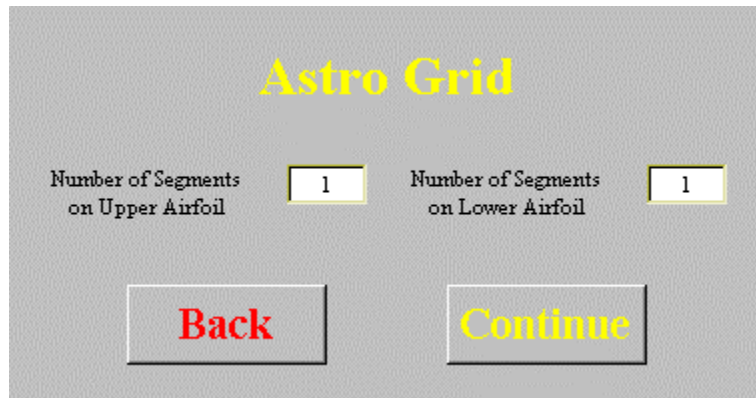


Figure 16. Number of Airfoil Segments Menu Should be on same page as figure

This menu allows the user to specify how many segments to divide the airfoil surfaces into. The maximum number for both the upper and lower surface is three. A description of the process and examples are included in Chapter 2, Section B on page 11 of this thesis.

The user has the option to go back to the Initial Data Menu if needed. Otherwise, the user should click on the Continue pushbutton, which will bring up the P and Q Data Menu. Figure 17 shows the P and Q Data Menu when the user specifies two segments on both the upper and lower airfoil surface.

Astro Grid

Upper Airfoil

Segment 1 Direction

P

Q

Number of Points

Edge in Percent Chord

Segment 2 Direction

P

Q

Number of Points

Lower Airfoil

Segment 1 Direction

P

Q

Number of Points

Edge in Percent Chord

Segment 2 Direction

P

Q

Number of Points

Trailing Edge Line

P

Q

Leading Edge Line

P

Q

Upper Right Boundary

P

Q

Lower Right Boundary

P

Q

Back

Rigid Surface

Continue

Figure 17. P and Q Data Menu – Two Segments Per Airfoil Surface

Figure 17 shows where the user specifies the constants mentioned in Chapter 2, Section B for the upper and lower airfoil surfaces. By assigning the Direction for Segment 1 of the upper airfoil surface a value of two, the points are packed more closely together at the right end of the segment. Conversely, assigning Direction any other value would pack the points more closely at the left end of the segment.

Next, the values of P and Q are chosen. P effectively controls the amount of spacing on the end with tight spacing, while Q effectively controls the spacing at the end with wide spacing. The values of P should be greater than zero and less than or equal to one. The values of Q should be greater than or equal to one. The one dimensional

stretching function has limitations however, and the ideal spacing, again, is best achieved through trial and error.

Before the spacing is viewed however, the number of points in the segment, and the edge of the segment in percent chord is specified. Any number of points may be specified, as long as enough points remain for the other segments on the airfoil surface and the *Trailing Edge Line*. For the edge of the segment, any value between zero and one may be chosen, non-inclusive.

With all of the data for the current segment entered, the distribution for the segment can be viewed by clicking on the View Results pushbutton. A figure will appear with only the appropriate airfoil segment displayed. The user can then make changes if needed, or continue to the next segment.

In a similar manner, the *Trailing Edge Line* P and Q control the point distribution on the *Trailing Edge Line*. By adjusting the P and Q values, the user should attempt to match the spacing on the left side of the *Trailing Edge Line* with the spacing at the right side of the airfoil to provide the most uniform grid.

The *Leading Edge Line* distribution is particularly important for the *Rigid Surface* since the distribution of circumferential gridlines in the *Rigid Surface* is set by the distribution on the *Leading Edge Line*. One can best see this effect in Figure 11.

The Upper and Lower Right Boundary distributions are set in the same exact manner. When each of the distributions have been specified, the user may view the *Rigid Surface* by clicking on the View *Rigid Surface* pushbutton at the bottom of the menu. This allows the user to view all of the gridlines within the *Rigid Surface*. The user should note it is not necessary to view each of the individual distributions before viewing the *Rigid Surface*. After viewing the *Rigid Surface*, the user can make changes to the distributions on the same menu, or change the *Rigid Surface* data by clicking on the Adjust *Rigid Surface* pushbutton. Doing so opens the Adjust *Rigid Surface* Data Menu shown in Figure 18. After making the changes, the user should click on the Continue pushbutton, and then click the View *Rigid Surface* pushbutton to see the changes.

Astro Grid

Size of Rigid Surface (Percent Chord)	7	Enforce Normal Constant at Airfoil Surface	0.01
Size of Upper Smoothing Region (Percent Chord)	0.7	Enforce Normal Constant at Top of Rigid Surface	0.8
Size of Lower Smoothing Region (Percent Chord)	0.7	N/A	-

Continue

Figure 18. Adjust Rigid Surface Data Menu

The pull-down menu in Figure 18 is used to switch between the automatic and manual selection of the intersection of the radial gridline emitting from the trailing edge and the *Rigid Surface*. This selection is only possible if the flat plate, or arbitrary airfoil read from a file is chosen. The default is automatic, but the user has the ability to move this point, which can dramatically improve the quality of the grid.

When satisfied with the appearance of the *Rigid Surface*, the user should click on the Continue pushbutton of the P and Q Data Menu, which opens the Outer Grid Menu, shown in Figure 19.

Astro Grid

Back

Outer Grid

View

Adjust

Done

Continue

Figure 19. Outer Grid Menu

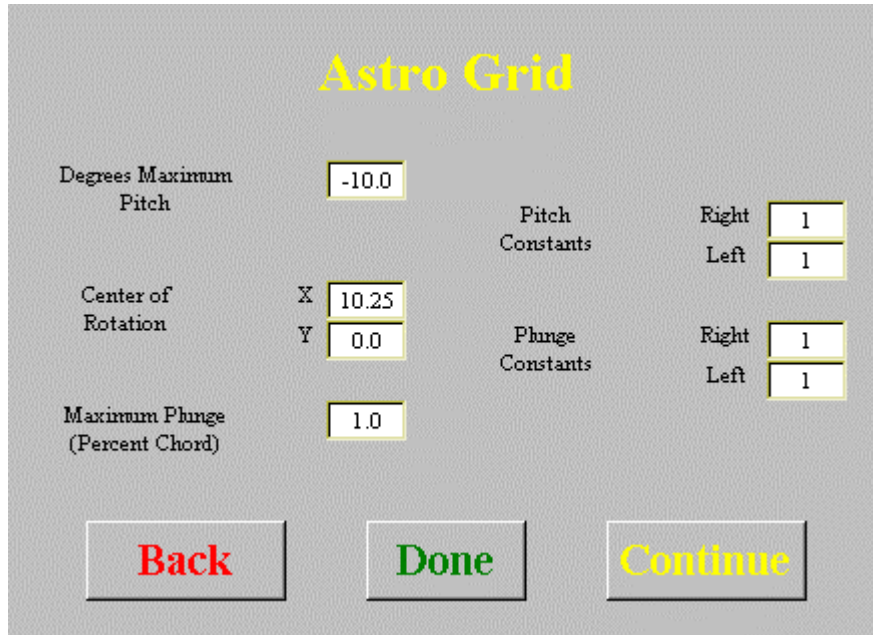
By clicking on the view pushbutton, the user is shown the entire grid with zero airfoil motion. If not satisfied with the grid's appearance, the user can click on Back to make changes to previously entered constants, or click on the Adjust pushbutton, which opens the Adjust Outer Grid Data Menu shown in Figure 20.



Figure 20. Adjust Outer Grid Data Menu

When satisfied with the appearance of the entire grid without any motion, the user can click the Done pushbutton to create the file to be read by the version of Astro Grid embedded in the flow solver source code, to execute the Metric function to determine the quality of the grid, and to write the coordinates of the grid to a file. This should only be done at this point if no airfoil motion is needed. Otherwise, the user should click on the Continue pushbutton to view the grid with the initial airfoil motion.

Next, the Adjust Motion Menu appears, which is shown in Figure 21.



The image shows a software interface titled "Astro Grid" in yellow text. It contains several input fields for motion parameters. On the left, "Degrees Maximum Pitch" is set to -10.0, "Center of Rotation" has X=10.25 and Y=0.0, and "Maximum Plunge (Percent Chord)" is set to 1.0. On the right, "Pitch Constants" shows Right=1 and Left=1, and "Plunge Constants" also shows Right=1 and Left=1. At the bottom are three buttons: "Back" in red, "Done" in green, and "Continue" in yellow.

Parameter	Value
Degrees Maximum Pitch	-10.0
Center of Rotation X	10.25
Center of Rotation Y	0.0
Maximum Plunge (Percent Chord)	1.0
Pitch Constants Right	1
Pitch Constants Left	1
Plunge Constants Right	1
Plunge Constants Left	1

Buttons: Back, Done, Continue

Figure 21. Adjust Motion Menu

To view the changes, the user should click on the Continue pushbutton, which will display the entire grid with the new motion parameters. Once again, the Adjust Motion Menu will reappear. The user can continue to view the grid with different airfoil motion as many times as necessary, or click on Done to create the file for the flow solver source code, execute the Metric function, and write the coordinates to a file.

IV. CONCLUSIONS

Overall, Astro Grid accomplishes each of its objectives. The code produces a grid free from gridline overlap with minimal distortion. In addition, it accomplishes this without a large amount of computer processing power.

However, work still remains. First, Astro Grid must be translated to Fortran and embedded in the flow solver. This could produce a potential problem if a rectangular grid is used. Some flow solvers expect the points on the left boundary before the airfoil is set in motion, to remain on the left boundary. Astro Grid does not do this. To maintain a satisfactory grid, Astro Grid allows the points on the left boundary to wrap around to the upper boundary. In order for the flow solver to work in this case, the flow solver code must be altered.

In addition, Astro Grid is expected to increase computation time above the existing grid generators slightly. The quantitative amount is unknown at this time. Therefore, processing time analysis is recommended.

As a result, this thesis was written in a thorough manner so others can understand the theory behind the code, and make changes to meet their specific needs, if necessary.

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX A

A. UTILITY FUNCTIONS

1. AirfoilReadIn.m

```
function AirfoilReadIn

load InitData
load Results

a = gcf;
FileLocation = get(findobj(a, 'Tag', 'NACA'), 'String');

FID = fopen(FileLocation, 'r');

temp = fscanf(FID, '%5d', 1);
UANUM = (temp - 1) / 2;
LANUM = UANUM;
NUM = UANUM;
NUM2 = LANUM;

close
EnterPQFile

a = gcf;
set(findobj(a, 'Tag', 'UANUM'), 'String', UANUM);
set(findobj(a, 'Tag', 'LANUM'), 'String', LANUM);

%READ-IN LOWER AIRFOIL
SECT = 2;
for i = 1:LANUM+1
    X(LANUM+2-i, m(2*SECT), SECT) = fscanf(FID, '%f', 1) + xB(1);
    Y(LANUM+2-i, m(2*SECT), SECT) = fscanf(FID, '%f', 1);
end

%READ-IN UPPER AIRFOIL
SECT = 1;
X(1, 1, SECT) = X(1, m(4), 2);
Y(1, 1, SECT) = Y(1, m(4), 2);

for i = 2:UANUM+1
    X(i, 1, SECT) = fscanf(FID, '%f', 1) + xB(1);
    Y(i, 1, SECT) = fscanf(FID, '%f', 1);
end

%READ-IN LOWER AIRFOIL NORMAL SLOPES
for i = 1:LANUM+1
    x = fscanf(FID, '%f', 1);
    z = fscanf(FID, '%f', 1);
    if x ~= 0
```

```

        nslope(LANUM+2-i) = z / x;
    else
        if z > 0
            nslope(LANUM+2-i) = 1e6;
        else
            nslope(LANUM+2-i) = -1e6;
        end
    end
end
end

save ReadInLower nslope

%READ-IN UPPER AIRFOIL NORMAL SLOPES
for i = 2:UANUM+1
    x = fscanf(FID,'%f',1);
    z = fscanf(FID,'%f',1);
    if x ~= 0
        nslope(i) = z / x;
    else
        if z > 0
            nslope(i) = -1e6;

            nslope(i) = 1e6;
        end
    end
end
end

save ReadInUpper nslope

save PQData UANUM LANUM FileLocation -APPEND
save Intermediate NUM NUM2
save Results X Y

```

2. ArcLength.m

```

function [S] = ArcLength(X1, Y1, SIZE)
%FUNCTION COMPUTES ARC LENGTH ALONG SURFACE DESCRIBED BY
% X1, Y1 - EACH OF LENGTH SIZE
% FIRST ENTRY IS ZERO, SECOND ENTRY IS DISTANCE BETWEEN FIRST AND
% SECOND POINTS,
% THIRD ENTRY IS DISTANCE BETWEEN FIRST AND THIRD POINTS
% AS MEASURED ALONG SURFACE ETC.

S(1) = 0;
for i = 2:SIZE %PERFORM (ONCE - 1) FOR EACH POINT ON SURFACE
    S(i) = S(i-1) + sqrt( (X1(i) - X1(i-1)) * (X1(i) - X1(i-1)) + ...
        (Y1(i) - Y1(i-1)) * (Y1(i) - Y1(i-1)) );
end

```

3. ArcLengthBottom.m

```

function ArcLengthBottom
%FUNCTION COMPUTES EXACT ARC LENGTH ALONG LOWER SURFACE WITH RADIUS r
% DESCRIBED BY

```

```

% X1, Y1 - EACH OF LENGTH SIZE
% FIRST ENTRY IS ZERO, SECOND ENTRY IS DISTANCE BETWEEN FIRST AND
% SECOND POINTS,
% THIRD ENTRY IS DISTANCE BETWEEN FIRST AND THIRD POINTS
% AS MEASURED ALONG SURFACE ETC.

load InitData
load Results

SECT = 2;    %FOR FUTURE MODIFICATIONS (WILL ALLOW MULTIPLE SECTORS)

X1 = X(1:m(2*SECT-1),1,SECT);
Y1 = Y(1:m(2*SECT-1),1,SECT);
SIZE = m(2*SECT-1);

ALBot(1) = 0;
for i = 2:SIZE    %PERFORM (ONCE - 1) FOR EACH POINT ON SURFACE
    if X1(i-1) >= r
        ALBot(i) = ALBot(i-1) + (X1(i) - X1(i-1));
    elseif X1(i) <= r
        ALBot(i) = ALBot(i-1) + r * (atan(-Y1(i) / (r - X1(i))) - ...
            atan(-Y1(i-1) / (r - X1(i-1))));
    else
        ALBot(i) = ALBot(i-1) + r * ((pi / 2) - atan(-Y1(i-1) / ...
            (r - X1(i-1)))) + (X1(i) - r);
    end
end

save Intermediate ALBot -APPEND

```

4. ArcLengthTop.m

```

function ArcLengthTop
%FUNCTION COMPUTES EXACT ARC LENGTH ALONG UPPER SURFACE WITH RADIUS r
% FIRST ENTRY IS ZERO, SECOND ENTRY IS DISTANCE BETWEEN FIRST AND
% SECOND POINTS,
% THIRD ENTRY IS DISTANCE BETWEEN FIRST AND THIRD POINTS
% AS MEASURED ALONG SURFACE ETC.

load InitData
load Results

SECT = 1;    %FOR FUTURE MODIFICATIONS (WILL ALLOW MULTIPLE SECTORS)

X1 = X(1:m(2*SECT-1),m(2*SECT),SECT);
Y1 = Y(1:m(2*SECT-1),m(2*SECT),SECT);
SIZE = m(2*SECT-1);

ALTop(1) = 0;
for i = 2:SIZE    %PERFORM (ONCE - 1) FOR EACH POINT ON SURFACE
    if X1(i-1) >= r
        ALTop(i) = ALTop(i-1) + (X1(i) - X1(i-1));
    elseif X1(i) <= r

```

```

        ALTop(i) = ALTop(i-1) + r * (atan(Y1(i) / (r - X1(i))) - ...
            atan(Y1(i-1) / (r - X1(i-1))));
    else
        %PART ON ARC, PART ON STRAIGHT SURFACE
        ALTop(i) = ALTop(i-1) + r * ((pi / 2) - atan(Y1(i-1) / ...
            (r - X1(i-1)))) + (X1(i) - r);
    end
end

save Intermediate ALTop -APPEND

```

5. CreateOutput.m

```

function CreateOutput(Motion)

load InitData
format long

if Motion == 0
    load Results
else
    load ResultsPP
end

xNew = zeros(2*m(1)-1,m(2));
yNew = zeros(2*m(1)-1,m(2));

for j = 1:m(2)
    for i = 1:m(1)
        xNew(i,j) = X(m(1)+1-i,m(2)+1-j,2);
        yNew(i,j) = Y(m(1)+1-i,m(2)+1-j,2);
        xNew(i+m(1)-1,j) = X(i,j,1);
        yNew(i+m(1)-1,j) = Y(i,j,1);
    end
end

xNew = (xNew - xB(1)) ./ CHORD;

QUAL = Metric(xNew,yNew);

if QUAL == 1
    fid=fopen('c:/Justin/NPS/Thesis/Final Version/FormattedData.txt','w');
    fprintf(fid,'%d %d \n ',[2*m(1)-1 m(2)]);
    for j = 1:m(2)
        for i = 1:2*m(1)-1
            fprintf(fid,'%13.10f %13.10f \n',xNew(i,j), yNew(i,j));
        end
    end

    fclose('all');
end

```

6. Duplicate.m

```
function Duplicate(Init)
%FUNCTION DUPLICATES UPPER SECTOR AIRFOIL WAKE AND LEFT SURFACE POINTS
%   ON LOWER SECTOR

load InitData
load Intermediate
if Init == 0
    load Results
else
    load ResultsPP
end

SECT = 2;    %FOR FUTURE MODIFICATIONS (WILL ALLOW MULTIPLE SECTORS)

%DUPPLICATE POINTS ON AIRFOIL WAKE
X(NUM2+1:m(2*SECT-1),m(2*SECT),SECT)=X(NUM+1:m(2*(SECT-1)-1),1,SECT-1);
Y(NUM2+1:m(2*SECT-1),m(2*SECT),SECT)=Y(NUM+1:m(2*(SECT-1)-1),1,SECT-1);

%DUPPLICATE POINTS ON LEFT SURFACE
X(1,1:m(2*SECT),SECT) = fliplr(X(1,1:m(2*(SECT-1)),SECT-1));
Y(1,1:m(2*SECT),SECT) = fliplr(Y(1,1:m(2*(SECT-1)),SECT-1));

if Init == 0
    save Results X Y -APPEND
else
    save ResultsPP X Y -APPEND
end
```

7. Metric.m

```
function qual = Metric(x,y)

load InitData

SECT = 1;
for i = 1:m(2*SECT-1)
    im1 = i - 1;
    ip1 = i + 1;
    for j = 1:m(2*SECT)
        if i == 1
            xxi = x(2,j,SECT) - x(1,j,SECT);
            yxi = y(2,j,SECT) - y(1,j,SECT);
        elseif i == m(2*SECT-1)
            xxi = x(m(2*SECT-1),j,SECT) - x(m(2*SECT-1)-1,j,SECT);
            yxi = y(m(2*SECT-1),j,SECT) - y(m(2*SECT-1)-1,j,SECT);
        else
            xxi = 0.5 * ( x(ip1,j,SECT) - x(im1,j,SECT) );
            yxi = 0.5 * ( y(ip1,j,SECT) - y(im1,j,SECT) );
        end

        if j == 1
            xye = 2 * x(i,2,SECT) - 1.5 * x(i,1,SECT) - ...
                0.5 * x(i,3,SECT);
```

```

        yye = 2 * y(i,2,SECT) - 1.5 * y(i,1,SECT) - ...
            0.5 * y(i,3,SECT);
elseif j == m(2*SECT)
    xye = 1.5 * x(i,m(2*SECT),SECT) - 2 * ...
        x(i,m(2*SECT)-1,SECT) + 0.5 * x(i,m(2*SECT)-2,SECT);
    yye = 1.5 * y(i,m(2*SECT),SECT) - 2 * ...
        y(i,m(2*SECT)-1,SECT) + 0.5 * y(i,m(2*SECT)-2,SECT);
else
    jml = j - 1;
    jpl = j + 1;
    xye = 0.5 * ( x(i,jpl,SECT) - x(i,jml,SECT) );
    yye = 0.5 * ( y(i,jpl,SECT) - y(i,jml,SECT) );
end

%NOTE: ALL METRIC TERMS ARE DE-SCALED BY THE JACOBIAN!
% (THAT IS, xix(i,j) SHOULD BE xix(i,j)/J(i,j)= xix(i,jT,) * aja(i,j)
% THIS IS TRUE FOR THE TIME METRICS AS WELL, SINCE THEY ARE
% MADE FROM THE OTHER DE-SCALED METRIC TERMS

    xix(i,j) = yye;
    xiy(i,j) = -xye;
    yex(i,j) = -yxi;
    yey(i,j) = xxi;

%
%      xtau = omega * (z(i,k)-z00) + plvx;
%      ztau = -omega * (x(i,k)-x00) + plvy;
%      xit(i,k) = -xtau*xix(i,k) - ztau*xiz(i,k);
%      zet(i,k) = -xtau*zex(i,k) - ztau*zez(i,k);

%COMPUTE JACOBIAN

    yacob = ( xxi * yye - xye * yxi );
    if yacob == 0
        disp('Zero Jacobian at ', i,j)
    end
    aja(i,j) = 1.0 / yacob;
end
end

%COMPUTE MAX AND MIN VALUES OF JACOBIAN AND CHECK FOR NEGATIVE VALUES

ajmax = -1.0;
ajmin = 1.0;
for j = 1:m(2*SECT)
    for i = 1:m(2*SECT)-1
        ajmax = max( ajmax,aja(i,j) );
        ajmin = min( ajmin,aja(i,j) );
    end
end

disp('ajmax = ')
disp(ajmax)
disp('ajmin = ')
disp(ajmin)

```

```

%WRITE NEGATIVE JACOBIANS
if ajmin < 0
    qual = 0;
    disp('NEGATIVE JACOBIAN')
    for j = 1:m(2*SECT)
        for i = 1:m(2*SECT-1)
            if aja(i,j) < 0
                disp(aja(i,j))
                disp(' at i = ')
                disp(i)
                disp(' j = ')
                disp(j)
            end
        end
    end
else
    qual = 1;
end

```

8. Naca.m

```

function [y] = Naca(x,SECT)
%FUNCTION COMPUTES UPPER SURFACE OF SYMMETRIC NACA FOUR DIGIT AIRFOIL
% IN CLOCKWISE DIRECTION
% INPUT ROW VECTOR x, (1 BY n VECTOR), BEGINS FROM LEFT TO RIGHT

load InitData

%define constants for NACA airfoils
a1 = 1.4779155;
a2 = -0.624424;
a3 = -1.727016;
a4 = 1.384087;
a5 = -0.510563;

if MaxCamber == 0
    if SECT == 1
        for i = 1:length(x)
            y(i) = Thickness * (a1 * sqrt(x(i)) + x(i) .* ...
                (a2 + x(i) .* (a3 + x(i) .* (a4 + x(i) * a5))));
        end
    else
        for i = 1:length(x)
            y(i) = -Thickness * (a1 * sqrt(x(i)) + x(i) .* ...
                (a2 + x(i) .* (a3 + x(i) .* (a4 + x(i) * a5))));
        end
    end
else
    if SECT == 1
        for i = 1:length(x)
            if x(i) > PosMaxCamber
                y(i) = Thickness * (a1 * sqrt(x(i)) + x(i) .* ...
                    (a2 + x(i) .* (a3 + x(i) .* ...
                        (a4 + x(i) * a5)))) + MaxCamber * ...
                    (1 + x(i) - 2 * PosMaxCamber) * ...

```

```

        (1 - x(i)) / (1 - PosMaxCamber) / (1 - PosMaxCamber);
    else
        y(i) = Thickness * (a1 * sqrt(x(i)) + x(i) .* ...
            (a2 + x(i) .* (a3 + x(i) .* ...
            (a4 + x(i) * a5)))) + MaxCamber * ...
            (2 * PosMaxCamber - x(i)) * x(i) / ...
            PosMaxCamber / PosMaxCamber;
    end
end
else
for i = 1:length(x)
    if x(i) > PosMaxCamber
        y(i) = -Thickness * (a1 * sqrt(x(i)) + x(i) .* ...
            (a2 + x(i) .* (a3 + x(i) .* ...
            (a4 + x(i) * a5)))) + MaxCamber * ...
            (1 + x(i) - 2 * PosMaxCamber) * ...
            (1 - x(i)) / (1 - PosMaxCamber) / (1 - PosMaxCamber);
    else
        y(i) = -Thickness * (a1 * sqrt(x(i)) + x(i) .* ...
            (a2 + x(i) .* (a3 + x(i) .* ...
            (a4 + x(i) * a5)))) + MaxCamber * ...
            (2 * PosMaxCamber - x(i)) * x(i) / ...
            PosMaxCamber / PosMaxCamber;
    end
end
end
end
end
end

```

9. NacaSlope.m

```

function [NormalSlope] = NacaSlope(x,SECT)
%FUNCTION COMPUTES THE NORMAL SLOPE OF A NACA FOUR DIGIT AIRFOIL IN
% CLOCKWISE DIRECTION
% INPUT ROW VECTOR x, (1 BY n VECTOR), BEGINS FROM LEFT TO RIGHT

load InitData

%define constants for NACA airfoils
a1 = 1.4779155;
a2 = -0.624424;
a3 = -1.727016;
a4 = 1.384087;
a5 = -0.510563;

if SECT == 1
    if MaxCamber == 0
        for i = 1:length(x)
            if x(i) == 0
                NormalSlope(i) = 0;
            else
                Slope = (Thickness * (0.5 * (a1 ./ sqrt(x(i))) + a2 + ...
                    x(i) .* (2 * a3 + x(i) .* (3 * a4 + x(i) .* 4 * a5))));
                if abs(Slope) < 1e-6
                    NormalSlope(i) = 1e6;
                end
            end
        end
    end
end

```

```

        else
            NormalSlope(i) = -1 ./ Slope;
        end
    end
end
else
    for i = 1:length(x)
        if x(i) > PosMaxCamber
            Slope = (Thickness * (0.5 * (a1 ./ sqrt(x(i))) + a2 + ...
                x(i) .* (2 * a3 + x(i) .* ...
                (3 * a4 + x(i) .* 4 * a5)))) + ...
                MaxCamber * 2 * ...
                (PosMaxCamber - x(i)) / (1 - PosMaxCamber) / ...
                (1 - PosMaxCamber);
            if abs(Slope) < 1e-6
                NormalSlope(i) = 1e6;
            else
                NormalSlope(i) = -1 ./ Slope;
            end
        else
            if x(i) == 0
                NormalSlope(i) = 0;
            else
                Slope = (Thickness * (0.5 * (a1 ./ sqrt(x(i))) + ...
                    a2 + x(i) .* (2 * a3 + x(i) .* ...
                    (3 * a4 + x(i) .* 4 * a5)))) + MaxCamber * 2 * ...
                    (PosMaxCamber - x(i)) / PosMaxCamber / PosMaxCamber;
                if abs(Slope) < 1e-6
                    NormalSlope(i) = 1e6;
                else
                    NormalSlope(i) = -1 ./ Slope;
                end
            end
        end
    end
end
end
else
    if MaxCamber == 0
        for i = 1:length(x)
            if x(i) == 0
                NormalSlope(i) = 0;
            else
                Slope = (Thickness * (0.5 * (a1 ./ sqrt(x(i))) + a2 + ...
                    x(i) .* (2 * a3 + x(i) .* (3 * a4 + x(i) .* 4 * a5))));
                if abs(Slope) < 1e-6
                    NormalSlope(i) = 1e6;
                else
                    NormalSlope(i) = 1 ./ Slope;
                end
            end
        end
    end
else
    for i = 1:length(x)
        if x(i) > PosMaxCamber
            Slope = -(Thickness * (0.5 * (a1 ./ sqrt(x(i))) + a2 + ...

```

```

        x(i) .* (2 * a3 + x(i) .* ...
        (3 * a4 + x(i) .* 4 * a5))) + MaxCamber * 2 * ...
        (PosMaxCamber - x(i)) / (1 - PosMaxCamber) / ...
        (1 - PosMaxCamber);
    if abs(Slope) < 1e-6
        NormalSlope(i) = 1e6;
    else
        NormalSlope(i) = -1 ./ Slope;
    end
else
    if x(i) == 0
        NormalSlope(i) = 0;
    else
        Slope = -(Thickness * (0.5 * (a1 ./ sqrt(x(i))) + ...
            a2 + x(i) .* (2 * a3 + x(i) .* (3 * a4 + x(i) .* ...
            4 * a5)))) + MaxCamber * 2 * ...
            (PosMaxCamber - x(i)) / PosMaxCamber / PosMaxCamber;
        if abs(Slope) < 1e-6
            NormalSlope(i) = 1e6;
        else
            NormalSlope(i) = -1 ./ Slope;
        end
    end
end
end
end
end
end
end
end

```

10. RTESlope.m

```

function [NormalSlope] = RTESlope(x, SECT)

load InitData

r = Thickness / 2;

if SECT == 1
    for i = 1:length(x)
        if x(i) < r
            NormalSlope(i) = sqrt(r * r - (x(i) - r) * (x(i) - r)) / ...
                (x(i) - r);
        elseif x(i) <= 1 - r
            NormalSlope(i) = 1e6;
        elseif x(i) < 1
            NormalSlope(i) = sqrt(r * r - (x(i) - 1 + r) * ...
                (x(i) - 1 + r)) / (x(i) + r - 1);
        else
            NormalSlope(i) = 0;
        end
    end
else
    for i = 1:length(x)
        if x(i) < r
            NormalSlope(i) = sqrt(r * r - (x(i) - r) * (x(i) - r)) / ...
                (r - x(i));

```

```

elseif x(i) <= 1 - r
    NormalSlope(i) = -1e6;
elseif x(i) < 1
    NormalSlope(i) = sqrt(r * r - (x(i) - 1 + r) * ...
        (x(i) - 1 + r)) / (1 - x(i) - r);
else
    NormalSlope(i) = 0;
end
end
end
end

```

11. RectArcLengthBottom.m

```

function RectArcLengthBottom
%FUNCTION COMPUTES EXACT ARC LENGTH ALONG LOWER SURFACE WITH HEIGHT r
% FIRST ENTRY IS ZERO, SECOND ENTRY IS DISTANCE BETWEEN FIRST AND
% SECOND POINTS, THIRD ENTRY IS DISTANCE BETWEEN FIRST AND THIRD
% POINTS AS MEASURED ALONG SURFACE ETC.

load InitData
load Results

SECT = 2;    %FOR FUTURE MODIFICATIONS (WILL ALLOW MULTIPLE SECTORS)

ALBot(1) = 0;
%PERFORM (ONCE - 1) FOR EACH POINT ON SURFACE
for i = 2:m(2*SECT-1)
    ALBot(i) = ALBot(i-1) + sqrt( (X(i,1,SECT) - X(i-1,1,SECT)) * ...
        (X(i,1,SECT) - X(i-1,1,SECT)) + (Y(i,1,SECT) - ...
        Y(i-1,1,SECT)) * (Y(i,1,SECT) - Y(i-1,1,SECT)) );
end

save Intermediate ALBot -APPEND

```

12. RectArcLengthTop.m

```

function RectArcLengthTop
%FUNCTION COMPUTES EXACT ARC LENGTH ALONG UPPER SURFACE WITH RADIUS r
% FIRST ENTRY IS ZERO, SECOND ENTRY IS DISTANCE BETWEEN FIRST AND
% SECOND POINTS, THIRD ENTRY IS DISTANCE BETWEEN FIRST AND THIRD
% POINTS AS MEASURED ALONG SURFACE ETC.

load InitData
load Results

SECT = 1;    %FOR FUTURE MODIFICATIONS (WILL ALLOW MULTIPLE SECTORS)

ALTop(1) = 0;
%PERFORM (ONCE - 1) FOR EACH POINT ON SURFACE
for i = 2:m(2*SECT-1)
    ALTop(i) = ALTop(i-1) + sqrt( (X(i,m(2*SECT),SECT) - ...
        X(i-1,m(2*SECT),SECT)) * ...

```

```

(X(i,m(2*SECT),SECT) - X(i-1,m(2*SECT),SECT)) + ...
(Y(i,m(2*SECT),SECT) - Y(i-1,m(2*SECT),SECT)) * ...
(Y(i,m(2*SECT),SECT) - Y(i-1,m(2*SECT),SECT)) );
end

```

```
save Intermediate ALTop -APPEND
```

13. RotateClockwise.m

```

function RotatedVector = RotateClockwise(ALPHA, X, Y, XC, YC, SIZE)
%FUNCTION ROTATES SURFACE DESCRIBED BY X AND Y CLOCKWISE BY ANGLE ALPHA

RotationMatrix = [cos(ALPHA) sin(ALPHA);-sin(ALPHA) cos(ALPHA)];

for j = 1:SIZE
    [RotatedVector(:,j)] = RotationMatrix * [(X(j) - XC);(Y(j) - YC)] +
[XC;YC];
end

```

14. Stretch.m

```

function s = Stretch(N, P, Q)
% Computes one-dimensional stretching function
% Places points along surface
% Stretching function developed by Roberts (1971)
% Modified by Eiseman (1979) - found in Fletcher Vol.2 (p.105)
% P effectively provides the slope of the distribution near eta = 0
% Q is damping factor and controls the departure from linear s vs. eta
% eta is normalized

for i = 1:N
    eta = (i - 1) / (N - 1);
    s(i) = P * eta + (1 - P) * ( 1 - (tanh(Q * (1 - eta)) / tanh(Q)));
end

```

15. UpperSurfaceNormal.m

```

function NormalSlope = UpperSurfaceNormal(X)
%FUNCTION COMPUTES NORMAL SLOPE OF UPPER SURFACE WITH A CIRCULAR
% BOUNDARY SURFACE ORIGIN IS BOTTOM, LEFT POINT OF SURFACE AND IS
% LOCATED AT (0,0)

load InitData

for i = 1:length(X)

    %ENSURE NEVER DIVIDE BY ZERO
    if X(i) < r
        NormalSlope(i) = sqrt( r * r - (X(i) - r) * (X(i) - r) ) ./ ...
            (X(i) - r) ;
    else

```

```

        NormalSlope(i) = 1e6;
    end

end

```

B. FUNCTIONS GENERATING GRIDLINES WITH USER-CONTROLLED DISTRIBUTIONS

1. AirfoilWakeInit.m

```

function AirfoilWakeInit(SeePlot)
%FUNCTION PLACES ACTUAL GRID POINTS ON UPPER AIRFOIL WAKE BOUNDARY

load PQData
load InitData

if AirfoilType == 1
    for i = 1:NumUpperSeg
        UpperAirfoil(1,i)
    end
elseif AirfoilType == 2
    for i = 1:NumUpperSeg
        UpperAirfoilRTE(1,i)
    end
end

load PQData
load InitData
load Intermediate
load Results

SECT = 1;    %FOR FUTURE MODIFICATIONS (WILL ALLOW MULTIPLE SECTORS)

if NUM > m(2*SECT-1) - 8
    error('Not enough points for Trailing Edge Line')
end

S = Stretch(m(2*SECT-1) - NUM, AWP, AWQ);
AWS = S;

%SCALE AND TRANSLATE S
% FIRST POINT IS THE SAME AS THE LAST POINT PLACED IN UPPER AIRFOIL
X(NUM+1:m(2*SECT-1),1,SECT) = S * (xB(2) - X(NUM+1,1,SECT)) + ...
    X(NUM+1,1,SECT);

%FIND RIGHT EDGE OF RIGID SURFACE AND PLACE POINTS WITHIN RIGID SURFACE
i = 2;
while X(NUM+i,1,SECT) < X(NUM+1,1,SECT) + SRigid * CHORD
    Y(NUM+i,1,SECT) = yB(1);
    i = i + 1;
end

```

```

%NEED AT LEAST FOUR POINTS IN RIGID SURFACE
if i < 5
    i = 5;
end

EDGER = NUM+i-1;

Y(EDGER+1:m(2*SECT-1)-1,1,SECT) = yB(1);

if SeePlot == 0
    figure
    plot(X(NUM+1:m(2*SECT-1),1,SECT),Y(NUM+1:m(2*SECT-1),1,SECT), ...
        X(NUM+1:m(2*SECT-1),1,SECT),Y(NUM+1:m(2*SECT-1),1,SECT),'o')
    axis equal
end

save Results X Y -APPEND
save Intermediate EDGER AWS -APPEND

```

2. AirfoilWake.m

```

function AirfoilWake
%FUNCTION PLACES ACTUAL GRID POINTS ON UPPER AIRFOIL WAKE BOUNDARY

load InitData
load Intermediate
load ResultsPP

SECT = 1;    %FOR FUTURE MODIFICATIONS (WILL ALLOW MULTIPLE SECTORS)

%*****
%ASSUMES RIGHT SURFACE IS A STRAIGHT VERTICAL LINE
X(m(2*SECT-1),1,SECT) = xB(2);

%ACCOUNT FOR PLUNGE AND AOA
%MOVE INTERSECTION OF WAKE AT RIGHT BOUNDARY
Y(m(2*SECT-1),1,SECT) = RBPlunge * plunge - tan(RBPitch * aoa) * ...
    sqrt( (xB(2) - X(NUM+1,1,1)) * (xB(2) - X(NUM+1,1,1)) + ...
    (Y(NUM+1,1,1) * Y(NUM+1,1,1)) );

if abs(Y(m(2*SECT-1),1,SECT)) >= r
    error('Too much motion on right boundary - reduce pitch and plunge
        constants')
end

%DETERMINE THE NEW X COORDINATE DISTRIBUTION AFTER ROTATION AND PLUNGE
X(EDGER+1:m(2*SECT-1)-1,1,SECT)=(AWS(EDGER-NUM+1:m(2*SECT-1)-NUM-1)-...
    AWS(EDGER-NUM)) / (AWS(m(2*SECT-1)-NUM) - AWS(EDGER-NUM)) * ...
    (X(m(2*SECT-1),1,SECT) - X(EDGER,1,SECT)) + X(EDGER,1,SECT);

%ENFORCE NORMAL ON RIGHT BOUNDARY - PLACE ONE POINT NORMAL
Y(m(2*SECT-1)-1,1,SECT) = Y(m(2*SECT-1),1,SECT);

```

```
%USE HYPERBOLIC TANGENT DISTRIBUTION TO SMOOTH DELTA IN HEIGHT BTWN
%EDGER AND RT BOUNDARY ALSO USE HYPERBOLIC DISTRIBUTION TO SMOOTH DELTA
%IN SLOPES AT EDGER AND RIGHT BOUNDARY
```

```
%HYPERBOLIC TANGENT SMOOTHING OF SLOPE AT RIGHT EDGE OF RIGID SURFACE
S = AWS(EDGER-NUM:m(2*SECT-1)-NUM-1);
L = length(S);
S = (S - S(1)) / (S(L) - S(1));
i = 1;
lim = 0.5;
while S(3) > lim
    lim = lim + 0.01;
end
while S(i) <= lim
    p(i) = S(i);
    i = i + 1;
end
p = p / (S(i-1) - S(1)) * 4 - 2;
q = -abs(tanh(p));
p = ((p + 2) / 4) * (S(i-1) - S(1));
q = q - q(1);
q = q ./ sum(q);
p(i:L) = S(i:L);
q(i:L) = 0;
SlopeL(1) = -tan(boa); %SLOPE AT EDGER OF RIGID SURFACE

SlopeDelta = -SlopeL(1);
for j = 2:L-1
    %area KEEPS TRACK OF AREA UNDER TANH CURVE
    area(j-1) = (q(j-1) + 0.5 * (q(j) - q(j-1))) * ...
        (p(j) - p(j-1));
    SlopeL(j) = (SlopeDelta) * sum(area(1:j-1));
end

SlopeL(2:L-1) = SlopeL(1) + SlopeL(2:L-1) / sum(area);

delta1(1,1) = 0;
for i = 2:L
    delta1(i,1) = delta1(i-1,1) + SlopeL(i-1) * (X(EDGER+i-1,1,SECT) - ...
        X(EDGER+i-2,1,SECT));
end

%DETERMINE THE Y COORDINATE DISTRIBUTION TO TAPER FROM EDGER TO THE
% RIGHT BOUNDARY
p = S * 4 - 2;
q = -abs(tanh(p));
q = q - q(1);
q = q ./ sum(q);

TotDelta = Y(EDGER,1,SECT) - Y(m(2*SECT-1)-1,1,SECT) + delta1(L);

for i = 2:L-1
    %area KEEPS TRACK OF AREA UNDER TANH CURVE
    area(i-1) = (q(i-1) + 0.5 * (q(i) - q(i-1))) * ...
        (p(i) - p(i-1));
```

```

    Y(i+EDGER-1,1,SECT) = (TotDelta) * sum(area(1:i-1));
end

Y(EDGER+1:m(2*SECT)-2,1,SECT) = Y(EDGER,1,SECT) + delta1(2:L-1) - ...
    Y(EDGER+1:m(2*SECT)-1)-2,1,SECT) / sum(area);

save ResultsPP X Y -APPEND

```

3. LeftSurfaceInit.m

```

function LeftSurfaceInit(SeePlot)
%FUNCTION PLACES ACTUAL GRID POINTS ON UPPER LEFT BOUNDARY

AirfoilWakeInit(1)

load InitData
load PQData
load Intermediate
load Results

SECT = 1;    %FOR FUTURE MODIFICATIONS (WILL ALLOW MULTIPLE SECTORS)

%FIRST POINT IS ALREADY SPECIFIED BY T1 - DISTRIBUTE 1 LESS AS A RESULT
S = stretch(m(2*SECT)-1, LSP, LSQ);
S = fliplr(abs(fliplr(S) - 1));
LSS = S;

%SCALE AND TRANSLATE S
X(1,2:m(2*SECT),SECT) = S .* (xB(1) - thick1);

%PLACE POINTS WITHIN RIGID SURFACE
i = 2;
while X(1,i,SECT) > (X(1,1,SECT) - SRigid * CHORD) & i < m(2*SECT)
    Y(1,i,SECT) = Y(1,1,SECT);
    i = i + 1;
end

if i == m(2*SECT)
    error('Size of Rigid Surface too large')
end

%NEED EDGE OF RIGID SURFACE FOR LATER
EDGEL = i-1;

%NEED AT LEAST FOUR POINTS IN RIGID SURFACE
if EDGEL < 5
    EDGEL = 5;
end

%DISTRIBUTE Y-COORDINATES
Y(1,EDGEL+1:m(2*SECT),SECT) = Y(1,1,SECT);

if SeePlot == 0
    figure

```

```

    plot(X(1,1:m(2*SECT),SECT),Y(1,1:m(2*SECT),SECT), ...
         X(1,1:m(2*SECT),SECT),Y(1,1:m(2*SECT),SECT),'o')
    axis equal
end

save Results X Y -APPEND

save Intermediate EDGEL LSS -APPEND

```

4. LeftSurface.m

```

function LeftSurface
%FUNCTION PLACES ACTUAL GRID POINTS ON UPPER LEFT BOUNDARY

load InitData
load Intermediate
load ResultsPP

SECT = 1; %FOR FUTURE MODIFICATIONS (WILL ALLOW MULTIPLE SECTORS)

%*****
%ASSUMES LEFT SIDE OF UPPER SURFACE IS A CIRCLE WITH RADIUS = R
%FIND X COORDINATE OF INTERSECTION WITH UPPER BOUNDARY
Y(1,m(2*SECT),SECT) = LBPlunge * plunge + tan(LBPitch * aoa) * ...
    sqrt( (X(1,1,1) * X(1,1,1)) + (Y(1,1,1) * Y(1,1,1)) );
%X = r - sqrt(r*r - y*y)
X(1,m(2*SECT),SECT) = r - sqrt( r * r - (Y(1,m(2*SECT),SECT) * ...
    Y(1,m(2*SECT),SECT)) );

%DETERMINE UPPER BOUNDARY NORMAL SLOPE AT INTERSECTION
UpperBndryNormal = UpperSurfaceNormal(X(1,m(2*SECT),SECT));
if Y(1,m(2*SECT),SECT) < 0
    UpperBndryNormal = -UpperBndryNormal;
end

%DETERMINE THE NEW X COORDINATE DISTRIBUTION AFTER ROTATION AND PLUNGE
%   SPREAD THE INCREASED ARC LENGTH OUT WITH A HYPERBOLIC
%   TANGENT DISTRIBUTION
X(1,EDGEL+1:m(2*SECT)-1,SECT) = (LSS(EDGEL:m(2*SECT)-2) - ...
    LSS(m(2*SECT)-1)) / (LSS(EDGEL-1) - LSS(m(2*SECT)-1)) * ...
    (X(1,EDGEL,SECT) - X(1,m(2*SECT),SECT)) + X(1,m(2*SECT),SECT);

%CREATE NEW S VECTOR
S = (X(1,EDGEL:m(2*SECT)-1,SECT) - X(1,m(2*SECT)-1,SECT)) / ...
    (X(1,EDGEL,SECT) - X(1,m(2*SECT)-1,SECT));

%ENFORCE NORMAL ON UPPER BOUNDARY
%   PLACE ONE POINT NORMAL
Y(1,m(2*SECT)-1,SECT) = UpperBndryNormal * (X(1,m(2*SECT)-1,SECT) - ...
    X(1,m(2*SECT),SECT)) + Y(1,m(2*SECT),SECT);

%DETERMINE HOW THE SHAPE OF THE LEFT BOUNDARY LINE WILL VARY
%   A HYPERBOLIC TANGENT DISTRIBUTION IS USED CURRENTLY TO TAPER THE
%   Y-COORDINATE DELTA

```

```

% MATCH SLOPES AT BOTH SIDES ALSO
% TAPER THE SLOPES WITH A HYPERBOLIC TANGENT DISTRIBUTION
L = length(S);

%HYPERBOLIC TANGENT SMOOTHING OF SLOPE OF RIGHT SIDE
if length(S) > 3
    i = 1;
    lim = 0.5;
    while S(3) < 1 - lim
        lim = lim + 0.01;
    end
    while S(i) >= 1 - lim
        p(i) = S(i);
        i = i + 1;
    end
    p = ((p - S(i-1)) / (S(1) - S(i-1))) * 4 - 2;
    q = -abs(tanh(p));
    p = ((p + 2) / 4) * (S(1) - S(i-1)) + S(i-1);
    q = q - q(1);
    q = q ./ sum(q);
    p(i:L) = S(i:L);
    q(i:L) = 0;

    SlopeR(1) = tan(-aoa); %SLOPE AT INTERSECTION WITH RIGID SURFACE

    SlopeDelta = -SlopeR(1);
    for j = 2:L-1
        %area KEEPS TRACK OF AREA UNDER TANH CURVE
        area(j-1) = (q(j-1) + 0.5 * (q(j) - q(j-1))) * ...
            (p(j) - p(j-1));
        SlopeR(j) = (SlopeDelta) * sum(area(1:j-1));
    end

    SlopeR(2:L-1) = SlopeR(1) + SlopeR(2:L-1) / sum(area);

    delta1(1) = 0;
    for i = 2:L
        delta1(i) = delta1(i-1) + SlopeR(i-1) * (X(1,EDGEL+i-1,SECT) - ...
            X(1,EDGEL+i-2,SECT));
    end

%HYPERBOLIC TANGENT SMOOTHING OF SLOPE OF LEFT SIDE
i = L;
lim = 0.5;
while S(L-2) > lim
    lim = lim + 0.01;
end
while S(i) <= lim
    p(i) = S(i);
    i = i - 1;
end
p(i+1:L) = (p(i+1:L) / S(i+1)) * 4 - 2;
q(i+1:L) = -abs(tanh(p(i+1:L)));
q(i+1:L) = q(i+1:L) - q(i+1);
q(1:i) = 0;

```

```

q = q ./ sum(q(1:L));
p(i+1:L) = ((p(i+1:L) + 2) / 4) * S(i+1);
p(1:i) = S(1:i);

SlopeL(1) = 0;
SlopeL(L) = UpperBndryNormal;

SlopeDelta = SlopeL(L);
for j = 2:L-1
    %area KEEPS TRACK OF AREA UNDER TANH CURVE
    area(j-1) = (q(j-1) + 0.5 * (q(j) - q(j-1))) * (p(j) - p(j-1));
    SlopeL(j) = (SlopeDelta) * sum(area(1:j-1));
end

SlopeL(2:L-1) = SlopeL(1) + SlopeL(2:L-1) / sum(area);

delta2(1) = 0;
for i = 2:L
    delta2(i) = delta2(i-1) + SlopeL(i-1) * (X(1,EDGEL+i-1,SECT)- ...
        X(1,EDGEL+i-2,SECT));
end
else
    delta1(1:L) = 0;
    delta2(1:L) = 0;
    disp('Rigid Surface May Be Too Large')
end

%DETERMINE THE Y COORDINATE DISTRIBUTION TO TAPER FROM EDGEmR TO EDGER
% OF THE RIGID SURFACE ALONG THE TOP OF THE RIGID SURFACE
p = S * 4 - 2;
q = -abs(tanh(p));
q = q - q(1);
q = q ./ sum(q);

TotDelta = Y(1,EDGEL,SECT) - Y(1,m(2*SECT)-1,SECT) + delta1(L) + ...
    delta2(L);

for i = 2:L-1
    %area KEEPS TRACK OF AREA UNDER TANH CURVE
    area(i-1) = (q(i-1) + 0.5 * (q(i) - q(i-1))) * (p(i) - p(i-1));
    Y(1,EDGEL+i-1,SECT) = (TotDelta) * sum(area(1:i-1));
end

Y(1,EDGEL+1:m(2*SECT)-2,SECT) = Y(1,EDGEL,SECT) + delta1(2:L-1) + ...
    delta2(2:L-1) - Y(1,EDGEL+1:m(2*SECT)-2,SECT) / sum(area);

save ResultsPP X Y -APPEND

```

5. RectLeftSurface.m

```

function RectLeftSurface
%FUNCTION PLACES ACTUAL GRID POINTS ON UPPER LEFT BOUNDARY

load InitData

```

```

load Intermediate
load ResultsPP

SECT = 1;    %FOR FUTURE MODIFICATIONS (WILL ALLOW MULTIPLE SECTORS)

%*****
%ASSUMES LEFT SIDE OF UPPER SURFACE IS A VERTICAL LINE
%FIND X COORDINATE OF INTERSECTION WITH UPPER BOUNDARY
Y(1,m(2*SECT),SECT) = LBPlunge * plunge + tan(LBPitch * aoa) * ...
    sqrt( (X(1,1,1) * X(1,1,1)) + (Y(1,1,1) * Y(1,1,1)) );
X(1,m(2*SECT),SECT) = 0;

if abs(Y(1,m(2*SECT),SECT)) > r
    error('Too much motion along upper boundary - reduce Pitch and
        Plunge constants')
end

%DETERMINE THE NEW X COORDINATE DISTRIBUTION AFTER ROTATION AND PLUNGE
%    SPREAD THE INCREASED ARC LENGTH OUT WITH A HYPERBOLIC TANGENT
DISTRIBUTION
X(1,EDGEL+1:m(2*SECT)-1,SECT) = (LSS(EDGEL:m(2*SECT)-2) - ...
    LSS(m(2*SECT)-1)) / (LSS(EDGEL-1) - LSS(m(2*SECT)-1)) * ...
    (X(1,EDGEL,SECT) - X(1,m(2*SECT),SECT)) + X(1,m(2*SECT),SECT);

%CREATE NEW S VECTOR
S = (X(1,EDGEL:m(2*SECT)-1,SECT) - X(1,m(2*SECT)-1,SECT)) / ...
    (X(1,EDGEL,SECT) - X(1,m(2*SECT)-1,SECT));

%ENFORCE NORMAL ON UPPER BOUNDARY
%    PLACE ONE POINT NORMAL
Y(1,m(2*SECT)-1,SECT) = Y(1,m(2*SECT),SECT);

%DETERMINE HOW THE SHAPE OF THE LEFT BOUNDARY LINE WILL VARY
%    A HYPERBOLIC TANGENT DISTRIBUTION IS USED CURRENTLY TO TAPER THE Y-
%    COORDINATE DELTA MATCH SLOPES AT BOTH SIDES ALSO
%    TAPER THE SLOPES WITH A HYPERBOLIC TANGENT DISTRIBUTION
L = length(S);

%HYPERBOLIC TANGENT SMOOTHING OF SLOPE OF RIGHT SIDE
i = 1;
lim = 0.5;
while S(3) < 1 - lim
    lim = lim + 0.01;
end
while S(i) >= 1 - lim
    p(i) = S(i);
    i = i + 1;
end
p = ((p - S(i-1)) / (S(1) - S(i-1))) * 4 - 2;
q = -abs(tanh(p));
p = ((p + 2) / 4) * (S(1) - S(i-1)) + S(i-1);
q = q - q(1);
q = q ./ sum(q);
p(i:L) = S(i:L);
q(i:L) = 0;

```

```

SlopeR(1) = tan(-aoa);          %SLOPE AT INTERSECTION WITH RIGID SURFACE

SlopeDelta = -SlopeR(1);
for j = 2:L-1
    %area KEEPS TRACK OF AREA UNDER TANH CURVE
    area(j-1) = (q(j-1) + 0.5 * (q(j) - q(j-1))) * ...
        (p(j) - p(j-1));
    SlopeR(j) = (SlopeDelta) * sum(area(1:j-1));
end

SlopeR(2:L-1) = SlopeR(1) + SlopeR(2:L-1) / sum(area);

delta1(1) = 0;
for i = 2:L
    delta1(i) = delta1(i-1) + SlopeR(i-1) * (X(1,EDGEL+i-1,SECT) - ...
        X(1,EDGEL+i-2,SECT));
end

%DETERMINE THE Y COORDINATE DISTRIBUTION TO TAPER FROM EDGEmR TO EDGER
%   OF THE RIGID SURFACE ALONG THE TOP OF THE RIGID SURFACE
p = S * 4 - 2;
q = -abs(tanh(p));
q = q - q(1);
q = q ./ sum(q);

TotDelta = Y(1,EDGEL,SECT) - Y(1,m(2*SECT)-1,SECT) + delta1(L);

for i = 2:L-1
    %area KEEPS TRACK OF AREA UNDER TANH CURVE
    area(i-1) = (q(i-1) + 0.5 * (q(i) - q(i-1))) * (p(i) - p(i-1));
    Y(1,EDGEL+i-1,SECT) = (TotDelta) * sum(area(1:i-1));
end

Y(1,EDGEL+1:m(2*SECT)-2,SECT) = Y(1,EDGEL,SECT) + delta1(2:L-1) + ...
    - Y(1,EDGEL+1:m(2*SECT)-2,SECT) / sum(area);

save ResultsPP X Y -APPEND

```

6. LowerAirfoil.m

```

function LowerAirfoil(SeePlot, Seg)
%FUNCTION PLACES ACTUAL GRID POINTS ON LOWER AIRFOIL BOUNDARY

load InitData
load PQData
load Intermediate
load Results

SECT = 2;    %FOR FUTURE MODIFICATIONS (WILL ALLOW MULTIPLE SECTORS)

NUM2 = 0;
for i = 1:Seg
    num2(Seg) = LANUM(Seg);
    NUM2 = num2(i) + NUM2;

```

```

end

%PLACE EXTRA POINT SO NO DUPLICATE POINTS
S = Stretch(num2(Seg)+1, LAP(Seg), LAQ(Seg));

if LDir(Seg) == 2
    S = abs(fliplr(S) - 1);
end

if Seg == 1
    x = [0:0.0001:EdgeLowerSeg(1)];
    X(1,m(2*SECT),SECT) = xB(1);
    Y(1,m(2*SECT),SECT) = yB(1);
else
    x = [EdgeLowerSeg(Seg-1):0.0001:EdgeLowerSeg(Seg)];
end

y = Naca(x,SECT);

x = x * CHORD + xB(1);
y = y * CHORD + yB(1);

s(1) = 0;
for i = 2:length(x)
    s(i) = s(i-1) + sqrt((x(i) - x(i-1)) * (x(i) - x(i-1)) + ...
        (y(i) - y(i-1)) * (y(i) - y(i-1))));
end
sTot = s(length(x));
s = s / sTot;

tempS = 0;
delta = S(2);
sdelta = s(2);
i = 2;
j = 2;
while i < num2(Seg) + 1
    while delta > sdelta & j < length(x)
        delta = delta - sdelta;
        j = j + 1;
        sdelta = s(j) - s(j-1);
    end
    angle = atan((y(j) - y(j-1)) / (x(j) - x(j-1)));
    while delta <= sdelta
        tempS = tempS + delta;
        X(i+NUM2-num2(Seg),m(2*SECT),SECT) = cos(angle) * tempS * ...
            sTot + x(j-1);
        Y(i+NUM2-num2(Seg),m(2*SECT),SECT) = sin(angle) * tempS * ...
            sTot + y(j-1);
        i = i + 1;
        sdelta = sdelta - delta;
        delta = S(i) - S(i-1);
    end
    tempS = 0;
end
end

```

```

Y(i+NUM2-num2(Seg),m(2*SECT),SECT)=Naca(X(i+NUM-num(Seg),1,SECT),SECT);
if Seg ~= NumLowerSeg
    X(NUM2+1,m(2*SECT),SECT) = EdgeLowerSeg(Seg) * CHORD + xB(1);
    Y(NUM2+1,m(2*SECT),SECT) = Naca(EdgeLowerSeg(Seg),2) * CHORD +yB(1);
else
    X(NUM2+1,m(2*SECT),SECT) = xB(1) + CHORD;
    Y(NUM2+1,m(2*SECT),SECT) = yB(1);
end

if SeePlot == 0
    figure
    plot(x,y, X(NUM2-num2(Seg)+1:NUM2+1,m(2*SECT),SECT), ...
        Y(NUM2-num2(Seg)+1:NUM2+1,m(2*SECT),SECT),'o')
    axis equal
end

save Results X Y -APPEND
save Intermediate NUM2 num2 -APPEND

```

7. LowerAirfoilRTE.m

```

function LowerAirfoilRTE(SeePlot, Seg)
%FUNCTION PLACES ACTUAL GRID POINTS ON LOWER AIRFOIL BOUNDARY

load InitData
load PQData
load Intermediate
load Results

SECT = 2; %FOR FUTURE MODIFICATIONS (WILL ALLOW MULTIPLE SECTORS)

r = CHORD * Thickness / 2;

NUM2 = 0;
for i = 1:Seg
    num2(Seg) = LANUM(Seg);
    NUM2 = num2(i) + NUM2;
end

%PLACE EXTRA POINT SO NO DUPLICATE POINTS
S = Stretch(num2(Seg)+1, LAP(Seg), LAQ(Seg));

if LDir(Seg) == 2
    S = abs(fliplr(S) - 1);
end

if Seg == 1
    %ON LE
    if EdgeLowerSeg(1) < Thickness
        S = S * (r * acos((r - CHORD * EdgeLowerSeg(1)) / r));
    %ON FLAT SURFACE
    elseif EdgeLowerSeg(1) < 1 - Thickness
        S = S * (pi * r / 2 + EdgeLowerSeg(1) * CHORD - r);
    %ON TE

```

```

elseif EdgeLowerSeg(1) < 1
    S = S * (pi * r / 2 + CHORD - 2 * r + r * ...
        asin((r + CHORD * EdgeLowerSeg(1) - CHORD) / r));
%ENTIRE AIRFOIL
else
    S = S * (CHORD - 2 * r + pi * r);
end
else
    %CURRENT AND PREVIOUS SEGMENT ON LE
    if EdgeLowerSeg(Seg) < Thickness
        S = S * ( r * (acos((r - CHORD * EdgeLowerSeg(Seg)) / r) - ...
            acos((r - CHORD * EdgeLowerSeg(Seg-1)) / r)) + r * ...
            acos((r - CHORD * EdgeLowerSeg(Seg-1)) / r);
        %CURRENT SEGMENT ON FLAT SURFACE
    elseif EdgeLowerSeg(Seg) < 1 - Thickness
        %PREVIOUS SEGMENT ON LE
        if EdgeLowerSeg(Seg-1) < Thickness
            S = S * (CHORD * EdgeLowerSeg(Seg) - r + pi * r / ...
                2 - r * acos((r - CHORD * EdgeLowerSeg(Seg-1)) / r)) + ...
                r * acos((r - CHORD * EdgeLowerSeg(Seg-1)) / r);
            %PREVIOUS SEGMENT ON FLAT SURFACE
        else
            S = S * CHORD * (EdgeLowerSeg(Seg) - ...
                EdgeLowerSeg(Seg-1)) + pi * r / 2 + CHORD * ...
                EdgeLowerSeg(Seg-1) - r;
        end
        %CURRENT SEGMENT ON TE
    elseif EdgeLowerSeg(Seg) < 1
        %PREVIOUS SEGMENT ON LE
        if EdgeLowerSeg(Seg-1) < Thickness
            S = S * (CHORD - 2 * r + pi * r / 2 - r * acos((r - ...
                CHORD * EdgeLowerSeg(Seg-1)) / r) + r * asin((r + ...
                CHORD * EdgeLowerSeg(Seg) - CHORD) / r)) + r * acos((r - ...
                CHORD * EdgeLowerSeg(Seg-1)) / r);
            %PREVIOUS SEGMENT ON FLAT SURFACE
        elseif EdgeLowerSeg(Seg-1) < 1 - Thickness
            S = S * (CHORD - r - CHORD * EdgeLowerSeg(Seg-1) + r * ...
                asin((r + CHORD * EdgeLowerSeg(Seg) - CHORD) / r)) + ...
                pi * r / 2 + CHORD * EdgeLowerSeg(Seg-1) - r;
            %PREVIOUS SEGMENT ON TE
        else
            S = S * (r * (asin((r + CHORD * EdgeLowerSeg(Seg) - ...
                CHORD) / r) - asin((r + CHORD * EdgeLowerSeg(Seg-1) - ...
                CHORD) / r))) + pi * r / 2 + CHORD - 2 * r + r * ...
                asin((r + CHORD * EdgeLowerSeg(Seg-1) - CHORD) / r);
        end
        %ENTIRE AIRFOIL
    else
        %PREVIOUS SEGMENT ON LE
        if EdgeLowerSeg(Seg-1) < Thickness
            S = S * (CHORD - 2 * r + pi * r - r * acos((r - CHORD * ...
                EdgeLowerSeg(Seg-1)) / r)) + r * acos((r - CHORD * ...
                EdgeLowerSeg(Seg-1)) / r);
            %PREVIOUS SEGMENT ON FLAT SURFACE
        elseif EdgeLowerSeg(Seg-1) < 1 - Thickness

```

```

        S = S * (CHORD - CHORD * EdgeLowerSeg(Seg-1) - r + pi * ...
            r / 2) + pi * r / 2 + CHORD * EdgeLowerSeg(Seg-1) - r;
%PREVIOUS SEGMENT ON TE
    else
        S = S * (pi * r / 2 - r * asin((r + CHORD * ...
            EdgeLowerSeg(Seg-1) - CHORD) / r)) + pi * r / 2 + ...
            CHORD - 2 * r + r * asin((r + CHORD * ...
            EdgeLowerSeg(Seg-1) - CHORD) / r);
    end
end
end

j = 1;

while j < num2(Seg) + 2

    %ON LE
    if S(j) < pi * r / 2
        X(j + NUM2 - num2(Seg),m(2*SECT),SECT) = r - r * ...
            cos(S(j) / r) + xB(1);
        Y(j + NUM2 - num2(Seg),m(2*SECT),SECT) = -r * sin(S(j) / ...
            r) + yB(1);
    %ON FLAT SURFACE
    elseif S(j) <= CHORD + pi * r / 2 - 2 * r
        X(j + NUM2 - num2(Seg),m(2*SECT),SECT) = S(j) - ...
            (pi * r / 2) + r + xB(1);
        Y(j + NUM2 - num2(Seg),m(2*SECT),SECT) = -r + yB(1);
    %ON TE
    else
        X(j + NUM2 - num2(Seg),m(2*SECT),SECT) = r * sin((S(j) - ...
            pi * r / 2 - CHORD + 2 * r) / r) + CHORD - r + xB(1);
        Y(j + NUM2 - num2(Seg),m(2*SECT),SECT) = -r * cos((S(j) - ...
            pi * r / 2 - CHORD + 2 * r) / r) + yB(1);
    end

    j = j + 1;

end

if SeePlot == 0
    figure
    plot(X(NUM2-num2(Seg)+1:NUM2+1,m(2*SECT),SECT), ...
        Y(NUM2-num2(Seg)+1:NUM2+1,m(2*SECT),SECT),'o')
    axis equal
end

save Results X Y -APPEND
save Intermediate NUM2 num2 -APPEND

```

8. UpperAirfoil.m

```

function UpperAirfoil(SeePlot, Seg)
%FUNCTION PLACES ACTUAL GRID POINTS ON UPPER AIRFOIL BOUNDARY

load InitData

```

```

load PQData
load Intermediate
load Results

SECT = 1;    %FOR FUTURE MODIFICATIONS (WILL ALLOW MULTIPLE SECTORS)

NUM = 0;
for i = 1:Seg
    num(Seg) = UANUM(Seg);
    NUM = num(i) + NUM;
end

S = Stretch(num(Seg)+1, UAP(Seg), UAQ(Seg)); %PLACE EXTRA POINT SO NO
DUPLICATE POINTS

if Dir(Seg) == 2
    S = abs(fliplr(S) - 1);
end

if Seg == 1
    x = [0:0.0001:EdgeUpperSeg(1)];
    X(1,1,SECT) = xB(1);
    Y(1,1,SECT) = yB(1);
else
    x = [EdgeUpperSeg(Seg-1):0.0001:EdgeUpperSeg(Seg)];
end

y = Naca(x,SECT);

x = x * CHORD + xB(1);
y = y * CHORD + yB(1);

s(1) = 0;
for i = 2:length(x)
    s(i) = s(i-1) + sqrt( (x(i) - x(i-1)) * (x(i) - x(i-1)) + ...
        (y(i) - y(i-1)) * (y(i) - y(i-1)) );
end
sTot = s(length(x));
s = s / sTot;

tempS = 0;
delta = S(2);
sdelta = s(2);
i = 2;
j = 2;
while i < num(Seg) + 1
    while delta > sdelta & j < length(x)
        delta = delta - sdelta;
        j = j + 1;
        sdelta = s(j) - s(j-1);
    end
    angle = atan((y(j) - y(j-1)) / (x(j) - x(j-1)));
    while delta <= sdelta & i < num(Seg) + 1
        tempS = tempS + delta;
        X(i+NUM-num(Seg),1,SECT) = cos(angle) * tempS * sTot + x(j-1);

```

```

        Y(i+NUM-num(Seg),1,SECT) = sin(angle) * tempS * sTot + y(j-1);
        i = i + 1;
        sdelta = sdelta - delta;
        delta = S(i) - S(i-1);
    end
    tempS = 0;
end

Y(i+NUM-num(Seg),1,SECT) = Naca(X(i+NUM-num(Seg),1,SECT),SECT);

if Seg ~= NumUpperSeg
    X(NUM+1,1,SECT) = EdgeUpperSeg(Seg) * CHORD + xB(1);
    Y(NUM+1,1,SECT) = Naca(EdgeUpperSeg(Seg),1) * CHORD + yB(1);
else
    X(NUM+1,1,SECT) = xB(1) + CHORD;
    Y(NUM+1,1,SECT) = yB(1);
end

if SeePlot == 0
    figure
    plot(x,y, X(NUM-num(Seg)+1:NUM+1,1,SECT), ...
        Y(NUM-num(Seg)+1:NUM+1,1,SECT), 'o')
    axis equal
end

save Results X Y -APPEND
save Intermediate NUM num -APPEND

```

9. UpperAirfoilRTE.m

```

function UpperAirfoilRTE(SeePlot, Seg)
%FUNCTION PLACES ACTUAL GRID POINTS ON UPPER AIRFOIL BOUNDARY

load InitData
load PQData
load Intermediate
load Results

r = CHORD * Thickness / 2;

SECT = 1;    %FOR FUTURE MODIFICATIONS (WILL ALLOW MULTIPLE SECTORS)

NUM = 0;
for i = 1:Seg
    num(Seg) = UANUM(Seg);
    NUM = num(i) + NUM;
end

%PLACE EXTRA POINT SO NO DUPLICATE POINTS
S = Stretch(num(Seg)+1, UAP(Seg), UAQ(Seg));

if Dir(Seg) == 2
    S = abs(fliplr(S) - 1);
end

```

```

if Seg == 1
    %ON LE
    if EdgeUpperSeg(1) < Thickness
        S = S * (r * acos((r - CHORD * EdgeUpperSeg(1)) / r));
    %ON FLAT SURFACE
    elseif EdgeUpperSeg(1) < 1 - Thickness
        S = S * (pi * r / 2 + EdgeUpperSeg(1) * CHORD - r);
    %ON TE
    elseif EdgeUpperSeg(1) < 1
        S = S * (pi * r / 2 + CHORD - 2 * r + r * asin((r + ...
            CHORD * EdgeUpperSeg(1) - CHORD) / r));
    %ENTIRE AIRFOIL
    else
        S = S * (CHORD - 2 * r + pi * r);
    end
else
    %CURRENT AND PREVIOUS SEGMENT ON LE
    if EdgeUpperSeg(Seg) < Thickness
        S = S * ( r * (acos((r - CHORD * EdgeUpperSeg(Seg)) / ...
            r) - acos((r - CHORD * EdgeUpperSeg(Seg-1)) / r)) + ...
            r * acos((r - CHORD * EdgeUpperSeg(Seg-1)) / r);
    %CURRENT SEGMENT ON FLAT SURFACE
    elseif EdgeUpperSeg(Seg) < 1 - Thickness
        %PREVIOUS SEGMENT ON LE
        if EdgeUpperSeg(Seg-1) < Thickness
            S = S * (CHORD * EdgeUpperSeg(Seg) - r + pi * r / ...
                2 - r * acos((r - CHORD * EdgeUpperSeg(Seg-1)) / r)) + ...
                r * acos((r - CHORD * EdgeUpperSeg(Seg-1)) / r);
        %PREVIOUS SEGMENT ON FLAT SURFACE
        else
            S = S * CHORD * (EdgeUpperSeg(Seg) - ...
                EdgeUpperSeg(Seg-1)) + ...
                pi * r / 2 + CHORD * EdgeUpperSeg(Seg-1) - r;
        end
    %CURRENT SEGMENT ON TE
    elseif EdgeUpperSeg(Seg) < 1
        %PREVIOUS SEGMENT ON LE
        if EdgeUpperSeg(Seg-1) < Thickness
            S = S * (CHORD - 2 * r + pi * r / 2 - r * acos((r - ...
                CHORD * EdgeUpperSeg(Seg-1)) / r) + r * asin((r + ...
                CHORD * EdgeUpperSeg(Seg) - CHORD) / r)) + r * acos((r - ...
                CHORD * EdgeUpperSeg(Seg-1)) / r);
        %PREVIOUS SEGMENT ON FLAT SURFACE
        elseif EdgeUpperSeg(Seg-1) < 1 - Thickness
            S = S * (CHORD - r - CHORD * EdgeUpperSeg(Seg-1) + r * ...
                asin((r + CHORD * EdgeUpperSeg(Seg) - CHORD) / r)) + ...
                pi * r / 2 + CHORD * EdgeUpperSeg(Seg-1) - r;
        %PREVIOUS SEGMENT ON TE
        else
            S = S * (r * (asin((r + CHORD * EdgeUpperSeg(Seg) - ...
                CHORD) / r) - asin((r + CHORD * EdgeUpperSeg(Seg-1) - ...
                CHORD) / r))) + pi * r / 2 + CHORD - 2 * r + r * ...
                asin((r + CHORD * EdgeUpperSeg(Seg-1) - CHORD) / r);
        end
    end
end

```

```

%ENTIRE AIRFOIL
else
    %PREVIOUS SEGMENT ON LE
    if EdgeUpperSeg(Seg-1) < Thickness
        S = S * (CHORD - 2 * r + pi * r - r * acos((r - CHORD * ...
            EdgeUpperSeg(Seg-1)) / r)) + r * acos((r - CHORD * ...
            EdgeUpperSeg(Seg-1)) / r);
        %PREVIOUS SEGMENT ON FLAT SURFACE%PREVIOUS SEGMENT ON FLAT
        % SURFACE
    elseif EdgeUpperSeg(Seg-1) < 1 - Thickness
        S = S * (CHORD - CHORD * EdgeUpperSeg(Seg-1) - r + pi * ...
            r / 2) + pi * r / 2 + CHORD * EdgeUpperSeg(Seg-1) - r;
        %PREVIOUS SEGMENT ON TE
    else
        S = S * (pi * r / 2 - r * asin((r + CHORD * ...
            EdgeUpperSeg(Seg-1) - CHORD) / r)) + pi * r / 2 + ...
            CHORD - 2 * r + r * asin((r + CHORD * ...
            EdgeUpperSeg(Seg-1) - CHORD) / r);
    end
end
end
end

j = 1;

while j < num(Seg) + 2

    %ON LE
    if S(j) < pi * r / 2
        X(j + NUM - num(Seg),1,SECT) = r - r * cos(S(j) / r) + xB(1);
        Y(j + NUM - num(Seg),1,SECT) = r * sin(S(j) / r) + yB(1);
        %ON FLAT SURFACE
    elseif S(j) <= CHORD + pi * r / 2 - 2 * r
        X(j + NUM - num(Seg),1,SECT) = S(j) - (pi * r / 2) + r + xB(1);
        Y(j + NUM - num(Seg),1,SECT) = r + yB(1);
        %ON TE
    else
        X(j + NUM - num(Seg),1,SECT) = r * sin((S(j) - pi * r / ...
            2 - CHORD + 2 * r) / r) + CHORD - r + xB(1);
        Y(j + NUM - num(Seg),1,SECT) = r * cos((S(j) - pi * r / ...
            2 - CHORD + 2 * r) / r) + yB(1);
    end

    j = j + 1;

end

if SeePlot == 0
    figure
    plot(X(NUM-num(Seg)+1:NUM+1,1,SECT),Y(NUM-
num(Seg)+1:NUM+1,1,SECT),'o')
    axis equal
end

save Results X Y -APPEND
save Intermediate NUM num -APPEND

```

10. LowerRightVerticalInit.m

```
function LowerRightVertical(SeePlot)
%FUNCTION PLACES ACTUAL GRID POINTS ON UPPER RIGHT BOUNDARY SURFACE

AirfoilWakeInit(1)

load InitData
load PQData
load Intermediate
load Results

SECT = 2;    %FOR FUTURE MODIFICATIONS (WILL ALLOW MULTIPLE SECTORS)

S = Stretch(m(2*SECT), LRP, LRQ);
LRS = fliplr(S);

%DETERMINE X COORDINATE FOR CORRESPONDING Y COORDINATE
%SCALE AND TRANSLATE RIGHT SURFACE - SCALING SAME IN BOTH X AND Y
DIRECTIONS (1-D)

X(m(2*SECT-1),1:m(2*SECT),SECT) = xB(3);
Y(m(2*SECT-1),1:m(2*SECT),SECT) = LRS * (-yB(3) - ...
    Y(m(2*(SECT-1)-1),1,SECT-1)) + Y(m(2*(SECT-1)-1),1,SECT-1);

if SeePlot == 0
    figure
    plot(X(m(2*SECT-1),1:m(2*SECT),SECT), ...
        Y(m(2*SECT-1),1:m(2*SECT),SECT), ...
        X(m(2*SECT-1),1:m(2*SECT),SECT), ...
        Y(m(2*SECT-1),1:m(2*SECT),SECT), 'o')
    axis equal
end

save Results X Y -APPEND
save Intermediate LRS -APPEND
```

11. LowerRightVertical.m

```
function LowerRightVertical
%FUNCTION PLACES ACTUAL GRID POINTS ON UPPER RIGHT BOUNDARY SURFACE

load InitData
load Intermediate
load ResultsPP

SECT = 2;    %FOR FUTURE MODIFICATIONS (WILL ALLOW MULTIPLE SECTORS)

%DETERMINE X COORDINATE FOR CORRESPONDING Y COORDINATE
%SCALE AND TRANSLATE RIGHT SURFACE - SCALING SAME IN BOTH X AND Y
DIRECTIONS (1-D)
Y(m(2*SECT-1),1:m(2*SECT),SECT) = LRS * (-yB(3) - ...
    Y(m(2*(SECT-1)-1),1,SECT-1)) + Y(m(2*(SECT-1)-1),1,SECT-1);

save ResultsPP X Y -APPEND
```

12. UpperRightVerticalInit.m

```
function UpperRightVertical(SeePlot)
%FUNCTION PLACES ACTUAL GRID POINTS ON UPPER RIGHT BOUNDARY SURFACE

AirfoilWakeInit(1)

load InitData
load PQData
load Intermediate
load Results

SECT = 1;    %FOR FUTURE MODIFICATIONS (WILL ALLOW MULTIPLE SECTORS)

S = Stretch(m(2*SECT), URP, URQ);
URS = S;

%DETERMINE X COORDINATE FOR CORRESPONDING Y COORDINATE
%SCALE AND TRANSLATE RIGHT SURFACE - SCALING SAME IN BOTH X AND Y
DIRECTIONS (1-D)

X(m(2*SECT-1),1:m(2*SECT),SECT) = xB(3);
Y(m(2*SECT-1),1:m(2*SECT),SECT) = S * (yB(3) - ...
    Y(m(2*SECT-1),1,SECT)) + Y(m(2*SECT-1),1,SECT);

if SeePlot == 0
    figure
    plot(X(m(2*SECT-1),1:m(2*SECT),SECT), ...
        Y(m(2*SECT-1),1:m(2*SECT),SECT), ...
        X(m(2*SECT-1),1:m(2*SECT),SECT), ...
        Y(m(2*SECT-1),1:m(2*SECT),SECT), 'o')
    axis equal
end

save Results X Y -APPEND
save Intermediate URS -APPEND
```

13. UpperRightVertical.m

```
function UpperRightVertical
%FUNCTION PLACES ACTUAL GRID POINTS ON UPPER RIGHT BOUNDARY SURFACE

load InitData
load Intermediate
load ResultsPP

SECT = 1;    %FOR FUTURE MODIFICATIONS (WILL ALLOW MULTIPLE SECTORS)

%DETERMINE X COORDINATE FOR CORRESPONDING Y COORDINATE
% SCALE AND TRANSLATE RIGHT SURFACE - SCALING SAME IN BOTH X AND Y
% DIRECTIONS (1-D)
Y(m(2*SECT-1),1:m(2*SECT),SECT) = URS * (yB(3) - ...
    Y(m(2*SECT-1),1,SECT)) + Y(m(2*SECT-1),1,SECT);

save ResultsPP X Y -APPEND
```

C. RIGID SURFACE GENERATING FUNCTIONS

1. LowerRigid.m

```
function LowerRigid

load InitData
load Intermediate
load Results

SECT = 2;    %FOR FUTURE MODIFICATIONS (WILL ALLOW MULTIPLE SECTORS)

%FIND MIDPOINT OF X COORDINATE ALONG AIRFOIL SURFACE
done = 0;
i = 1;
while done == 0
    if X(i,m(2*SECT),SECT) - X(1,m(2*SECT),SECT) >= 0.5 * CHORD
        done = 1;
    end
    i = i + 1;
end
Middle = i - 1;

%CALCULATE THE GRID HEIGHT SLOPE FROM THE LEADING EDGE TO THE MIDPOINT
slope1 = (thick2 - thick1) / (X(Middle,m(2*SECT),SECT) - ...
    X(1,m(2*SECT),SECT));

%CALCULATE THE GRID HEIGHT SLOPE FROM THE MIDPOINT TO THE TRAILING EDGE
slope2 = (thick3 - thick2) / (X(NUM+1,m(2*SECT),SECT) - ...
    X(Middle,m(2*SECT),SECT));

%CALCULATE THE GRID THICKNESS AT EACH POINT
for i = 1:Middle
    delta(i) = slope1 * (X(i,m(2*SECT),SECT) - ...
        X(1,m(2*SECT),SECT)) + thick1;
end

for i = Middle+1:NUM+1
    delta(i) = slope2 * (X(i,m(2*SECT),SECT) - ...
        X(Middle,m(2*SECT),SECT)) + thick2;
end

%FIND THE SLOPE OF THE LINE PERPENDICULAR TO THE AIRFOIL SURFACE AT
% EACH POINT
if AirfoilType == 1
    nslope = NacaSlope((X(1:NUM+1,m(2*SECT),SECT) - ...
        X(1,m(2*SECT),SECT)) / (X(NUM+1,m(2*SECT),SECT) - ...
        X(1,m(2*SECT),SECT)),SECT);
elseif AirfoilType == 2
    nslope = RTESlope((X(1:NUM+1,m(2*SECT),SECT) - ...
        X(1,m(2*SECT),SECT)) / ...
        (X(NUM+1,m(2*SECT),SECT) - X(1,m(2*SECT),SECT)),SECT);
else
    load ReadInLower
```

```

end

%CALCULATE THE AVERAGE NORMAL SLOPE OF THE AIRFOIL TRAILING EDGE
% AND THE BEGINNING OF THE AIRFOIL WAKE
%ASSUME nslope(NUM+1) IS ALWAYS NEGATIVE
if AirfoilType == 1
    nslope(NUM+1) = -tan((atan(abs(nslope(NUM+1))) + (pi / 2)) / 2);
else
    nslope(NUM+1) = nslope(NUM) / 2;
end

for i = 1:NUM+1

    %CALCULATE THE FIRST CIRCUMFERENTIAL GRID LINE ADJACENT TO THE
    % AIRFOIL SURFACE - ASSUME nslope(i) IS ONLY = 0 AT LE
    if nslope(i) >= 0
        %MOST OF AIRFOIL WITH POSITIVE AIRFOIL SLOPE
        %AIRFOIL LEADING EDGE MUST SEPARATE SECTORS
        X(i,m(2*SECT)-1,SECT) = X(i,m(2*SECT),SECT) - ...
            cos(atan(nslope(i))) * delta(i);
        Y(i,m(2*SECT)-1,SECT) = Y(i,m(2*SECT),SECT) - ...
            sin(atan(nslope(i))) * delta(i);
    else
        %AROUND LEADING EDGE WITH NEGATIVE AIRFOIL SLOPE
        X(i,m(2*SECT)-1,SECT) = X(i,m(2*SECT),SECT) + ...
            cos(atan(nslope(i))) * delta(i);
        Y(i,m(2*SECT)-1,SECT) = Y(i,m(2*SECT),SECT) + ...
            sin(atan(nslope(i))) * delta(i);
    end    %END IF nslope >= 0

end    %END for i = 1:NUM+1

%PLACE SECOND ROW POINTS IN RIGID SURFACE ON AIRFOIL WAKE
for i = NUM+2:EDGER
    X(i,m(2*SECT)-1,SECT) = X(i,m(2*SECT),SECT);
    Y(i,m(2*SECT)-1,SECT) = Y(i,m(2*SECT),SECT) - thick3;
end

%PLACE REMAINING POINTS IN RIGID SURFACE OVER THE AIRFOIL FROM THE
% LEADING EDGE TO APPROXIMATELY SIZESMOOTH * CHORD BEFORE THE
% TRAILING EDGE - PLACE REMAINING POINTS BY ASSUMING RADIAL GRIDLINES
% REMAIN PERPENDICULAR TO AIRFOIL - ALSO, INITIALLY ASSUME CELL
% HEIGHT IS DICTATED SOLELY BY LEFT SURFACE DISTRIBUTION
%ACCOUNT FOR CELL HEIGHT VARIATION DUE TO thick1, 2 and 3
% USE HYPERBOLIC DISTRIBUTION TO SMOOTH VARIATION - ADD ADDITIONAL
% DELTA - DO NOT CHANGE OVERALL HEIGHT OF RIGID SURFACE DETERMINED BY
% USING CELL HEIGHT DICTATED BY LEFT SURFACE

p = (LSS(2:EDGEL-1) - LSS(2)) / (LSS(EDGEL-1) - LSS(2)) * 4 - 2;
q = tanh(p) - 1;
q = q ./ q(1);
q(EDGEL-2) = 0;

i = 2;
deltaLE1 = sqrt( (X(1,m(2*SECT)-2,SECT) - X(1,m(2*SECT)-1,SECT)) * ...

```

```

(X(1,m(2*SECT)-2,SECT) - X(1,m(2*SECT)-1,SECT)) + ...
(Y(1,m(2*SECT)-2,SECT) - Y(1,m(2*SECT)-1,SECT)) * ...
(Y(1,m(2*SECT)-2,SECT) - Y(1,m(2*SECT)-1,SECT)) );

while X(i-1,m(2*SECT),SECT) < X(NUM+1,m(2*SECT),SECT) - ...
    sizesmoothL * CHORD
    if nslope(i) >= 0
        %NEAR AIRFOIL LEADING EDGE WITH NEGATIVE AIRFOIL SLOPE
        X(i,m(2*SECT)-2,SECT) = X(i,m(2*SECT)-1,SECT) - ...
            cos(atan(nslope(i))) * (deltaLE1 + q(1) * (delta(i) - ...
                deltaLE1));
        Y(i,m(2*SECT)-2,SECT) = Y(i,m(2*SECT)-1,SECT) - ...
            sin(atan(nslope(i))) * (deltaLE1 + q(1) * (delta(i) - ...
                deltaLE1));
    else
        %REMAINDER OF AIRFOIL WITH POSITIVE AIRFOIL SLOPE
        X(i,m(2*SECT)-2,SECT) = X(i,m(2*SECT)-1,SECT) + ...
            cos(atan(nslope(i))) * (deltaLE1 + q(1) * (delta(i) - ...
                deltaLE1));
        Y(i,m(2*SECT)-2,SECT) = Y(i,m(2*SECT)-1,SECT) + ...
            sin(atan(nslope(i))) * (deltaLE1 + q(1) * (delta(i) - ...
                deltaLE1));
    end
    i = i + 1;
end

%PLACE THIRD POINT ON VERTICAL LINE AT RIGHT EDGE OF THE RIGID SURFACE
X(EDGER,m(2*SECT)-2,SECT) = X(EDGER,m(2*SECT)-1,SECT);
Y(EDGER,m(2*SECT)-2,SECT) = Y(EDGER,m(2*SECT)-1,SECT) - ...
    (deltaLE1 + q(1) * (thick3 - deltaLE1));

EDGEEmR = i - 1;

for j = m(2*SECT)-3:-1:m(2*SECT)-EDGEEmR+1
    deltaLE = sqrt( (X(1,j,SECT) - X(1,j+1,SECT)) * (X(1,j,SECT) - ...
        X(1,j+1,SECT)) + (Y(1,j,SECT) - Y(1,j+1,SECT)) * ...
        (Y(1,j,SECT) - Y(1,j+1,SECT)) );
    for i = 2:EDGEEmR
        if nslope(i) >= 0
            %NEAR AIRFOIL LEADING EDGE WITH NEGATIVE AIRFOIL SLOPE
            X(i,j,SECT) = X(i,j+1,SECT) - cos(atan(nslope(i))) * ...
                (deltaLE + q(m(2*SECT)-1-j) * (delta(i) - deltaLE1));
            Y(i,j,SECT) = Y(i,j+1,SECT) - sin(atan(nslope(i))) * ...
                (deltaLE + q(m(2*SECT)-1-j) * (delta(i) - deltaLE1));
        else
            %MAJORITY OF AIRFOIL WITH POSITIVE AIRFOIL SLOPE
            X(i,j,SECT) = X(i,j+1,SECT) + cos(atan(nslope(i))) * ...
                (deltaLE + q(m(2*SECT)-1-j) * (delta(i) - deltaLE1));
            Y(i,j,SECT) = Y(i,j+1,SECT) + sin(atan(nslope(i))) * ...
                (deltaLE + q(m(2*SECT)-1-j) * (delta(i) - deltaLE1));
        end
    end
end
X(EDGER,j,SECT) = X(EDGER,j+1,SECT);
Y(EDGER,j,SECT) = Y(EDGER,j+1,SECT) - (deltaLE + ...
    q(m(2*SECT)-1-j) * (thick3 - deltaLE1));

```

```

end

%PLACE POINTS ON THE BOTTOM OF THE RIGID SURFACE BETWEEN EDGEmR AND
%   EDGER
%   USE HYPERBOLIC TANGENT DISTRIBUTION
%   MATCH SLOPES AT BOTH SIDES BY TAPERING SLOPE WITH HYBERBOLIC
%   TANGENT DISTRIBUTIONS

%USE SAME SPACING FOR X COORDINATES ON TOP SIDE OF RIGID SURFACE AS ON
%   SURFACE OF AIRFOIL AND WAKE
%IF AirfoilType = 2, USE SAME ARC LENGTH DISTRIBUTION ALONG AIRFOIL TO
%   PRODUCE X COORDINATE DISTRIBUTION

if AirfoilType == 1
    %CALCULATE THE X COORDINATE ON BOTTOM OF THE RIGID SURFACE ALIGNED
    %   WITH THE AIRFOIL TRAILING EDGE - POINT WILL FALL ON LINE FROM
    %   THE TRAILING EDGE TO THE BOTTOM OF THE RIGID SURFACE WITH
    %   AVERAGE PERPENDICULAR SLOPE BETWEEN AIRFOIL TRAILING EDGE AND
    %   WAKE - THE Y COORDINATE IS THE AVERAGE BETWEEN THE Y COORDINATE
    %   AT EDGEmR AND EDGER

    xTE = (((Y(EDGEmR,m(2*SECT))-EDGEL+1,SECT) + ...
        Y(EDGER,m(2*SECT)-EDGEL+1,SECT)) / 2) - ...
        Y(NUM+1,m(2*SECT),SECT)) / ...
        nslope(NUM+1)) + X(NUM+1,m(2*SECT),SECT);

    temp = (X(EDGEmR:NUM+1,m(2*SECT),SECT) - ...
        X(EDGEmR,m(2*SECT),SECT)) / ...
        (X(NUM+1,m(2*SECT),SECT) - X(EDGEmR,m(2*SECT),SECT));
    X(EDGEmR:NUM+1,m(2*SECT)-EDGEL+1,SECT) = temp * (xTE - ...
        X(EDGEmR,m(2*SECT)-EDGEL+1,SECT)) + ...
        X(EDGEmR,m(2*SECT)-EDGEL+1,SECT);
else
    temp=ArcLength(X(EDGEmR:NUM+1,m(2*SECT),SECT),Y(EDGEmR:NUM+1,m(2*SECT),
        SECT),NUM+2-EDGEmR);
    temp = temp / temp(NUM+2-EDGEmR);
    X(EDGEmR:NUM+1,m(2*SECT)-EDGEL+1,SECT) = temp * ...
        (xTE - X(EDGEmR,m(2*SECT)-EDGEL+1,SECT)) + ...
        X(EDGEmR,m(2*SECT)-EDGEL+1,SECT);
end

temp = (X(NUM+1:EDGER,m(2*SECT),SECT) - X(NUM+1,m(2*SECT),SECT)) ./ ...
    (X(EDGER,m(2*SECT),SECT) - X(NUM+1,m(2*SECT),SECT));
X(NUM+2:EDGER,m(2*SECT)-EDGEL+1,SECT) = temp(2:EDGER-NUM) * ...
    (X(EDGER,m(2*SECT)-EDGEL+1,SECT) - xTE) + xTE;

%CREATE NEW "S" VECTOR WITH NEWLY DETERMINED X-COORDINATES
temp = 0;
temp = (X(EDGEmR:EDGER,m(2*SECT)-EDGEL+1,SECT) - ...
    X(EDGEmR,m(2*SECT)-EDGEL+1,SECT)) ./ ...
    (X(EDGER,m(2*SECT)-EDGEL+1,SECT) - ...
    X(EDGEmR,m(2*SECT)-EDGEL+1,SECT));
L = EDGER - EDGEmR + 1;

```

```

%USE HYPERBOLIC TANGENT DISTRIBUTION TO SMOOTH DELTA IN HEIGHT BETWEEN
%   EDGEmR AND EDGER - ALSO USE HYPERBOLIC DISTRIBUTION TO SMOOTH DELTA
%   IN SLOPES AT EDGEmR AND EDGER

%HYPERBOLIC TANGENT SMOOTHING OF SLOPE AT EDGEmR
i = 1;
while temp(i) <= 0.8
    p(i) = temp(i);
    i = i + 1;
end
p = (p / temp(i-1)) * 4 - 2;
q = -abs(tanh(p));
p = ((p + 2) / 4) * temp(i-1);
q = q - q(1);
q = q ./ sum(q);
p(i:L) = temp(i:L);
q(i:L) = 0;
SlopeL(1) = -1 / nslope(EDGEmR); %SLOPE AT EDGEmR
SlopeL(L) = 0;

SlopeDelta = SlopeL(L) - SlopeL(1);
for j = 2:L-1
    %area KEEPS TRACK OF AREA UNDER TANH CURVE
    area(j-1) = (q(j-1) + 0.5 * (q(j) - q(j-1))) * ...
        (p(j) - p(j-1));
    SlopeL(j) = (SlopeDelta) * sum(area(1:j-1));
end

SlopeL(2:L-1) = SlopeL(1) + SlopeL(2:L-1) / sum(area);

delta1(1,1) = 0;
for i = 2:L
    delta1(i,1) = delta1(i-1,1) + SlopeL(i-1) * ...
        (X(EDGEmR+i-1,m(2*SECT)-EDGEL+1,SECT) - ...
        X(EDGEmR+i-2,m(2*SECT)-EDGEL+1,SECT));
end

%CALCULATE SLOPE OF LINE CONNECTING LOWER RIGID SURFACE FROM EDGEmR TO
EDGER
SlopeB = (Y(EDGER,m(2*SECT)-EDGEL+1,SECT) - ...
    Y(EDGEmR,m(2*SECT)-EDGEL+1,SECT)) / ...
    (X(EDGER,m(2*SECT)-EDGEL+1,SECT) - ...
    X(EDGEmR,m(2*SECT)-EDGEL+1,SECT));

%DETERMINE THE Y COORDINATE DISTRIBUTION TO TAPER FROM EDGEmR TO EDGER
%   OF THE RIGID SURFACE ALONG THE BOTTOM OF THE RIGID SURFACE
%   A LINEAR DISTRIBUTION IS USED
p = 0;
q = 0;
p = temp;
q = SlopeB * temp;
q = q - q(1);
q = q ./ sum(q);

TotDelta = Y(EDGEmR,m(2*SECT)-EDGEL+1,SECT) - ...

```

```

Y(EDGER,m(2*SECT)-EDGEL+1,SECT) + delta1(L);

for i = EDGEmR+1:EDGER-1
    %area KEEPS TRACK OF AREA UNDER CURVE
    area(i-EDGEmR) = (q(i-EDGEmR) + 0.5 * (q(i-EDGEmR+1) - ...
        q(i-EDGEmR))) * (p(i-EDGEmR+1) - p(i-EDGEmR));
    Y(i,m(2*SECT)-EDGEL+1,SECT) = (TotDelta) * sum(area(1:i-EDGEmR));
end

Y(EDGEmR+1:EDGER-1,m(2*SECT)-EDGEL+1,SECT) = ...
    Y(EDGEmR,m(2*SECT)-EDGEL+1,SECT) + delta1(2:L-1) - ...
    Y(EDGEmR+1:EDGER-1,m(2*SECT)-EDGEL+1,SECT) / sum(area);

%APPROXIMATE THE SLOPE AT EACH POINT ALONG THE BOTTOM OF THE RIGID
% BETWEEN EDGEmR AND EDGER
for i = 1:EDGER-EDGEmR-2
    if (Y(EDGEmR+i+1,m(2*SECT)-EDGEL+1,SECT) - ...
        Y(EDGEmR+i-1,m(2*SECT)-EDGEL+1,SECT)) ~= 0
        nslopeU(i) = (X(EDGEmR+i-1,m(2*SECT)-EDGEL+1,SECT) - ...
            X(EDGEmR+i+1,m(2*SECT)-EDGEL+1,SECT)) / ...
            (Y(EDGEmR+i+1,m(2*SECT)-EDGEL+1,SECT) - ...
            Y(EDGEmR+i-1,m(2*SECT)-EDGEL+1,SECT));
    else
        nslopeU(i) = 1e6;
    end
end

%SLOPE AT EDGER-1 - AVERAGE SLOPE BETWEEN POINT BEFORE AND POINT AFTER
if nslopeU(EDGER-EDGEmR-2) >= 0
    nslopeU(EDGER-EDGEmR-1) = tan((atan(nslopeU(EDGER-EDGEmR-2)) + ...
        (pi/2)) / 2);
else
    nslopeU(EDGER-EDGEmR-1) = -tan((atan(-nslopeU(EDGER-EDGEmR-2)) + ...
        (pi/2)) / 2);
end

nslopeU(EDGER-EDGEmR) = 1e6;

%DETERMINE nslope FOR AIRFOIL WAKE IN RIGID SURFACE
nslope(NUM+2:EDGER) = -1e6;

LRnSlope = [nslope(1:EDGEmR), nslopeU];

%PLACE REMAINING POINTS INSIDE RIGID SURFACE BY CONNECTING AIRFOIL
% SURFACE TO BOTTOM OF RIGID SURFACE WITH STRAIGHT LINES
% USE DISTRIBUTION SPECIFIED BY LEFT SURFACE CORRECTED FOR BY CELL
% HEIGHT DICTATED BY thick1, 2, and 3

delta(NUM+2:EDGER) = thick3;
S = 0;

for i = EDGEmR+1:EDGER-1

    p = (LSS(2:EDGEL-1) - LSS(2)) / (LSS(EDGEL-1) - LSS(2)) * 4 - 2;
    q = tanh(p) - 1;

```

```

q = q ./ q(1);
q(EDGEL-2) = 0;

TotDelta = sqrt( (X(i,m(2*SECT))-EDGEL+1,SECT) - ...
    X(i,m(2*SECT),SECT)) * ...
    (X(i,m(2*SECT))-EDGEL+1,SECT) - X(i,m(2*SECT),SECT)) + ...
    (Y(i,m(2*SECT))-EDGEL+1,SECT) - Y(i,m(2*SECT),SECT)) * ...
    (Y(i,m(2*SECT))-EDGEL+1,SECT) - Y(i,m(2*SECT),SECT)) );

%CREATE NEW "S" VECTOR BY SCALING LENGTH
S = TotDelta * ((X(1,m(2*SECT):-1:m(2*SECT))-EDGEL+1,SECT) - ...
    X(1,m(2*SECT),SECT)) / ((X(1,m(2*SECT))-EDGEL+1,SECT) - ...
    X(1,m(2*SECT),SECT)));

for j = 2:EDGEL-1
    deltaLEInit(j-1) = S(j+1) - S(j);
end

S(2) = S(1) + delta(i);
for j = 3:EDGEL
    S(j) = S(j-1) + deltaLEInit(j-2) + q(j-2) * (delta(i) - ...
        deltaLEInit(1));
end

%ENFORCE NORMAL AT BOTTOM OF RIGID SURFACE
if nslopeU(i-EDGEEmR) >= 0
    X(i,m(2*SECT)-EDGEL+2,SECT) = X(i,m(2*SECT)-EDGEL+1,SECT) + ...
        cos(atan(nslopeU(i-EDGEEmR))) * (S(EDGEL) - S(EDGEL-1));
    Y(i,m(2*SECT)-EDGEL+2,SECT) = Y(i,m(2*SECT)-EDGEL+1,SECT) + ...
        sin(atan(nslopeU(i-EDGEEmR))) * (S(EDGEL) - S(EDGEL-1));
else
    X(i,m(2*SECT)-EDGEL+2,SECT) = X(i,m(2*SECT)-EDGEL+1,SECT) - ...
        cos(atan(nslopeU(i-EDGEEmR))) * (S(EDGEL) - S(EDGEL-1));
    Y(i,m(2*SECT)-EDGEL+2,SECT) = Y(i,m(2*SECT)-EDGEL+1,SECT) - ...
        sin(atan(nslopeU(i-EDGEEmR))) * (S(EDGEL) - S(EDGEL-1));
end

%FIND SLOPE OF LINE CONNECTING AIRFOIL AND BOTTOM OF RIGID SURFACE
% ASSUMING LINEAR CONNECTION
if X(i,m(2*SECT)-EDGEL+2,SECT) ~= X(i,m(2*SECT)-1,SECT)
    Slope = (Y(i,m(2*SECT)-EDGEL+2,SECT) - ...
        Y(i,m(2*SECT)-1,SECT)) / ...
        (X(i,m(2*SECT)-EDGEL+2,SECT) - X(i,m(2*SECT)-1,SECT));
else
    Slope = -1e6;
end

%USE HYPERBOLIC DISTRIBUTION TO SMOOTH DELTA IN NORMAL SLOPES AT
% AIRFOIL SURFACE/WAKE AND TOP OF RIGID SURFACE BETWEEN EDGEEmR AND
% EDGER
%HYPERBOLIC TANGENT TAPERING OF NORMAL TO AIRFOIL/WAKE SURFACE
k = 1;
p = 0;
Snew = S / S(EDGEL);

```

```

lim = bendslopebot;
while Snew(3) > lim
    lim = lim + 0.01;
end
while Snew(k) <= lim
    p(k) = Snew(k);
    k = k + 1;
end

p = (p / Snew(k-1)) * 4 - 2;
qb = (-tanh(p) + 1);
qb = qb / qb(1);
p = ((p + 2) / 4) * Snew(k-1);
p(k:EDGEL-1) = Snew(k:EDGEL-1);
qb(k:EDGEL-1) = 0;

%HYPERBOLIC TANGENT TAPERING OF SLOPE AT EDGEL IF bendslopetop ~= 0
p = 0;
k = EDGEL-1;
lim = bendslopetop;
while Snew(EDGEL-3) < 1 - lim
    lim = lim + 0.01;
end
while Snew(k) >= 1 - lim
    p(k) = Snew(k);
    k = k - 1;
end

p(k+1:EDGEL-1) = ((p(k+1:EDGEL-1) - p(k+1)) / ...
    (p(EDGEL-1) - p(k+1))) * 4 - 2;
qt(k+1:EDGEL-1) = tanh(p(k+1:EDGEL-1)) + 1;
qt(k+1:EDGEL-1) = qt(k+1:EDGEL-1) / qt(EDGEL-1);
qt(1:k) = 0;
p(k+1:EDGEL-1) = (((p(k+1:EDGEL-1) + 2) / 4) * (Snew(EDGEL-1) - ...
    Snew(k+1))) + Snew(k+1);
p(1:k) = Snew(1:k);

Theta = atan(Slope);
ThetaDiffBot = Theta - atan(nslope(i));

%DRAW STRAIGHT LINE, BUT ALSO TAPER NORMAL SLOPE NEAR AIRFOIL
% SURFACE
for j = 3:EDGEL-2
    if Theta - qb(j-1) * ThetaDiffBot < 0
        X(i,m(2*SECT)+1-j,SECT) = X(i,m(2*SECT)+2-j,SECT) + ...
            (S(j) - S(j-1)) * cos(Theta - qb(j-1) * ThetaDiffBot);
        Y(i,m(2*SECT)+1-j,SECT) = Y(i,m(2*SECT)+2-j,SECT) + ...
            (S(j) - S(j-1)) * sin(Theta - qb(j-1) * ThetaDiffBot);
    else
        X(i,m(2*SECT)+1-j,SECT) = X(i,m(2*SECT)+2-j,SECT) - ...
            (S(j) - S(j-1)) * cos(Theta - qb(j-1) * ThetaDiffBot);
        Y(i,m(2*SECT)+1-j,SECT) = Y(i,m(2*SECT)+2-j,SECT) - ...
            (S(j) - S(j-1)) * sin(Theta - qb(j-1) * ThetaDiffBot);
    end
end
end

```

```

%TAPER NORMAL SLOPE AT BOTTOM OF RIGID SURFACE
for j = 3:EDGEL-2
    if nslopeU(i-EDGEEmR) >= 0
        X(i,m(2*SECT)+1-j,SECT) = X(i,m(2*SECT)+1-j,SECT) + ...
            qt(j-1) * ( ( X(i,m(2*SECT)-EDGEL+1,SECT) + ...
                (S(EDGEL) - S(j)) * ...
                cos(atan(LRnSlope(i))) ) - X(i,m(2*SECT)+1-j,SECT) );
        Y(i,m(2*SECT)+1-j,SECT) = Y(i,m(2*SECT)+1-j,SECT) + ...
            qt(j-1) * ( ( Y(i,m(2*SECT)-EDGEL+1,SECT) + ...
                (S(EDGEL) - S(j)) * ...
                sin(atan(LRnSlope(i))) ) - Y(i,m(2*SECT)+1-j,SECT) );
    else
        X(i,m(2*SECT)+1-j,SECT) = X(i,m(2*SECT)+1-j,SECT) + ...
            qt(j-1) * ( ( X(i,m(2*SECT)-EDGEL+1,SECT) - ...
                (S(EDGEL) - S(j)) * ...
                cos(atan(LRnSlope(i))) ) - X(i,m(2*SECT)+1-j,SECT) );
        Y(i,m(2*SECT)+1-j,SECT) = Y(i,m(2*SECT)+1-j,SECT) + ...
            qt(j-1) * ( ( Y(i,m(2*SECT)-EDGEL+1,SECT) - ...
                (S(EDGEL) - S(j)) * ...
                sin(atan(LRnSlope(i))) ) - Y(i,m(2*SECT)+1-j,SECT) );
    end
end

end

save Intermediate LRnSlope -APPEND
save Results X Y -APPEND

```

2. UpperRigid.m

```

function UpperRigid

load InitData
load Intermediate
load Results

SECT = 1;    %FOR FUTURE MODIFICATIONS (WILL ALLOW MULTIPLE SECTORS)

%FIND MIDPOINT OF X COORDINATE ALONG AIRFOIL SURFACE
done = 0;
i = 1;
while done == 0
    if X(i,1,SECT) - X(1,1,SECT) >= 0.5 * CHORD
        done = 1;
    end
    i = i + 1;
end
Middle = i - 1;

%CALCULATE THE GRID HEIGHT SLOPE FROM THE LEADING EDGE TO THE MIDPOINT
slope1 = (thick2 - thick1) / (X(Middle,1,SECT) - X(1,1,SECT));

%CALCULATE THE GRID HEIGHT SLOPE FROM THE MIDPOINT TO THE TRAILING EDGE
slope2 = (thick3 - thick2) / (X(NUM+1,1,SECT) - X(Middle,1,SECT));

```

```

%CALCULATE THE GRID THICKNESS AT EACH POINT
for i = 1:Middle
    delta(i) = slope1 * (X(i,1,SECT) - X(1,1,SECT)) + thick1;
end

for i = Middle+1:NUM+1
    delta(i) = slope2 * (X(i,1,SECT) - X(Middle,1,SECT)) + thick2;
end

%FIND THE SLOPE OF THE LINE PERPINDICULAR TO THE AIRFOIL SURFACE AT
EACH POINT
if AirfoilType == 1
    nslope = NacaSlope((X(1:NUM+1,1,SECT) - X(1,1,SECT)) / ...
        (X(NUM+1,1,SECT) - X(1,1,SECT)), SECT);
elseif AirfoilType == 2
    nslope = RTESlope((X(1:NUM+1,1,SECT) - X(1,1,SECT)) / ...
        (X(NUM+1,1,SECT) - X(1,1,SECT)), SECT);
else
    load ReadInUpper
end

%CALCULATE THE AVERAGE NORMAL SLOPE OF THE AIRFOIL TRAILING EDGE
% AND THE BEGINNING OF THE AIRFOIL WAKE
%ASSUME nslope(NUM+1) IS ALWAYS POSITIVE
if AirfoilType == 1
    nslope(NUM+1) = tan((atan(nslope(NUM+1)) + (pi / 2)) / 2);
else
    nslope(NUM+1) = nslope(NUM) / 2;
end

for i = 1:NUM+1

    %CALCULATE THE FIRST CIRCUMFERENTIAL GRID LINE ADJACENT TO THE
    % AIRFOIL SURFACE
    %ASSUME nslope(i) IS ONLY = 0 AT LE
    if nslope(i) > 0
        %MOST OF AIRFOIL WITH NEGATIVE AIRFOIL SLOPE
        %AIRFOIL LEADING EDGE MUST SEPARATE SECTORS
        X(i,2,SECT) = X(i,1,SECT) + cos(atan(nslope(i))) * delta(i);
        Y(i,2,SECT) = Y(i,1,SECT) + sin(atan(nslope(i))) * delta(i);
    else
        %AROUND LEADING EDGE WITH POSITIVE AIRFOIL SLOPE
        X(i,2,SECT) = X(i,1,SECT) - cos(atan(nslope(i))) * delta(i);
        Y(i,2,SECT) = Y(i,1,SECT) - sin(atan(nslope(i))) * delta(i);
    end
end

end %END for i = 1:NUM+1

%PLACE SECOND ROW POINTS IN RIGID SURFACE ON AIRFOIL WAKE
for i = NUM+2:EDGER
    X(i,2,SECT) = X(i,1,SECT);
    Y(i,2,SECT) = thick3 + Y(i,1,SECT);
end

```

```

%PLACE REMAINING POINTS IN RIGID SURFACE OVER THE AIRFOIL FROM THE
%   LEADING EDGE TO   APPROXIMATELY SIZESMOOTH * CHORD BEFORE THE
%   TRAILING EDGE
%PLACE REMAINING POINTS BY ASSUMING RADIAL GRIDLINES REMAIN
%   PERPENDICULAR TO AIRFOIL - ALSO, INITIALLY ASSUME CELL HEIGHT IS
%   DICTATED SOLELY BY LEFT SURFACE DISTRIBUTION
%ACCOUNT FOR CELL HEIGHT VARIATION DUE TO thick1, 2 and 3
%   USE HYPERBOLIC DISTRIBUTION TO SMOOTH VARIATION - ADD ADDITIONAL
%   DELTA
%DO NOT CHANGE OVERALL HEIGHT OF RIGID SURFACE DETERMINED BY USING
%   CELL HEIGHT DICTATED BY LEFT SURFACE

p = (LSS(2:EDGEL-1) - LSS(2)) / (LSS(EDGEL-1) - LSS(2)) * 4 - 2;
q = tanh(p) - 1;
q = q ./ q(1);
q(EDGEL-2) = 0;

i = 2;
deltaLE1 = sqrt( (X(1,3,SECT) - X(1,2,SECT)) * (X(1,3,SECT) - ...
    X(1,2,SECT)) + (Y(1,3,SECT) - Y(1,2,SECT)) * ...
    (Y(1,3,SECT) - Y(1,2,SECT)) );

while X(i-1,1,SECT) < X(NUM+1,1,SECT) - sizesmooth * CHORD
    if nslope(i) > 0
        %MAJORITY OF AIRFOIL WITH NEGATIVE AIRFOIL SLOPE
        X(i,3,SECT) = X(i,2,SECT) + cos(atan(nslope(i))) * ...
            (deltaLE1 + q(1) * (delta(i) - deltaLE1));
        Y(i,3,SECT) = Y(i,2,SECT) + sin(atan(nslope(i))) * ...
            (deltaLE1 + q(1) * (delta(i) - deltaLE1));
    else
        %NEAR AIRFOIL LEADING EDGE WITH POSITIVE AIRFOIL SLOPE
        X(i,3,SECT) = X(i,2,SECT) - cos(atan(nslope(i))) * ...
            (deltaLE1 + q(1) * (delta(i) - deltaLE1));
        Y(i,3,SECT) = Y(i,2,SECT) - sin(atan(nslope(i))) * ...
            (deltaLE1 + q(1) * (delta(i) - deltaLE1));
    end
    i = i + 1;
end

%PLACE THIRD POINT ON VERTICAL LINE AT RIGHT EDGE OF THE RIGID SURFACE
X(EDGER,3,SECT) = X(EDGER,2,SECT);
Y(EDGER,3,SECT) = Y(EDGER,2,SECT) + deltaLE1 + q(1) * ...
    (thick3 - deltaLE1);

EDGEEmR = i - 1;

for j = 4:EDGEL
    deltaLE = sqrt( (X(1,j,SECT) - X(1,j-1,SECT)) * (X(1,j,SECT) - ...
        X(1,j-1,SECT)) + ...
        (Y(1,j,SECT) - Y(1,j-1,SECT)) * (Y(1,j,SECT) - Y(1,j-1,SECT)) );
    for i = 2:EDGEEmR
        if nslope(i) > 0
            %MAJORITY OF AIRFOIL WITH NEGATIVE AIRFOIL SLOPE
            X(i,j,SECT) = X(i,j-1,SECT) + cos(atan(nslope(i))) * ...
                (deltaLE + q(j-2) * (delta(i) - deltaLE1));

```

```

        Y(i,j,SECT) = Y(i,j-1,SECT) + sin(atan(nslope(i))) * ...
            (deltaLE + q(j-2) * (delta(i) - deltaLE1));
    else
        %NEAR AIRFOIL LEADING EDGE WITH POSITIVE AIRFOIL SLOPE
        X(i,j,SECT) = X(i,j-1,SECT) - cos(atan(nslope(i))) * ...
            (deltaLE + q(j-2) * (delta(i) - deltaLE1));
        Y(i,j,SECT) = Y(i,j-1,SECT) - sin(atan(nslope(i))) * ...
            (deltaLE + q(j-2) * (delta(i) - deltaLE1));
    end
end
X(EDGER,j,SECT) = X(EDGER,j-1,SECT);
Y(EDGER,j,SECT) = Y(EDGER,j-1,SECT) + (deltaLE + q(j-2) * ...
    (thick3 - deltaLE1));
end

%PLACE POINTS ON THE TOP OF THE RIGID SURFACE BETWEEN EDGEmR AND EDGER
%   USE HYPERBOLIC TANGENT DISTRIBUTION
%   MATCH SLOPES AT BOTH SIDES BY TAPERING SLOPE WITH HYPERBOLIC
%   TANGENT DISTRIBUTIONS

%USE SAME SPACING FOR X COORDINATES ON TOP SIDE OF RIGID SURFACE AS ON
%   SURFACE OF AIRFOIL AND WAKE
%IF AirfoilType = 2, USE SAME ARC LENGTH DISTRIBUTION ALONG AIRFOIL TO
%   PRODUCE X COORDINATE DISTRIBUTION

if AirfoilType == 1
    %CALCULATE THE X COORDINATE ON TOP OF THE RIGID SURFACE ALIGNED
    %   WITH THE AIRFOIL TRAILING EDGE - POINT WILL FALL ON LINE FROM
    THE
    %   TRAILING EDGE TO THE TOP OF THE RIGID SURFACE WITH AVERAGE
    %   PERPENDICULAR SLOPE BETWEEN AIRFOIL TRAILING EDGE AND WAKE
    %   THE Y-COORDINATE IS THE AVERAGE BETWEEN THE Y-COORDINATE AT
    %   EDGEmR AND EDGER

    xTE = (((Y(EDGEmR,EDGEL,SECT) + Y(EDGER,EDGEL,SECT)) / 2) - ...
        Y(NUM+1,1,SECT)) / nslope(NUM+1)) + X(NUM+1,1,SECT);

    temp = (X(EDGEmR:NUM+1,1,SECT) - X(EDGEmR,1,SECT)) ./ ...
        (X(NUM+1,1,SECT) - X(EDGEmR,1,SECT));
    X(EDGEmR:NUM+1,EDGEL,SECT) = temp * ...
        (xTE - X(EDGEmR,EDGEL,SECT)) + X(EDGEmR,EDGEL,SECT);
else
    temp = ArcLength(X(EDGEmR:NUM+1,1,SECT),Y(EDGEmR:NUM+1,1,SECT), ...
        NUM+2-EDGEmR);
    temp = temp ./ temp(NUM+2-EDGEmR);
    X(EDGEmR:NUM+1,EDGEL,SECT) = temp * ...
        (xTE - X(EDGEmR,EDGEL,SECT)) + X(EDGEmR,EDGEL,SECT);
end

temp = (X(NUM+1:EDGER,1,SECT) - X(NUM+1,1,SECT)) ./ ...
    (X(EDGER,1,SECT) - X(NUM+1,1,SECT));
X(NUM+2:EDGER,EDGEL,SECT) = temp(2:EDGER-NUM) * ...
    (X(EDGER,EDGEL,SECT) - xTE) + xTE;

%CREATE NEW "S" VECTOR WITH NEWLY DETERMINED X-COORDINATES

```

```

temp = 0;
temp = (X(EDGEEmR:EDGER,EDGEL,SECT) - X(EDGEEmR,EDGEL,SECT)) ./ ...
      (X(EDGER,EDGEL,SECT) - X(EDGEEmR,EDGEL,SECT));
L = EDGER - EDGEEmR + 1;

%USE HYBERBOLIC TANGENT DISTRIBUTION TO SMOOTH DELTA IN HEIGHT BETWEEN
%   EDGEEmR AND EDGER ALSO USE HYBERBOLIC DISTRIBUTION TO SMOOTH DELTA
%   IN SLOPES AT EDGEEmR AND EDGER

%HYBERBOLIC TANGENT SMOOTHING OF SLOPE AT EDGEEmR
i = 1;
while temp(i) <= 0.8
    p(i) = temp(i);
    i = i + 1;
end
p = (p / temp(i-1)) * 4 - 2;
q = -abs(tanh(p));
p = ((p + 2) / 4) * temp(i-1);
q = q - q(1);
q = q ./ sum(q);
p(i:L) = temp(i:L);
q(i:L) = 0;
SlopeL(1) = -1 / nslope(EDGEEmR);    %SLOPE AT EDGEEmR
SlopeL(L) = 0;

SlopeDelta = SlopeL(L) - SlopeL(1);
for j = 2:L-1
    %area KEEPS TRACK OF AREA UNDER TANH CURVE
    area(j-1) = (q(j-1) + 0.5 * (q(j) - q(j-1))) * ...
        (p(j) - p(j-1));
    SlopeL(j) = (SlopeDelta) * sum(area(1:j-1));
end

SlopeL(2:L-1) = SlopeL(1) + SlopeL(2:L-1) / sum(area);

delta1(1,1) = 0;
for i = 2:L
    delta1(i,1) = delta1(i-1,1) + SlopeL(i-1) * (X(EDGEEmR+i-
1,EDGEL,SECT) - ...
    X(EDGEEmR+i-2,EDGEL,SECT));
end

%CALCULATE SLOPE OF LINE CONNECTING UPPER RIGID SURFACE FROM EDGEEmR TO
%   EDGER
SlopeT = (Y(EDGER,EDGEL,SECT) - Y(EDGEEmR,EDGEL,SECT)) / ...
    (X(EDGER,EDGEL,SECT) - X(EDGEEmR,EDGEL,SECT));

%DETERMINE THE Y COORDINATE DISTRIBUTION TO TAPER FROM EDGEEmR TO EDGER
%   OF THE RIGID SURFACE ALONG THE TOP OF THE RIGID SURFACE
%   LINEAR DISTRIBUTION IS USED TO MINIMIZE CURVATURE ON TOP OF RIGID
SURFACE
p = 0;
q = 0;
p = temp;
q = SlopeT * temp;

```

```

q = q - q(1);
q = q ./ sum(q);

TotDelta = Y(EDGEEmR, EDGEL, SECT) - Y(EDGER, EDGEL, SECT) + delta1(L);

for i = EDGEEmR+1:EDGER-1
    %area KEEPS TRACK OF AREA UNDER CURVE
    area(i-EDGEEmR) = (q(i-EDGEEmR) + 0.5 * (q(i-EDGEEmR+1) - ...
        q(i-EDGEEmR))) * (p(i-EDGEEmR+1) - p(i-EDGEEmR));
    Y(i, EDGEL, SECT) = (TotDelta) * sum(area(1:i-EDGEEmR));
end

Y(EDGEEmR+1:EDGER-1, EDGEL, SECT) = Y(EDGEEmR, EDGEL, SECT) + ...
    delta1(2:L-1) - Y(EDGEEmR+1:EDGER-1, EDGEL, SECT) / sum(area);

%APPROXIMATE THE SLOPE AT EACH POINT ALONG THE TOP OF THE RIGID BETWEEN
%   EDGEEmR AND EDGER
for i = 1:EDGER-EDGEEmR-2
    if (Y(EDGEEmR+i+1, EDGEL, SECT) - Y(EDGEEmR+i-1, EDGEL, SECT)) ~= 0
        nslopeU(i) = (X(EDGEEmR+i-1, EDGEL, SECT) - ...
            X(EDGEEmR+i+1, EDGEL, SECT)) / (Y(EDGEEmR+i+1, EDGEL, SECT) - ...
            Y(EDGEEmR+i-1, EDGEL, SECT));
    else
        nslopeU(i) = 1e6;
    end
end

%SLOPE AT EDGER-1 - AVERAGE SLOPE BETWEEN POINT BEFORE AND POINT AFTER
if nslopeU(EDGER-EDGEEmR-2) >= 0
    nslopeU(EDGER-EDGEEmR-1) = tan((atan(nslopeU(EDGER-EDGEEmR-2)) + ...
        (pi/2)) / 2);
else
    nslopeU(EDGER-EDGEEmR-1) = -tan((atan(-nslopeU(EDGER-EDGEEmR-2)) + ...
        (pi/2)) / 2);
end

%SLOPE AT EDGER
nslopeU(EDGER-EDGEEmR) = 1e6;

nslope(NUM+2:EDGER) = 1e6;

URnSlope = [nslope(1:EDGEEmR), nslopeU];

%PLACE REMAINING POINTS INSIDE RIGID SURFACE BY CONNECTING AIRFOIL
%   SURFACE TO TOP OF RIGID SURFACE WITH STRAIGHT LINES
%USE DISTRIBUTION SPECIFIED
%   BY LEFT SURFACE CORRECTED FOR BY CELL HEIGHT DICTATED BY thick1, 2,
%   and 3

delta(NUM+2:EDGER) = thick3;
S = 0;

for i = EDGEEmR+1:EDGER-1

    p = (LSS(2:EDGEL-1) - LSS(2)) / (LSS(EDGEL-1) - LSS(2)) * 4 - 2;

```

```

q = tanh(p) - 1;
q = q ./ q(1);
q(EDGE1-2) = 0;

TotDelta = sqrt( (X(i,EDGE1,SECT) - X(i,1,SECT)) * ...
    (X(i,EDGE1,SECT) - X(i,1,SECT)) + ...
    (Y(i,EDGE1,SECT) - Y(i,1,SECT)) * ...
    (Y(i,EDGE1,SECT) - Y(i,1,SECT)) );

%CREATE NEW "S" VECTOR BY SCALING LENGTH
S = TotDelta * ((X(1,1:EDGE1,SECT) - X(1,1,SECT)) / ...
    (X(1,EDGE1,SECT) - X(1,1,SECT)));

for j = 2:EDGE1-1
    deltaLEInit(j-1) = S(j+1) - S(j);
end

S(2) = S(1) + delta(i);
for j = 3:EDGE1
    S(j) = S(j-1) + deltaLEInit(j-2) + q(j-2) * (delta(i) - ...
        deltaLEInit(1));
end

%ENFORCE NORMAL AT TOP OF RIGID SURFACE
if nslopeU(i-EDGE1R) >= 0
    X(i,EDGE1-1,SECT) = X(i,EDGE1,SECT) - ...
        cos(atan(nslopeU(i-EDGE1R))) * (S(EDGE1) - S(EDGE1-1));
    Y(i,EDGE1-1,SECT) = Y(i,EDGE1,SECT) - ...
        sin(atan(nslopeU(i-EDGE1R))) * (S(EDGE1) - S(EDGE1-1));
else
    X(i,EDGE1-1,SECT) = X(i,EDGE1,SECT) + ...
        cos(atan(nslopeU(i-EDGE1R))) * (S(EDGE1) - S(EDGE1-1));
    Y(i,EDGE1-1,SECT) = Y(i,EDGE1,SECT) + ...
        sin(atan(nslopeU(i-EDGE1R))) * (S(EDGE1) - S(EDGE1-1));
end

%FIND SLOPE OF LINE CONNECTING AIRFOIL AND TOP OF RIGID SURFACE
% ASSUMING LINEAR CONNECTION
if X(i,EDGE1-1,SECT) ~= X(i,2,SECT)
    Slope = (Y(i,EDGE1-1,SECT) - Y(i,2,SECT)) / ...
        (X(i,EDGE1-1,SECT) - X(i,2,SECT));
else
    Slope = 1e6;
end

%USE HYPERBOLIC DISTRIBUTION TO SMOOTH DELTA IN NORMAL SLOPES AT
% AIRFOIL SURFACE/WAKE AND TOP OF RIGID SURFACE BETWEEN EDGE1R AND
% EDGER

%HYPERBOLIC TANGENT TAPERING OF NORMAL TO AIRFOIL/WAKE SURFACE
k = 1;
p = 0;
Snew = S / S(EDGE1);

lim = bendslopebot;

```

```

while Snew(3) > lim
    lim = lim + 0.01;
end
while Snew(k) <= lim
    p(k) = Snew(k);
    k = k + 1;
end

p = (p / Snew(k-1)) * 4 - 2;
qb = (-tanh(p) + 1);
qb = qb / qb(1);
p = ((p + 2) / 4) * Snew(k-1);
p(k:EDGEL-1) = Snew(k:EDGEL-1);
qb(k:EDGEL-1) = 0;

%HYPERBOLIC TANGENT TAPERING OF SLOPE AT EDGEL
p = 0;
k = EDGEL-1;
lim = bendslopetop;
while Snew(EDGEL-3) < 1 - lim
    lim = lim + 0.01;
end
while Snew(k) >= 1 - lim
    p(k) = Snew(k);
    k = k - 1;
end

p(k+1:EDGEL-1) = ((p(k+1:EDGEL-1) - p(k+1)) / ...
    (p(EDGEL-1) - p(k+1))) * 4 - 2;
qt(k+1:EDGEL-1) = tanh(p(k+1:EDGEL-1)) + 1;
qt(k+1:EDGEL-1) = qt(k+1:EDGEL-1) / qt(EDGEL-1);
qt(1:k) = 0;
p(k+1:EDGEL-1) = (((p(k+1:EDGEL-1) + 2) / 4) * ...
    (Snew(EDGEL-1) - Snew(k+1))) + Snew(k+1);
p(1:k) = Snew(1:k);

Theta = atan(Slope);
ThetaDiffBot = Theta - atan(nslope(i));

for j = 3:EDGEL-2
    if Theta - qb(j-1) * ThetaDiffBot >= 0
        X(i,j,SECT) = X(i,j-1,SECT) + (S(j) - S(j-1)) * ...
            cos(Theta - qb(j-1) * ThetaDiffBot);
        Y(i,j,SECT) = Y(i,j-1,SECT) + (S(j) - S(j-1)) * ...
            sin(Theta - qb(j-1) * ThetaDiffBot);
    else
        X(i,j,SECT) = X(i,j-1,SECT) - (S(j) - S(j-1)) * ...
            cos(Theta - qb(j-1) * ThetaDiffBot);
        Y(i,j,SECT) = Y(i,j-1,SECT) - (S(j) - S(j-1)) * ...
            sin(Theta - qb(j-1) * ThetaDiffBot);
    end
end

%CORRECT NEW LINE FOR TAPERING SLOPE NEAR EDGE OF RIGID SURFACE

```

```

for j = 3:EDGEL-2
    if nslopeU(i-EDGEEmR) >= 0
        X(i,j,SECT) = X(i,j,SECT) + qt(j-1) * ...
            ( ( X(i,EDGEL,SECT) - (S(EDGEL) - S(j)) * ...
                cos(atan(nslopeU(i-EDGEEmR))) ) - X(i,j,SECT) );
        Y(i,j,SECT) = Y(i,j,SECT) + qt(j-1) * ...
            ( ( Y(i,EDGEL,SECT) - (S(EDGEL) - S(j)) * ...
                sin(atan(nslopeU(i-EDGEEmR))) ) - Y(i,j,SECT) );
    else
        X(i,j,SECT) = X(i,j,SECT) + qt(j-1) * ...
            ( ( X(i,EDGEL,SECT) + (S(EDGEL) - S(j)) * ...
                cos(atan(nslopeU(i-EDGEEmR))) ) - X(i,j,SECT) );
        Y(i,j,SECT) = Y(i,j,SECT) + qt(j-1) * ...
            ( ( Y(i,EDGEL,SECT) + (S(EDGEL) - S(j)) * ...
                sin(atan(nslopeU(i-EDGEEmR))) ) - Y(i,j,SECT) );
    end
end
end

save Intermediate URnSlope -APPEND
save Results X Y -APPEND

```

D. OUTER GRID GENERATING FUNCTIONS

1. LowerBottom.m

```

function LowerBottom(Init)
%FUNCTION PLACES ACTUAL GRID POINTS ON LOWER BOUNDARY

load InitData
load Intermediate
if Init == 0
    load Results
else
    load ResultsPP
end

SECT = 2; %FOR FUTURE MODIFICATIONS (WILL ALLOW MULTIPLE SECTORS)

%PLACE POINTS ALONG BOTTOM SURFACE AT THE INTERSECTION WITH RADIAL
% LINES EMITTING FROM THE BOTTOM OF THE RIGID SURFACE
for i = 2:m(2*SECT-1)-1
    if LnSlope(i) < 0
        %ENSURE NORMAL POINTS DN
        if X(i,m(2*SECT)-EDGEL+1,SECT) >= X(i,m(2*SECT)-EDGEL+2,SECT)
            if X(i,m(2*SECT)-EDGEL+1,SECT) >= r
                X(i,1,SECT) = ((-r - Y(i,m(2*SECT)-EDGEL+1,SECT)) / ...
                    LnSlope(i)) + X(i,m(2*SECT)-EDGEL+1,SECT);
                Y(i,1,SECT) = -r;
            %ENSURE LINE STARTS INSIDE CIRCLE BEFORE FINDING INTERSECTION
        else

```

```

X(i,1,SECT) = (r + LnSlope(i) * (LnSlope(i) * ...
  X(i,m(2*SECT)-EDGEL+1,SECT) - ...
  Y(i,m(2*SECT)-EDGEL+1,SECT)) + sqrt( r * r + ...
  LnSlope(i) * ( 2 * Y(i,m(2*SECT)-EDGEL+1,SECT) * ...
  (X(i,m(2*SECT)-EDGEL+1,SECT) - r) + LnSlope(i) * ...
  X(i,m(2*SECT)-EDGEL+1,SECT) * (2 * r - ...
  X(i,m(2*SECT)-EDGEL+1,SECT)) ) - ...
  Y(i,m(2*SECT)-EDGEL+1,SECT) * ...
  Y(i,m(2*SECT)-EDGEL+1,SECT) )) / ...
  (LnSlope(i) * LnSlope(i) + 1);
if X(i,1,SECT) >= r
  X(i,1,SECT) = ((r - Y(i,m(2*SECT)-EDGEL+1,SECT)) / ...
    LnSlope(i)) + X(i,m(2*SECT)-EDGEL+1,SECT);
  Y(i,1,SECT) = -r;
else
  Y(i,1,SECT) = LnSlope(i) * (X(i,1,SECT) - ...
    X(i,m(2*SECT)-EDGEL+1,SECT)) + ...
    Y(i,m(2*SECT)-EDGEL+1,SECT);
end
end
else
  X(i,1,SECT) = (r + LnSlope(i) * (LnSlope(i) * ...
    X(i,m(2*SECT)-EDGEL+1,SECT) - ...
    Y(i,m(2*SECT)-EDGEL+1,SECT)) - sqrt( r * r + ...
    LnSlope(i) * ( 2 * Y(i,m(2*SECT)-EDGEL+1,SECT) * ...
    (X(i,m(2*SECT)-EDGEL+1,SECT) - r) + ...
    LnSlope(i) * X(i,m(2*SECT)-EDGEL+1,SECT) * (2 * r - ...
    X(i,m(2*SECT)-EDGEL+1,SECT)) ) - ...
    Y(i,m(2*SECT)-EDGEL+1,SECT) * ...
    Y(i,m(2*SECT)-EDGEL+1,SECT) )) / ...
    (LnSlope(i) * LnSlope(i) + 1);
  Y(i,1,SECT) = LnSlope(i) * (X(i,1,SECT) - ...
    X(i,m(2*SECT)-EDGEL+1,SECT)) + ...
    Y(i,m(2*SECT)-EDGEL+1,SECT);
end
else
  if X(i,m(2*SECT)-EDGEL+1,SECT) >= 2*r
    X(i,1,SECT) = ((-r - Y(i,m(2*SECT)-EDGEL+1,SECT)) / ...
      LnSlope(i)) + X(i,m(2*SECT)-EDGEL+1,SECT);
    Y(i,1,SECT) = -r;
  else
    X(i,1,SECT) = (r + LnSlope(i) * (LnSlope(i) * ...
      X(i,m(2*SECT)-EDGEL+1,SECT) - ...
      Y(i,m(2*SECT)-EDGEL+1,SECT)) - sqrt( r * r + ...
      LnSlope(i) * ( 2 * ... Y(i,m(2*SECT)-EDGEL+1,SECT) * ...
      (X(i,m(2*SECT)-EDGEL+1,SECT) - r) + ...
      LnSlope(i) * X(i,m(2*SECT)-EDGEL+1,SECT) * (2 * r - ...
      X(i,m(2*SECT)-EDGEL+1,SECT)) ) - ...
      Y(i,m(2*SECT)-EDGEL+1,SECT) * ...
      Y(i,m(2*SECT)-EDGEL+1,SECT) )) / (LnSlope(i) * ...
      LnSlope(i) + 1);
    if X(i,1,SECT) >= r
      X(i,1,SECT) = ((-r - Y(i,m(2*SECT)-EDGEL+1,SECT)) / ...
        LnSlope(i)) + X(i,m(2*SECT)-EDGEL+1,SECT);
      Y(i,1,SECT) = -r;
    end
  end
end

```

```

        else
            Y(i,1,SECT) = LnSlope(i) * (X(i,1,SECT) - ...
                X(i,m(2*SECT)-EDGEL+1,SECT)) + ...
                Y(i,m(2*SECT)-EDGEL+1,SECT);
        end
    end
end
end

%FIND BEGINNING OF OVERLAP SECTION
%LocOverlap IS RIGHT EDGE OF OVERLAP REGION
LocOverlap = 0;
%ASSUME NO OVERLAPPING IN THE FIRST FOURTH OF THE AIRFOIL
i = floor(NUM2 / 4);
while i <= m(2*SECT-1)
    if X(i,1,SECT) < X(i-1,1,SECT)
        LocOverlap = i - 1;
    end

    %FIND END OF OVERLAP SECTION
    %FirstPtOverlap IS FIRST POINT IN OVERLAPPING SECTION
    %LastPtOverlap IS LAST POINT IN OVERLAPPING SECTION
    if LocOverlap ~= i - 1;
        i = i + 1;
    else
        %FIND LAST POINT IN OVERLAPPING SECTION
        j = LocOverlap + 1;
        while j <= m(2*SECT-1) - 1 & X(j+1,1,SECT) < X(LocOverlap,1,SECT)
            j = j + 1;
        end
        LastPtOverlap = j;

        %FIND LEFTMOST POINT IN OVERLAPPING SECTION
        j = LocOverlap;
        while j <= m(2*SECT-1) - 1 & X(j+1,1,SECT) < X(j,1,SECT)
            j = j + 1;
        end
        LtEndOverlap = j;

        %FIND FIRST POINT IN OVERLAPPING SECTION
        j = LocOverlap - 1;
        while X(j,1,SECT) > X(LtEndOverlap,1,SECT)
            j = j - 1;
        end
        FirstPtOverlap = j + 1;

        %CHECK TO ENSURE POINTS REMAIN WITHIN BOUNDARY
        if X(LocOverlap,1,SECT) <= X(m(2*SECT-1),m(2*SECT),SECT)
            %REDISTRIBUTE POINTS (LINEAR DISTRIBUTION)
            int = (X(LocOverlap,1,SECT) - X(LtEndOverlap,1,SECT)) / ...
                (LastPtOverlap - FirstPtOverlap);
            X(FirstPtOverlap:LastPtOverlap,1,SECT) = ...
                [X(LtEndOverlap,1,SECT):int:X(LocOverlap,1,SECT)];
        else
            %REDISTRIBUTE POINTS (LINEAR DISTRIBUTION)

```

```

        int = (X(m(2*SECT-1),m(2*SECT),SECT) - ...
            X(LtEndOverlap,1,SECT)) / ...
            (LastPtOverlap - FirstPtOverlap);
        X(FirstPtOverlap>LastPtOverlap,1,SECT) = ...
            [X(LtEndOverlap,1,SECT):int:X(m(2*SECT-1),m(2*SECT),SECT)];
    end
    i = LastPtOverlap + 1;
end
end

if Init == 0
    save Results X Y -APPEND
else
    save ResultsPP X Y -APPEND
end

```

2. RectLowerBottom.m

```

function RectLowerBottom(Init)
%FUNCTION PLACES ACTUAL GRID POINTS ON LOWER BOUNDARY

load InitData
load Intermediate
if Init == 0
    load Results
else
    load ResultsPP
end

SECT = 2;    %FOR FUTURE MODIFICATIONS (WILL ALLOW MULTIPLE SECTORS)

%PLACE POINTS ALONG BOTTOM SURFACE AT THE INTERSECTION WITH RADIAL
% LINES EMITTING FROM THE BOTTOM OF THE RIGID SURFACE
for i = 2:m(2*SECT-1)-1
    if LnSlope(i) > 0
        Y(i,1,SECT) = -LnSlope(i) * X(i,m(2*SECT)-EDGEL+1,SECT) + ...
            Y(i,m(2*SECT)-EDGEL+1,SECT);
        X(i,1,SECT) = 0;
        if Y(i,1,SECT) < -r
            X(i,1,SECT) = X(i,m(2*SECT)-EDGEL+1,SECT) - ...
                (r + Y(i,m(2*SECT)-EDGEL+1,SECT)) / LnSlope(i);
            Y(i,1,SECT) = -r;
        end
    else
        if X(i,m(2*SECT),SECT) > X(i-1,m(2*SECT),SECT)
            X(i,1,SECT) = X(i,m(2*SECT)-EDGEL+1,SECT) - ...
                (r + Y(i,m(2*SECT)-EDGEL+1,SECT)) / LnSlope(i);
            Y(i,1,SECT) = -r;
        else
            Y(i,1,SECT) = -LnSlope(i) * X(i,m(2*SECT)-EDGEL+1,SECT) + ...
                Y(i,m(2*SECT)-EDGEL+1,SECT);
            X(i,1,SECT) = 0;
        end
    end
end
end

```

```

end

%FIND BEGINNING OF OVERLAP SECTION
%LocOverlap IS RIGHT EDGE OF OVERLAP REGION
LocOverlap = 0;
i = 2;
while i <= m(2*SECT-1)
    if X(i,1,SECT) < X(i-1,1,SECT)
        LocOverlap = i - 1;
    end

    %FIND END OF OVERLAP SECTION
    %FirstPtOverlap IS FIRST POINT IN OVERLAPPING SECTION
    %LastPtOverlap IS LAST POINT IN OVERLAPPING SECTION
    if LocOverlap ~= i - 1;
        i = i + 1;
    else
        %FIND LAST POINT IN OVERLAPPING SECTION
        j = LocOverlap + 1;
        while j <= m(2*SECT-1) - 1 & X(j+1,1,SECT) < X(LocOverlap,1,SECT)
            j = j + 1;
        end
        LastPtOverlap = j;

        %FIND LEFTMOST POINT IN OVERLAPPING SECTION
        j = LocOverlap;
        while j <= m(2*SECT-1) - 1 & X(j+1,1,SECT) < X(j,1,SECT)
            j = j + 1;
        end
        LtEndOverlap = j;

        %FIND FIRST POINT IN OVERLAPPING SECTION
        j = LocOverlap - 1;
        while X(j,1,SECT) > X(LtEndOverlap,1,SECT)
            j = j - 1;
        end
        FirstPtOverlap = j + 1;

        %CHECK TO ENSURE POINTS REMAIN WITHIN BOUNDARY
        if X(LocOverlap,1,SECT) <= X(m(2*SECT-1),m(2*SECT),SECT)
            %REDISTRIBUTE POINTS (LINEAR DISTRIBUTION)
            int = (X(LocOverlap,1,SECT) - X(LtEndOverlap,1,SECT)) / ...
                (LastPtOverlap - FirstPtOverlap);
            X(FirstPtOverlap>LastPtOverlap,1,SECT) = ...
                [X(LtEndOverlap,1,SECT):int:X(LocOverlap,1,SECT)];
        else
            %REDISTRIBUTE POINTS (LINEAR DISTRIBUTION)
            int = (X(m(2*SECT-1),m(2*SECT),SECT) - ...
                X(LtEndOverlap,1,SECT)) / (LastPtOverlap - FirstPtOverlap);
            X(FirstPtOverlap>LastPtOverlap,1,SECT) = ...
                [X(LtEndOverlap,1,SECT):int:X(m(2*SECT-1),m(2*SECT),SECT)];
        end

        i = LastPtOverlap + 1;
    end
end

```

```

end

if Init == 0
    save Results X Y -APPEND
else
    save ResultsPP X Y -APPEND
end

```

3. UpperTop.m

```

function UpperTop(Init)
%FUNCTION PLACES ACTUAL GRID POINTS ON UPPER BOUNDARY

load InitData
load Intermediate
if Init == 0
    load Results
else
    load ResultsPP
end

SECT = 1;    %FOR FUTURE MODIFICATIONS (WILL ALLOW MULTIPLE SECTORS)

%PLACE POINTS ALONG UPPER SURFACE AT INTERSECTION WITH RADIAL LINES
%   EMITTING FROM THE TOP OF THE RIGID SURFACE
for i = 2:m(2*SECT-1)-1
    if Unslope(i) > 0

        %ENSURE NORMAL POINTS UP
        if X(i,EDGEL,SECT) >= X(i,EDGEL-1,SECT)
            if X(i,EDGEL,SECT) >= r
                X(i,m(2*SECT),SECT) = ((r - Y(i,EDGEL,SECT)) / ...
                    Unslope(i)) + X(i,EDGEL,SECT);
                Y(i,m(2*SECT),SECT) = r;
            %ENSURE LINE STARTS INSIDE CIRCLE BEFORE FINDING INTERSECTION
            else
                X(i,m(2*SECT),SECT) = (r + Unslope(i) * ...
                    (Unslope(i) * X(i,EDGEL,SECT) - ...
                    Y(i,EDGEL,SECT)) + sqrt( r * r + Unslope(i) * ...
                    ( 2 * Y(i,EDGEL,SECT) * (X(i,EDGEL,SECT) - r) + ...
                    Unslope(i) * X(i,EDGEL,SECT) * (2 * r - ...
                    X(i,EDGEL,SECT)) ) - Y(i,EDGEL,SECT) * ...
                    Y(i,EDGEL,SECT) )) / (Unslope(i) * Unslope(i) + 1);
            if X(i,m(2*SECT),SECT) >= r
                X(i,m(2*SECT),SECT) = ((r - Y(i,EDGEL,SECT)) / ...
                    Unslope(i)) + X(i,EDGEL,SECT);
                Y(i,m(2*SECT),SECT) = r;
            else
                Y(i,m(2*SECT),SECT) = Unslope(i) * ...
                    (X(i,m(2*SECT),SECT) - ...
                    X(i,EDGEL,SECT)) + Y(i,EDGEL,SECT);
            end
        end
    end
else

```

```

X(i,m(2*SECT),SECT) = (r + UnSlope(i) * (UnSlope(i) * ...
    X(i,EDGEL,SECT) - Y(i,EDGEL,SECT)) - sqrt( r * r + ...
    UnSlope(i) * ( 2 * Y(i,EDGEL,SECT) * ...
    (X(i,EDGEL,SECT) - r) + UnSlope(i) * X(i,EDGEL,SECT) * ...
    (2 * r - X(i,EDGEL,SECT)) ) - Y(i,EDGEL,SECT) * ...
    Y(i,EDGEL,SECT) )) / (UnSlope(i) * UnSlope(i) + 1);
Y(i,m(2*SECT),SECT) = UnSlope(i) * (X(i,m(2*SECT),SECT) - ...
    X(i,EDGEL,SECT)) + Y(i,EDGEL,SECT);
end
else

X(i,m(2*SECT),SECT) = (r + UnSlope(i) * (UnSlope(i) * ...
    X(i,EDGEL,SECT) - Y(i,EDGEL,SECT)) - ...
    sqrt( r * r + UnSlope(i) * ( 2 * Y(i,EDGEL,SECT) * ...
    (X(i,EDGEL,SECT) - r) + UnSlope(i) * X(i,EDGEL,SECT) * ...
    (2 * r - X(i,EDGEL,SECT)) ) - Y(i,EDGEL,SECT) * ...
    Y(i,EDGEL,SECT) )) / (UnSlope(i) * UnSlope(i) + 1);
if X(i,m(2*SECT),SECT) >= r
    X(i,m(2*SECT),SECT) = ((r - Y(i,EDGEL,SECT)) / ...
        UnSlope(i)) + X(i,EDGEL,SECT);
    Y(i,m(2*SECT),SECT) = r;
else
    Y(i,m(2*SECT),SECT) = UnSlope(i) * (X(i,m(2*SECT),SECT) - ...
        X(i,EDGEL,SECT)) + Y(i,EDGEL,SECT);
end
end
end

%FIND BEGINNING OF OVERLAP SECTION
%LocOverlap IS RIGHT EDGE OF OVERLAP REGION
LocOverlap = 0;
%ASSUME NO OVERLAPPING IN THE FIRST FOURTH OF THE AIRFOIL
i = floor(NUM / 4);
while i <= m(2*SECT-1)
    if X(i,m(2*SECT),SECT) < X(i-1,m(2*SECT),SECT)
        LocOverlap = i - 1;
    end

    %FIND END OF OVERLAP SECTION
    %FirstPtOverlap IS FIRST POINT IN OVERLAPPING SECTION
    %LastPtOverlap IS LAST POINT IN OVERLAPPING SECTION
    if LocOverlap ~= i - 1
        i = i + 1;
    else
        %FIND LAST POINT IN OVERLAPPING SECTION
        j = LocOverlap + 1;
        while j <= m(2*SECT-1) - 1 ...
            & X(j+1,m(2*SECT),SECT) < X(LocOverlap,m(2*SECT),SECT)
            j = j + 1;
        end
        LastPtOverlap = j;

        %FIND LEFTMOST POINT IN OVERLAPPING SECTION
        j = LocOverlap;
        while j <= m(2*SECT-1) - 1 ...

```

```

        & X(j+1,m(2*SECT),SECT) < X(j,m(2*SECT),SECT)
        j = j + 1;
    end
    LtEndOverlap = j;

    %FIND FIRST POINT IN OVERLAPPING SECTION
    j = LocOverlap - 1;
    while X(j,m(2*SECT),SECT) > X(LtEndOverlap,m(2*SECT),SECT)
        j = j - 1;
    end
    FirstPtOverlap = j + 1;

    %CHECK TO ENSURE POINTS REMAIN WITHIN BOUNDARY
    if X(LocOverlap,m(2*SECT),SECT) <= X(m(2*SECT-1),1,SECT)
        %REDISTRIBUTE POINTS (LINEAR DISTRIBUTION)
        int = (X(LocOverlap,m(2*SECT),SECT) - ...
            X(LtEndOverlap,m(2*SECT),SECT)) / ...
            (LastPtOverlap - FirstPtOverlap);
        X(FirstPtOverlap:LastPtOverlap,m(2*SECT),SECT) = ...
        [X(LtEndOverlap,m(2*SECT),SECT):int:X(LocOverlap,m(2*SECT),SECT)];
    else
        %REDISTRIBUTE POINTS (LINEAR DISTRIBUTION)
        int = (X(m(2*SECT-1),1,SECT) - ...
            X(LtEndOverlap,m(2*SECT),SECT)) / ...
            (LastPtOverlap - FirstPtOverlap);
        X(FirstPtOverlap:LastPtOverlap,m(2*SECT),SECT) = ...
        [X(LtEndOverlap,m(2*SECT),SECT):int:X(m(2*SECT-1),1,SECT)];
    end
    i = LastPtOverlap + 1;
end
end
if Init == 0
    save Results X Y -APPEND
else
    save ResultsPP X Y -APPEND
end
end

```

4. RectUpperTop.m

```

function RectUpperTop(Init)
%FUNCTION PLACES ACTUAL GRID POINTS ON UPPER BOUNDARY

load InitData
load Intermediate
if Init == 0
    load Results
else
    load ResultsPP
end

SECT = 1;    %FOR FUTURE MODIFICATIONS (WILL ALLOW MULTIPLE SECTORS)

%PLACE POINTS ALONG UPPER SURFACE AT INTERSECTION WITH RADIAL LINES
%    EMITTING FROM THE TOP OF THE RIGID SURFACE

```

```

for i = 2:m(2*SECT)-1
    if Unslope(i) < 0
        Y(i,m(2*SECT),SECT) = -Unslope(i) * X(i,EDGEL,SECT) + ...
            Y(i,EDGEL,SECT);
        X(i,m(2*SECT),SECT) = 0;
        if Y(i,m(2*SECT),SECT) > r
            Y(i,m(2*SECT),SECT) = r;
            X(i,m(2*SECT),SECT) = X(i,EDGEL,SECT) + ...
                (r - Y(i,EDGEL,SECT)) / Unslope(i);
        end
    else
        if X(i,1,SECT) > X(i-1,1,SECT)
            Y(i,m(2*SECT),SECT) = r;
            X(i,m(2*SECT),SECT) = X(i,EDGEL,SECT) + ...
                (r - Y(i,EDGEL,SECT)) / Unslope(i);
        else
            Y(i,m(2*SECT),SECT) = -Unslope(i) * X(i,EDGEL,SECT) + ...
                Y(i,EDGEL,SECT);
            X(i,m(2*SECT),SECT) = 0;
        end
    end
end

%FIND BEGINNING OF OVERLAP SECTION
%LocOverlap IS RIGHT EDGE OF OVERLAP REGION
LocOverlap = 0;
i = 2;
while i <= m(2*SECT)-1
    if X(i,m(2*SECT),SECT) < X(i-1,m(2*SECT),SECT)
        LocOverlap = i - 1;
    end

    %FIND END OF OVERLAP SECTION
    %LtEndOverlap IS FIRST OVERLAPPING POINT
    %RtEndOverlap IS LAST OVERLAPPING POINT
    if LocOverlap ~= i - 1
        i = i + 1;
    else
        %FIND LAST POINT IN OVERLAPPING SECTION
        j = LocOverlap + 1;
        while j <= m(2*SECT)-1 ...
            & X(j+1,m(2*SECT),SECT) < X(LocOverlap,m(2*SECT),SECT)
            j = j + 1;
        end
        LastPtOverlap = j;

        %FIND LEFTMOST POINT IN OVERLAPPING SECTION
        j = LocOverlap;
        while j <= m(2*SECT)-1 ...
            & X(j+1,m(2*SECT),SECT) < X(j,m(2*SECT),SECT)
            j = j + 1;
        end
        LtEndOverlap = j;

        %FIND FIRST POINT IN OVERLAPPING SECTION

```

```

j = LocOverlap - 1;
while X(j,m(2*SECT),SECT) > X(LtEndOverlap,m(2*SECT),SECT)
    j = j - 1;
end
FirstPtOverlap = j + 1;

%CHECK TO ENSURE POINTS REMAIN WITHIN BOUNDARY
if X(LocOverlap,m(2*SECT),SECT) <= X(m(2*SECT-1),1,SECT)
    %REDISTRIBUTE POINTS (LINEAR DISTRIBUTION)
    int = (X(LocOverlap,m(2*SECT),SECT) - ...
        X(LtEndOverlap,m(2*SECT),SECT)) / ...
        (LastPtOverlap - FirstPtOverlap);
    X(FirstPtOverlap>LastPtOverlap,m(2*SECT),SECT) = ...
[X(LtEndOverlap,m(2*SECT),SECT):int:X(LocOverlap,m(2*SECT),SECT)];
else
    %REDISTRIBUTE POINTS (LINEAR DISTRIBUTION)
    int = (X(m(2*SECT-1),1,SECT) - ...
        X(LtEndOverlap,m(2*SECT),SECT)) / ...
        (LastPtOverlap - FirstPtOverlap);
    X(FirstPtOverlap>LastPtOverlap,m(2*SECT),SECT) = ...
[X(LtEndOverlap,m(2*SECT),SECT):int:X(m(2*SECT-1),1,SECT)];
end

i = LastPtOverlap + 1;
end
end

if Init == 0
    save Results X Y -APPEND
else
    save ResultsPP X Y -APPEND
end

```

5. LowerHorizontal.m

```

function LowerHorizontal(Init)

load InitData
load Intermediate

if Init == 0
    load Results
else
    load ResultsPP
end

SECT = 2;    %FOR FUTURE MODIFICATIONS (WILL ALLOW MULTIPLE SECTORS)

%SPREAD THE DELTA BETWEEN BOTTOM AND LOWER RIGID SURFACES LINEARLY
%    LINEARLY VARY THE GRID HEIGHT DELTA BETWEEN THE LEFT AND RIGHT
%    SURFACES
S1 = ArcLength(X(1,m(2*SECT)-EDGEL+1:-1:1,SECT),...
    Y(1,m(2*SECT)-EDGEL+1:-1:1,SECT),m(2*SECT)-EDGEL+1);
S1 = S1 ./ S1(m(2*SECT)-EDGEL+1);

```

```

S2 = ArcLength(X(m(2*SECT-1),m(2*SECT)-EDGEL+1:-1:1,SECT),...
Y(m(2*SECT-1),m(2*SECT)-EDGEL+1:-1:1,SECT),m(2*SECT)-EDGEL+1);
S2 = S2 ./ S2(m(2*SECT)-EDGEL+1);
for j = 1:m(2*SECT)-EDGEL-1
    S3(:,j) = ArcLength(X(:,m(2*SECT)-EDGEL+1-j,SECT),...
Y(:,m(2*SECT)-EDGEL+1-j,SECT),m(2*SECT-1))';
    S3(:,j) = S3(:,j) ./ S3(m(2*SECT-1),j);
end

for i = 1:m(2*SECT-1)-2

    xDelta(i) = X(i+1,1,SECT) - X(i+1,m(2*SECT)-EDGEL+1,SECT);
    yDelta(i) = Y(i+1,1,SECT) - Y(i+1,m(2*SECT)-EDGEL+1,SECT);

    for j = 1:m(2*SECT)-EDGEL-1

        X(i+1,m(2*SECT)-EDGEL+1-j,SECT) = xDelta(i) * (S3(i,j) * ...
S1(j+1) + ...
(1 - S3(i,j)) * S2(j+1)) + X(i+1,m(2*SECT)-EDGEL+1,SECT);
        Y(i+1,m(2*SECT)-EDGEL+1-j,SECT) = yDelta(i) * (S3(i,j) * ...
S1(j+1) + ...
(1 - S3(i,j)) * S2(j+1)) + Y(i+1,m(2*SECT)-EDGEL+1,SECT);
    end

end

%MATCH CELL HEIGHT BETWEEN RIGID SURFACE AND OUTER GRIDLINES AND MATCH
SLOPE AT EDGE OF RIGID SURFACE
for i = 1:m(2*SECT-1)-2

    for j = m(2*SECT)-EDGEL+1:-1:1
        S(m(2*SECT)-EDGEL-j+2) = sqrt( (X(i+1,j,SECT) - ...
X(i+1,m(2*SECT)-EDGEL+1,SECT)) * (X(i+1,j,SECT) - ...
X(i+1,m(2*SECT)-EDGEL+1,SECT)) + ...
(Y(i+1,j,SECT) - Y(i+1,m(2*SECT)-EDGEL+1,SECT)) * ...
(Y(i+1,j,SECT) - Y(i+1,m(2*SECT)-EDGEL+1,SECT)) );
    end
    Snew = S ./ S(length(S));

    %HYPERBOLIC DISTRIBUTION TO MATCH THE CELL HEIGHT BETWEEN RIGID
    % SURFACE AND OUTER BOUNDARY
    p = 0;
    p = S ./ S(m(2*SECT)-EDGEL+1) * 4 - 2;
    q = tanh(p) + 1;
    q = q ./ q(m(2*SECT)-EDGEL+1);

    delta = sqrt( (X(i+1,m(2*SECT)-EDGEL+2,SECT) - ...
X(i+1,m(2*SECT)-EDGEL+1,SECT)) * ...
(X(i+1,m(2*SECT)-EDGEL+2,SECT) - ...
X(i+1,m(2*SECT)-EDGEL+1,SECT)) + ...
(Y(i+1,m(2*SECT)-EDGEL+2,SECT) - ...
Y(i+1,m(2*SECT)-EDGEL+1,SECT)) * ...
(Y(i+1,m(2*SECT)-EDGEL+2,SECT) - ...
Y(i+1,m(2*SECT)-EDGEL+1,SECT)) ) - ...
sqrt( (X(i+1,m(2*SECT)-EDGEL+1,SECT) - ...

```

```

X(i+1,m(2*SECT)-EDGEL,SECT)) * ...
(X(i+1,m(2*SECT)-EDGEL+1,SECT) - ...
X(i+1,m(2*SECT)-EDGEL,SECT)) + ...
(Y(i+1,m(2*SECT)-EDGEL+1,SECT) - ...
Y(i+1,m(2*SECT)-EDGEL,SECT)) * ...
(Y(i+1,m(2*SECT)-EDGEL+1,SECT) - ...
Y(i+1,m(2*SECT)-EDGEL,SECT)) );

if X(i+1,1,SECT) ~= X(i+1,m(2*SECT)-EDGEL+1,SECT)
    Slope = (Y(i+1,1,SECT) - Y(i+1,m(2*SECT)-EDGEL+1,SECT)) / ...
            (X(i+1,1,SECT) - X(i+1,m(2*SECT)-EDGEL+1,SECT));
else
    Slope = 1e6;
end

%HYPERBOLIC TANGENT TAPERING OF SLOPE AT EDGEL
k = 1;
lim = obendslopebot;
while Snew(3) > lim
    lim = lim + 0.01;
end
while Snew(k) <= lim
    p(k) = Snew(k);
    k = k + 1;
end

p = (p / Snew(k-1)) * 4 - 2;
qb = (-tanh(p) + 1);
qb = qb / qb(1);
p = ((p + 2) / 4) * Snew(k-1);
p(k:m(2*SECT)-EDGEL+1) = Snew(k:m(2*SECT)-EDGEL+1);
qb(k:m(2*SECT)-EDGEL+1) = 0;

for j = m(2*SECT)-EDGEL:-1:2

    %MATCH CELL HEIGHT BETWEEN RIGID SURFACE AND OUTER BOUNDARY
    if Slope >= 0
        X(i+1,j,SECT) = X(i+1,j,SECT) - q(m(2*SECT)-EDGEL-j+2) * ...
            cos(atan(Slope)) * delta;
        Y(i+1,j,SECT) = Y(i+1,j,SECT) - q(m(2*SECT)-EDGEL-j+2) * ...
            sin(atan(Slope)) * delta;
    else
        X(i+1,j,SECT) = X(i+1,j,SECT) + q(m(2*SECT)-EDGEL-j+2) * ...
            cos(atan(Slope)) * delta;
        Y(i+1,j,SECT) = Y(i+1,j,SECT) + q(m(2*SECT)-EDGEL-j+2) * ...
            sin(atan(Slope)) * delta;
    end

end

end

if Init == 0
    save Results X Y -APPEND
else
    save ResultsPP X Y -APPEND
end

```

end

6. UpperHorizontal.m

```
function UpperHorizontal(Init)

load InitData
load Intermediate

if Init == 0
    load Results
else
    load ResultsPP
end

SECT = 1;    %FOR FUTURE MODIFICATIONS (WILL ALLOW MULTIPLE SECTORS)

%SPREAD THE DELTA BETWEEN TOP AND UPPER RIGID SURFACES LINEARLY
%    LINEARLY VARY THE GRID HEIGHT DELTA BETWEEN THE LEFT AND RIGHT
%    SURFACES
S1 = ArcLength(X(1,EDGEL:m(2*SECT),SECT),Y(1,EDGEL:m(2*SECT),SECT), ...
    m(2*SECT)-EDGEL+1);
S1 = S1 ./ S1(m(2*SECT)-EDGEL+1);
S2 = ArcLength(X(m(2*SECT-1),EDGEL:m(2*SECT),SECT), ...
    Y(m(2*SECT-1),EDGEL:m(2*SECT),SECT),m(2*SECT)-EDGEL+1);
S2 = S2 ./ S2(m(2*SECT)-EDGEL+1);

S3 = zeros(m(2*SECT-1),m(2*SECT)-EDGEL-1);
for j = 1:m(2*SECT)-EDGEL-1
    S3(:,j) = ArcLength(X(:,j+EDGEL,SECT),Y(:,j+EDGEL,SECT), ...
        m(2*SECT-1))';
    S3(:,j) = S3(:,j) ./ S3(m(2*SECT-1),j);
end

for i = 1:m(2*SECT-1)-2

    xDelta(i) = X(i+1,m(2*SECT),SECT) - X(i+1,EDGEL,SECT);
    yDelta(i) = Y(i+1,m(2*SECT),SECT) - Y(i+1,EDGEL,SECT);

    for j = 1:m(2*SECT)-EDGEL-1

        X(i+1,j+EDGEL,SECT) = xDelta(i) * (S3(i,j) * S1(j+1) + ...
            (1 - S3(i,j)) * S2(j+1)) + X(i+1,EDGEL,SECT);
        Y(i+1,j+EDGEL,SECT) = yDelta(i) * (S3(i,j) * S1(j+1) + ...
            (1 - S3(i,j)) * S2(j+1)) + Y(i+1,EDGEL,SECT);
    end
end

%MATCH CELL HEIGHT BETWEEN RIGID SURFACE AND OUTER GRIDLINES AND
%    MATCH SLOPE AT EDGE OF RIGID SURFACE
for i = 1:m(2*SECT-1)-2

    for j = EDGEL:m(2*SECT)
```

```

    S(j-EDGEL+1) = sqrt( (X(i+1,j,SECT) - X(i+1,EDGEL,SECT)) * ...
        (X(i+1,j,SECT) - X(i+1,EDGEL,SECT)) + (Y(i+1,j,SECT) - ...
        Y(i+1,EDGEL,SECT)) * (Y(i+1,j,SECT) - Y(i+1,EDGEL,SECT)) );
end
Snew = S ./ S(length(S));

%HYPERBOLIC DISTRIBUTION TO MATCH THE CELL HEIGHT BETWEEN RIGID
% SURFACE AND OUTER BOUNDARY
p = 0;
p = S ./ S(m(2*SECT)-EDGEL+1) * 4 - 2;
q = tanh(p) + 1;
q = q ./ q(m(2*SECT)-EDGEL+1);

delta = sqrt( (X(i+1,EDGEL-1,SECT) - X(i+1,EDGEL,SECT)) * ...
    (X(i+1,EDGEL-1,SECT) - X(i+1,EDGEL,SECT)) + ...
    (Y(i+1,EDGEL-1,SECT) - Y(i+1,EDGEL,SECT)) * ...
    (Y(i+1,EDGEL-1,SECT) - Y(i+1,EDGEL,SECT)) ) - ...
    sqrt( (X(i+1,EDGEL,SECT) - X(i+1,EDGEL+1,SECT)) * ...
    (X(i+1,EDGEL,SECT) - X(i+1,EDGEL+1,SECT)) + ...
    (Y(i+1,EDGEL,SECT) - Y(i+1,EDGEL+1,SECT)) * ...
    (Y(i+1,EDGEL,SECT) - Y(i+1,EDGEL+1,SECT)) );

if X(i+1,m(2*SECT),SECT) ~= X(i+1,EDGEL,SECT)
    Slope = (Y(i+1,m(2*SECT),SECT) - Y(i+1,EDGEL,SECT)) / ...
        (X(i+1,m(2*SECT),SECT) - X(i+1,EDGEL,SECT));
else
    Slope = 1e6;
end

%HYPERBOLIC TANGENT TAPERING OF SLOPE AT EDGEL
k = 1;
lim = obendslopebot;
while Snew(3) > lim
    lim = lim + 0.01;
end
while Snew(k) <= lim
    p(k) = Snew(k);
    k = k + 1;
end

p = (p / Snew(k-1)) * 4 - 2;
qb = (-tanh(p) + 1);
qb = qb / qb(1);
p = ((p + 2) / 4) * Snew(k-1);
p(k:m(2*SECT)-EDGEL+1) = Snew(k:m(2*SECT)-EDGEL+1);
qb(k:m(2*SECT)-EDGEL+1) = 0;

for j = EDGEL+1:m(2*SECT)-1

    %MATCH CELL HEIGHT BETWEEN RIGID SURFACE AND OUTER BOUNDARY
    if Slope >= 0
        X(i+1,j,SECT) = X(i+1,j,SECT) + q(j-EDGEL+1) * ...
            cos(atan(Slope)) * delta;
        Y(i+1,j,SECT) = Y(i+1,j,SECT) + q(j-EDGEL+1) * ...
            sin(atan(Slope)) * delta;
    end
end

```

```

else
    X(i+1,j,SECT) = X(i+1,j,SECT) - q(j-EDGEL+1) * ...
        cos(atan(Slope)) * delta;
    Y(i+1,j,SECT) = Y(i+1,j,SECT) - q(j-EDGEL+1) * ...
        sin(atan(Slope)) * delta;
end

end

end

if Init == 0
    save Results X Y -APPEND
else
    save ResultsPP X Y -APPEND
end

```

7. WakeLower.m

```

function WakeLower(Init)
%FUNCTION PLACES POINTS AROUND AIRFOIL WAKE TO EDGEL IN LOWER SECTOR

load InitData
load Intermediate
if Init == 0
    load Results
    aoa = 0;
else
    load ResultsPP
end

SECT = 2;    %FOR FUTURE MODIFICATIONS (WILL ALLOW MULTIPLE SECTORS)

for j = m(2*SECT)-1:-1:m(2*SECT)-EDGEL+1

    %DETERMINE GRID HEIGHT FOR EACH ROW
    %   LINEARLY VARY FROM RIGHT EDGE OF RIGID SURFACE CELL HEIGHT TO
    %   THE RIGHT BOUNDARY CELL HEIGHT
    %   DO NOT USE ARC LENGTH TO INCREASE EFFICIENCY
    rtDelta = Y(m(2*SECT-1),j+1,SECT) - Y(m(2*SECT-1),j,SECT);
    ltDelta = Y(EDGER,j+1,SECT) - Y(EDGER,j,SECT);

    htSlope = (rtDelta - ltDelta) / (X(m(2*SECT-1),j+1,SECT) - ...
        X(EDGER,j+1,SECT));

    for i = EDGER+1:m(2*SECT-1)-1
        ht = (htSlope * (X(i,j+1,SECT) - X(EDGER,j+1,SECT)) + ltDelta);
        if UWnSlope(i-EDGER) >= 0
            angle = atan(UWnSlope(i-EDGER));
            X(i,j,SECT) = X(i,j+1,SECT) - ht * cos(angle);
            Y(i,j,SECT) = Y(i,j+1,SECT) - ht * sin(angle);
        else
            angle = atan(abs(UWnSlope(i-EDGER)));
            X(i,j,SECT) = X(i,j+1,SECT) + ht * cos(angle);
            Y(i,j,SECT) = Y(i,j+1,SECT) - ht * sin(angle);
        end
    end
end

```

```

        end
    end

end %END for j

for i = 1:EDGER
    if LRnSlope(i) > 0
        LAngle(i) = pi + atan(LRnSlope(i)) - aoa;
        LnSlope(i) = tan(LAngle(i));
    elseif LRnSlope(i) < 0
        LAngle(i) = atan(LRnSlope(i)) - aoa;
        LnSlope(i) = tan(LAngle(i));
    elseif i == EDGER
        LAngle(i) = -aoa;
        LnSlope(i) = tan(LAngle(i));
    else
        LAngle(i) = pi - aoa;
        LnSlope(i) = tan(LAngle(i));
    end
end

for i = EDGER+1:m(2*SECT-1)-1
    if UWnSlope(i-EDGER) >= 0
        LAngle(i) = pi + atan(UWnSlope(i-EDGER));
        LnSlope(i) = tan(LAngle(i));
    else
        LAngle(i) = 2 * pi + atan(UWnSlope(i-EDGER));
        LnSlope(i) = tan(LAngle(i));
    end
end

LAngle(m(2*SECT-1)) = atan(-1e6);
LnSlope(m(2*SECT-1)) = -1e6;

save Intermediate LnSlope LAngle -APPEND

if Init == 0
    save Results X Y -APPEND
else
    save ResultsPP X Y -APPEND
end

```

8. WakeUpper.m

```

function WakeUpper(Init)
%FUNCTION PLACES POINTS AROUND AIRFOIL WAKE TO EDGEL IN UPPER SECTOR

load InitData
load Intermediate
if Init == 0
    load Results
    aoa = 0;
else
    load ResultsPP

```

```

end

SECT = 1;    %FOR FUTURE MODIFICATIONS (WILL ALLOW MULTIPLE SECTORS)

%DETERMINE NORMAL SLOPE OF AIRFOIL WAKE FROM EDGER TO RIGHT BOUNDARY
for i = EDGER+1:m(2*SECT-1)-1
    if (Y(i-1,1,SECT) - Y(i+1,1,SECT)) ~= 0
        UWnSlope(i-EDGER) = (X(i+1,1,SECT) - X(i-1,1,SECT)) / ...
            (Y(i-1,1,SECT) - Y(i+1,1,SECT));
    else
        UWnSlope(i-EDGER) = 1e6;
    end
end

for j = 2:EDGEL

    %DETERMINE GRID HEIGHT FOR EACH ROW
    %   LINEARLY VARY FROM RIGHT EDGE OF RIGID SURFACE CELL HEIGHT TO
    %   THE RIGHT BOUNDARY CELL HEIGHT
    %   DO NOT USE ARC LENGTH TO INCREASE EFFICIENCY
    rtDelta = Y(m(2*SECT-1),j,SECT) - Y(m(2*SECT-1),j-1,SECT);
    ltDelta = Y(EDGER,j,SECT) - Y(EDGER,j-1,SECT);
    htSlope = (rtDelta - ltDelta) / (X(m(2*SECT-1),j-1,SECT) - ...
        X(EDGER,j-1,SECT));

    for i = EDGER+1:m(2*SECT-1)-1
        ht = (htSlope * (X(i,j-1,SECT) - X(EDGER,j-1,SECT)) + ltDelta);
        if UWnSlope(i-EDGER) >= 0
            angle = atan(UWnSlope(i-EDGER));
            X(i,j,SECT) = X(i,j-1,SECT) + ht * cos(angle);
            Y(i,j,SECT) = Y(i,j-1,SECT) + ht * sin(angle);
        else
            angle = atan(abs(UWnSlope(i-EDGER)));
            X(i,j,SECT) = X(i,j-1,SECT) - ht * cos(angle);
            Y(i,j,SECT) = Y(i,j-1,SECT) + ht * sin(angle);
        end
    end

end %END for j

for i = 1:EDGER
    if URnSlope(i) >= 0 & i ~= 1
        UAngle(i) = atan(URnSlope(i)) - aoa;
        UnSlope(i) = tan(UAngle(i));
    else
        UAngle(i) = pi + atan(URnSlope(i)) - aoa;
        UnSlope(i) = tan(UAngle(i));
    end
end

for i = EDGER+1:m(2*SECT-1)-1
    if UWnSlope(i-EDGER) >= 0
        UAngle(i) = atan(UWnSlope(i-EDGER));
        UnSlope(i) = tan(UAngle(i));
    end
end

```

```

    else
        UAngle(i) = pi + atan(UWnSlope(i-EDGER));
        UnSlope(i) = tan(UAngle(i));
    end
end

UAngle(m(2*SECT-1)) = atan(1e6);
UnSlope(m(2*SECT-1)) = 1e6;

save Intermediate UWnSlope UnSlope UAngle -APPEND

if Init == 0
    save Results X Y -APPEND
else
    save ResultsPP X Y -APPEND
end

```

E. PITCH AND PLUNGE MOTION FUNCTION

1. PitchPlungeRigid.m

```

function PitchPlungeRigid

load InitData
load Intermediate
load Results

%Rotate Upper Rigid Surface
for i = 1:EDGEL
    temp = RotateClockwise(boa, X(1:EDGER,i,1), Y(1:EDGER,i,1), ...
        xo, yo, EDGER);
    X(1:EDGER,i,1) = temp(1,:);
    Y(1:EDGER,i,1) = temp(2,:);
end

SECT = 2;    %FOR FUTURE MODIFICATIONS - WILL ALLOW MULTIPLE SECTORS

%Rotate Lower Rigid Surface
for i = m(2*SECT):-1:m(2*SECT)-EDGEL+1
    temp = RotateClockwise(boa, X(1:EDGER,i,SECT), ...
        Y(1:EDGER,i,SECT), xo, yo, EDGER);
    X(1:EDGER,i,2) = temp(1,:);
    Y(1:EDGER,i,2) = temp(2,:);
end

%Plunge Upper Rigid Surface
Y(1:EDGER,1:EDGEL,1) = Y(1:EDGER,1:EDGEL,1) + plunge;

%Plunge Lower Rigid Surface
Y(1:EDGER,m(2*SECT):-1:m(2*SECT)-EDGEL+1,2) = ...
    Y(1:EDGER,m(2*SECT):-1:m(2*SECT)-EDGEL+1,2) + plunge;

```

save ResultsPP X Y

F. GRAPHICAL USER INTERFACE CALLBACK FILES

1. AdjustMotion.m

```
function AdjustMotion

load InitData

AdjMot

a = gcf;

set(findobj(a, 'Tag', 'AOA'), 'String', -aoa * 180 / pi);
set(findobj(a, 'Tag', 'xo'), 'String', xo);
set(findobj(a, 'Tag', 'yo'), 'String', yo);
set(findobj(a, 'Tag', 'PLUNGE'), 'String', plunge);
set(findobj(a, 'Tag', 'RtBPitch'), 'String', RBPitch);
set(findobj(a, 'Tag', 'LtBPitch'), 'String', LBPitch);
set(findobj(a, 'Tag', 'RtBPlunge'), 'String', RBPlunge);
set(findobj(a, 'Tag', 'LtBPlunge'), 'String', LBPlunge);
```

2. AdjustOuterGrid.m

```
function AdjustOuterGrid

load InitData

AdjOutGrd

a = gcf;

set(findobj(a, 'Tag', 'OBendSlopeBot'), 'String', obendslopebot);
```

3. AdjustRigidData.m

```
function AdjustRigidData

load InitData

SRigid = str2num(get(findobj(gcf, 'Tag', 'SizeRigid'), 'String'));
sizesmooth = str2num(get(findobj(gcf, 'Tag', 'SizeSmooth'), 'String'));
sizesmoothL = str2num(get(findobj(gcf, 'Tag', 'SizeSmoothL'), 'String'));
bendslopebot=str2num(get(findobj(gcf, 'Tag', 'BendSlopeBot'), 'String'));
bendslopetop=str2num(get(findobj(gcf, 'Tag', 'BendSlopeTop'), 'String'));
if AirfoilType ~= 1
    xTE = str2num(get(findobj(gcf, 'Tag', 'Xte'), 'String'));
    xTESelect = get(findobj(gcf, 'Tag', 'xTESelect'), 'Value');
    save InitData SRigid sizesmooth sizesmoothL bendslopebot ...
        bendslopetop xTE xTESelect -APPEND
```

```

else
    save InitData SRigid sizesmooth sizesmoothL bendslopebot ...
        bendslopetop -APPEND
end

close

```

4. GridMotion.m

```

function GridMotion

load InitData

WakeUpper(0)
WakeLower(0)

if Type == 1
    UpperTop(0)
    ArcLengthTop
    LowerBottom(0)
    ArcLengthBottom
else
    RectUpperTop(0)
    RectArcLengthTop
    RectLowerBottom(0)
    RectArcLengthBottom
end

PitchPlungeRigid
AirfoilWake
UpperRightVertical

if Type == 1
    LeftSurface
else
    RectLeftSurface
end

WakeUpper(1)

if Type == 1
    UpperTop(1)
else
    RectUpperTop(1)
end

UpperHorizontal(1)

Duplicate(1)
LowerRightVertical
WakeLower(1)

if Type == 1
    LowerBottom(1)

```

```

else
    RectLowerBottom(1)
end

```

```

LowerHorizontal(1)

```

```

close
PlotEntire(1)
AdjustMotion

```

5. GridMotion2.m

```

function GridMotion2

```

```

load InitData

```

```

aoa = str2num(get(findobj(gcf,'Tag','AOA'),'String'));
aoa = aoa * pi / 180;
xo = str2num(get(findobj(gcf,'Tag','xo'),'String'));
yo = str2num(get(findobj(gcf,'Tag','yo'),'String'));
plunge = str2num(get(findobj(gcf,'Tag','PLUNGE'),'String'));
RBPitch = str2num(get(findobj(gcf,'Tag','RtBPitch'),'String'));
LBPitch = str2num(get(findobj(gcf,'Tag','LtBPitch'),'String'));
RBPlunge = str2num(get(findobj(gcf,'Tag','RtBPlunge'),'String'));
LBPlunge = str2num(get(findobj(gcf,'Tag','LtBPlunge'),'String'));

```

```

save InitData aoa xo yo plunge RBPitch LBPitch RBPlunge ...
    LBPlunge -APPEND

```

```

WakeUpper(0)
WakeLower(0)

```

```

if Type == 1
    UpperTop(0)
    ArcLengthTop
    LowerBottom(0)
    ArcLengthBottom
else
    RectUpperTop(0)
    RectArcLengthTop
    RectLowerBottom(0)
    RectArcLengthBottom
end

```

```

PitchPlungeRigid
AirfoilWake
UpperRightVertical

```

```

if Type == 1
    LeftSurface
else
    RectLeftSurface
end

```

```

WakeUpper(1)

```

```

if Type == 1
    UpperTop(1)
else
    RectUpperTop(1)
end

UpperHorizontal(1)

Duplicate(1)
LowerRightVertical
WakeLower(1)

if Type == 1
    LowerBottom(1)
else
    RectLowerBottom(1)
end

LowerHorizontal(1)

close
PlotEntire(1)
AdjustMotion

```

6. InitialData.m

```

function InitialData

%Type == 1, Circular Boundary - == 2, Rectangular Boundary
Type = get(findobj(gcf,'Tag','BOUNDARY'),'Value');
NSectors = 2;
xB(1) = str2num(get(findobj(gcf,'Tag','xLE'),'String'));
yB(1) = str2num(get(findobj(gcf,'Tag','yLE'),'String'));
CHORD = str2num(get(findobj(gcf,'Tag','chord'),'String'));
xB(2) = str2num(get(findobj(gcf,'Tag','xRTCTR'),'String'));
yB(2) = 0;
xB(3) = xB(2);
r = str2num(get(findobj(gcf,'Tag','R'),'String'));
yB(3) = r;
m(1) = str2num(get(findobj(gcf,'Tag','M1'),'String'));
m(2) = str2num(get(findobj(gcf,'Tag','M2'),'String'));
m(3) = str2num(get(findobj(gcf,'Tag','M3'),'String'));
m(4) = str2num(get(findobj(gcf,'Tag','M4'),'String'));
%AirfoilType == 1 NACA, == 2 Flat Plate, == 3 Read-in From File
AirfoilType = get(findobj(gcf,'Tag','AIRFOIL'),'Value');
NACA4 = str2num(get(findobj(gcf,'Tag','NACA'),'String'));
thick1 = str2num(get(findobj(gcf,'Tag','T1'),'String'));
thick2 = str2num(get(findobj(gcf,'Tag','T2'),'String'));
thick3 = str2num(get(findobj(gcf,'Tag','T3'),'String'));
SRigid = str2num(get(findobj(gcf,'Tag','SizeRigid'),'String'));
sizesmooth = str2num(get(findobj(gcf,'Tag','SizeSmooth'),'String'));
sizesmoothL = str2num(get(findobj(gcf,'Tag','SizeSmoothL'),'String'));
bendslopebot=str2num(get(findobj(gcf,'Tag','BendSlopeBot'),'String'));
bendslopetop=str2num(get(findobj(gcf,'Tag','BendSlopeTop'),'String'));

```

```

    aoa = str2num(get(findobj(gcf,'Tag','AOA'),'String'));
    aoa = aoa * pi / 180;
    xo = str2num(get(findobj(gcf,'Tag','xo'),'String'));
    yo = str2num(get(findobj(gcf,'Tag','yo'),'String'));
    plunge = str2num(get(findobj(gcf,'Tag','PLUNGE'),'String'));
    RBPitch = str2num(get(findobj(gcf,'Tag','RtBPitch'),'String'));
    LBPitch = str2num(get(findobj(gcf,'Tag','LtBPitch'),'String'));
    RBPlunge = str2num(get(findobj(gcf,'Tag','RtBPlunge'),'String'));
    LBPlunge = str2num(get(findobj(gcf,'Tag','LtBPlunge'),'String'));
    obendslopebot=str2num(get(findobj(gcf,'Tag','OBendSlopeBot'),'String'));
    obendslopetop=str2num(get(findobj(gcf,'Tag','OBendSlopeTop'),'String'));

    if NACA4 < 1000
        MaxCamber = 0;
        PosMaxCamber = 0;
    else
        MaxCamber = floor(NACA4 / 1000) * 0.01;
        PosMaxCamber = floor((NACA4 / 100) - 1000 * MaxCamber) * 0.1;
    end
    Thickness = floor(NACA4 - 100000 * MaxCamber - ...
        1000 * PosMaxCamber) * 0.01;

    xTE = xB(1) + CHORD + 0.5 * SRigid;
    xTESelect = 1;

    save InitData Type NSectors xB yB CHORD r m thick1 thick2 thick3 ...
        SRigid sizesmooth sizesmoothL bendslopebot bendslopetop aoa ...
        xo yo plunge RBPitch RBPlunge LBPitch LBPlunge AirfoilType ...
        obendslopetop obendslopebot MaxCamber PosMaxCamber Thickness ...
        xTE xTESelect;
    X = zeros(max(m(1),m(3)),max(m(2),m(4)),2);
    Y = zeros(max(m(1),m(3)),max(m(2),m(4)),2);
    save Results X Y
    UAP = 0;
    save PQData UAP
    NUM = 0;
    save Intermediate NUM
    COMPLETE = 1;
    save check COMPLETE
    if AirfoilType == 3
        AirfoilReadIn
    else
        close
        NumAirfoilSegments
    end
end

```

7. PlotEntire.m

```

function PlotEntire(Init)

load InitData
load Intermediate

if Init == 0
    load Results

```

```

else
    load ResultsPP
end

SECT = 1;    %FOR FUTURE MODIFICATIONS (WILL ALLOW MULTIPLE SECTORS)

%PLOT UPPER HORIZONTAL GRID LINES
figure
hold
for j = 1:m(2*SECT)
    plot(X(:,j,SECT),Y(:,j,SECT))
end

%PLOT UPPER VERTICAL GRID LINES
for i = 1:m(2*SECT-1)
    plot(X(i, :, SECT),Y(i, :, SECT))
end

SECT = SECT + 1;

%PLOT LOWER HORIZONTAL GRID LINES
for j = 1:m(2*SECT)
    plot(X(:,j,SECT),Y(:,j,SECT))
end

%PLOT LOWER VERTICAL GRID LINES
for i = 1:m(2*SECT-1)
    plot(X(i, :, SECT),Y(i, :, SECT))
end

axis equal

```

8. PlotRigid.m

```

function PlotRigid

load InitData
load Intermediate
load Results

SECT = 1;    %FOR FUTURE MODIFICATIONS (WILL ALLOW MULTIPLE SECTORS)

xUppNew = X(1:EDGER,1:EDGEL,SECT);
yUppNew = Y(1:EDGER,1:EDGEL,SECT);

%PLOT UPPER HORIZONTAL GRID LINES
figure
hold
for j = 1:EDGEL
    plot(xUppNew(:,j),yUppNew(:,j))
end

%PLOT UPPER VERTICAL GRID LINES

```

```

for i = 1:EDGER
    plot(xUppNew(i,:),yUppNew(i,:))
end

SECT = SECT + 1;

xLowNew = X(1:EDGER,m(2*SECT):-1:m(2*SECT)-EDGEL+1,SECT);
yLowNew = Y(1:EDGER,m(2*SECT):-1:m(2*SECT)-EDGEL+1,SECT);

%PLOT LOWER HORIZONTAL GRID LINES
for j = 1:EDGEL
    plot(xLowNew(:,j),yLowNew(:,j))
end

%PLOT LOWER VERTICAL GRID LINES
for i = 1:EDGER
    plot(xLowNew(i,:),yLowNew(i,:))
end

axis equal

```

9. PQData.m

```

function PQData

NumUpperSeg = str2num(get(findobj(gcf,'Tag','NumUpperSeg'),'String'));
NumLowerSeg = str2num(get(findobj(gcf,'Tag','NumLowerSeg'),'String'));
close
if NumUpperSeg == 1 & NumLowerSeg == 1
    EnterPQ11
elseif NumUpperSeg == 2 & NumLowerSeg == 2
    EnterPQ22
elseif NumUpperSeg == 3 & NumLowerSeg == 3
    EnterPQ33
elseif NumUpperSeg == 2 & NumLowerSeg == 1
    EnterPQ21
elseif NumUpperSeg == 1 & NumLowerSeg == 2
    EnterPQ12
elseif NumUpperSeg == 3 & NumLowerSeg == 1
    EnterPQ31
elseif NumUpperSeg == 1 & NumLowerSeg == 3
    EnterPQ13
elseif NumUpperSeg == 3 & NumLowerSeg == 2
    EnterPQ32
elseif NumUpperSeg == 2 & NumLowerSeg == 3
    EnterPQ23
elseif NumUpperSeg > 3 | NumLowerSeg > 3
    error('Maximum Number of Segments is 3!')
    NumAirfoilSegments
else
    error('Error Entering Number of Segments')
    NumAirfoilSegments
end

```

```
save PQData NumUpperSeg NumLowerSeg -APPEND
```

10. RedoInitial.m

```
function RedoInitial

load InitData

close

AstroGrid

a = gcf;

set(findobj(a, 'Tag', 'BOUNDARY'), 'Value', Type);
if Type == 1
    set(findobj(a, 'Tag', 'RadORHt'), 'String', 'Boundary Circle Radius');
else
    set(findobj(a, 'Tag', 'RadORHt'), 'String', 'Boundary Semi-Height');
end
set(findobj(a, 'Tag', 'xLE'), 'String', xB(1));
set(findobj(a, 'Tag', 'yLE'), 'String', yB(1));
set(findobj(a, 'Tag', 'chord'), 'String', CHORD);
set(findobj(a, 'Tag', 'xRTCTR'), 'String', xB(2));
set(findobj(a, 'Tag', 'R'), 'String', r);
set(findobj(a, 'Tag', 'M1'), 'String', m(1));
set(findobj(a, 'Tag', 'M2'), 'String', m(2));
set(findobj(a, 'Tag', 'M3'), 'String', m(3));
set(findobj(a, 'Tag', 'M4'), 'String', m(4));
set(findobj(a, 'Tag', 'AIRFOIL'), 'Value', AirfoilType);
if AirfoilType == 1
    set(findobj(a, 'Tag', 'AirfoilDesig'), 'String', '4-Digit NACA Airfoil Designation');
    set(findobj(a, 'Tag', 'NACA'), 'String', '0014');
elseif AirfoilType == 2
    set(findobj(a, 'Tag', 'AirfoilDesig'), 'String', 'Flat Plate Percent Thickness');
    set(findobj(a, 'Tag', 'NACA'), 'String', '5');
else
    load PQData
    set(findobj(a, 'Tag', 'AirfoilDesig'), 'String', 'File Location');
    set(findobj(a, 'Tag', 'NACA'), 'String', FileLocation);
    set(findobj(a, 'Tag', 'NACA'), 'Position', [63.800000000000004
35.38461538461539 32.2 1.6153846153846154]);
    set(findobj(a, 'Tag', 'AirfoilDesig'), 'Position', [47.80000000000001
35.38461538461539 16.2 1.6153846153846154]);
    set(findobj(a, 'Tag', 'NACA'), 'HorizontalAlignment', 'left');
end
set(findobj(a, 'Tag', 'T1'), 'String', thick1);
set(findobj(a, 'Tag', 'T2'), 'String', thick2);
set(findobj(a, 'Tag', 'T3'), 'String', thick3);
set(findobj(a, 'Tag', 'SizeRigid'), 'String', SRigid);
set(findobj(a, 'Tag', 'SizeSmooth'), 'String', sizesmooth);
set(findobj(a, 'Tag', 'SizeSmoothL'), 'String', sizesmoothL);
```

```

set(findobj(a, 'Tag', 'BendSlopeBot'), 'String', bendslopebot);
set(findobj(a, 'Tag', 'BendSlopeTop'), 'String', bendslopetop);
aoa = aoa * 180 / pi;
set(findobj(a, 'Tag', 'AOA'), 'String', aoa);
set(findobj(a, 'Tag', 'xo'), 'String', xo);
set(findobj(a, 'Tag', 'yo'), 'String', yo);
set(findobj(a, 'Tag', 'PLUNGE'), 'String', plunge);
set(findobj(a, 'Tag', 'RtBPitch'), 'String', RBPitch);
set(findobj(a, 'Tag', 'LtBPitch'), 'String', LBPitch);
set(findobj(a, 'Tag', 'RtBPlunge'), 'String', RBPlunge);
set(findobj(a, 'Tag', 'LtBPlunge'), 'String', LBPlunge);
set(findobj(a, 'Tag', 'OBendSlopeBot'), 'String', obendslopebot);
set(findobj(a, 'Tag', 'OBendSlopeTop'), 'String', obendslopetop);

```

11. RedoNumAirfoilSegments.m

```

function RedoNumAirfoilSegments

close

load InitData
load PQData

if AirfoilType == 3
    RedoInitial
else
    NumAirfoilSegments

    a = gcf;

    set(findobj(a, 'Tag', 'NumUpperSeg'), 'String', NumUpperSeg);
    set(findobj(a, 'Tag', 'NumLowerSeg'), 'String', NumLowerSeg);
end

```

12. RedoOuterGrid.m

```

function RedoOuterGrid

close

OuterGrid

```

13. RedoPQData.m

```

function RedoPQData

close

load PQData
load InitData

```

```

if AirfoilType ~= 3
    if NumUpperSeg == 1 & NumLowerSeg == 1
        EnterPQ11
    elseif NumUpperSeg == 2 & NumLowerSeg == 2
        EnterPQ22
    elseif NumUpperSeg == 3 & NumLowerSeg == 3
        EnterPQ33
    elseif NumUpperSeg == 2 & NumLowerSeg == 1
        EnterPQ21
    elseif NumUpperSeg == 1 & NumLowerSeg == 2
        EnterPQ12
    elseif NumUpperSeg == 3 & NumLowerSeg == 1
        EnterPQ31
    elseif NumUpperSeg == 1 & NumLowerSeg == 3
        EnterPQ13
    elseif NumUpperSeg == 3 & NumLowerSeg == 2
        EnterPQ32
    elseif NumUpperSeg == 2 & NumLowerSeg == 3
        EnterPQ23
    elseif NumUpperSeg > 3 | NumLowerSeg > 3
        error('Maximum Number of Segments is 3!')
        NumAirfoilSegments
    else
        error('Error Entering Number of Segments')
        NumAirfoilSegments
    end
end

a = gcf;

if NumUpperSeg == 1
    set(findobj(a, 'Tag', 'UANUM1'), 'String', UANUM(1));
    set(findobj(a, 'Tag', 'UAP1'), 'String', UAP(1));
    set(findobj(a, 'Tag', 'UAQ1'), 'String', UAQ(1));
    set(findobj(a, 'Tag', 'Dir1'), 'String', Dir(1));
elseif NumUpperSeg == 2
    set(findobj(a, 'Tag', 'UANUM1'), 'String', UANUM(1));
    set(findobj(a, 'Tag', 'UAP1'), 'String', UAP(1));
    set(findobj(a, 'Tag', 'UAQ1'), 'String', UAQ(1));
    set(findobj(a, 'Tag', 'Dir1'), 'String', Dir(1));
    set(findobj(a, 'Tag', 'EdgeUpperSeg1'), 'String', EdgeUpperSeg(1));
    set(findobj(a, 'Tag', 'UANUM2'), 'String', UANUM(2));
    set(findobj(a, 'Tag', 'UAP2'), 'String', UAP(2));
    set(findobj(a, 'Tag', 'UAQ2'), 'String', UAQ(2));
    set(findobj(a, 'Tag', 'Dir2'), 'String', Dir(2));
elseif NumUpperSeg == 3
    set(findobj(a, 'Tag', 'UANUM1'), 'String', UANUM(1));
    set(findobj(a, 'Tag', 'UAP1'), 'String', UAP(1));
    set(findobj(a, 'Tag', 'UAQ1'), 'String', UAQ(1));
    set(findobj(a, 'Tag', 'Dir1'), 'String', Dir(1));
    set(findobj(a, 'Tag', 'EdgeUpperSeg1'), 'String', EdgeUpperSeg(1));
    set(findobj(a, 'Tag', 'UANUM2'), 'String', UANUM(2));
    set(findobj(a, 'Tag', 'UAP2'), 'String', UAP(2));
    set(findobj(a, 'Tag', 'UAQ2'), 'String', UAQ(2));
    set(findobj(a, 'Tag', 'Dir2'), 'String', Dir(2));
end

```

```

        set(findobj(a, 'Tag', 'EdgeUpperSeg2'), 'String', EdgeUpperSeg(2));
        set(findobj(a, 'Tag', 'UANUM3'), 'String', UANUM(3));
        set(findobj(a, 'Tag', 'UAP3'), 'String', UAP(3));
        set(findobj(a, 'Tag', 'UAQ3'), 'String', UAQ(3));
        set(findobj(a, 'Tag', 'Dir3'), 'String', Dir(3));
    else
        error('NumUpperSeg ~= 1, 2, or 3')
    end

    if NumLowerSeg == 1
        set(findobj(a, 'Tag', 'LANUM1'), 'String', LANUM(1));
        set(findobj(a, 'Tag', 'LAP1'), 'String', LAP(1));
        set(findobj(a, 'Tag', 'LAQ1'), 'String', LAQ(1));
        set(findobj(a, 'Tag', 'LDir1'), 'String', LDir(1));
    elseif NumLowerSeg == 2
        set(findobj(a, 'Tag', 'LANUM1'), 'String', LANUM(1));
        set(findobj(a, 'Tag', 'LAP1'), 'String', LAP(1));
        set(findobj(a, 'Tag', 'LAQ1'), 'String', LAQ(1));
        set(findobj(a, 'Tag', 'LDir1'), 'String', LDir(1));
        set(findobj(a, 'Tag', 'EdgeLowerSeg1'), 'String', EdgeLowerSeg(1));
        set(findobj(a, 'Tag', 'LANUM2'), 'String', LANUM(2));
        set(findobj(a, 'Tag', 'LAP2'), 'String', LAP(2));
        set(findobj(a, 'Tag', 'LAQ2'), 'String', LAQ(2));
        set(findobj(a, 'Tag', 'LDir2'), 'String', LDir(2));
    elseif NumLowerSeg == 3
        set(findobj(a, 'Tag', 'LANUM1'), 'String', LANUM(1));
        set(findobj(a, 'Tag', 'LAP1'), 'String', LAP(1));
        set(findobj(a, 'Tag', 'LAQ1'), 'String', LAQ(1));
        set(findobj(a, 'Tag', 'LDir1'), 'String', LDir(1));
        set(findobj(a, 'Tag', 'EdgeLowerSeg1'), 'String', EdgeLowerSeg(1));
        set(findobj(a, 'Tag', 'LANUM2'), 'String', LANUM(2));
        set(findobj(a, 'Tag', 'LAP2'), 'String', LAP(2));
        set(findobj(a, 'Tag', 'LAQ2'), 'String', LAQ(2));
        set(findobj(a, 'Tag', 'LDir2'), 'String', LDir(2));
        set(findobj(a, 'Tag', 'EdgeLowerSeg2'), 'String', EdgeLowerSeg(2));
        set(findobj(a, 'Tag', 'LANUM3'), 'String', LANUM(3));
        set(findobj(a, 'Tag', 'LAP3'), 'String', LAP(3));
        set(findobj(a, 'Tag', 'LAQ3'), 'String', LAQ(3));
        set(findobj(a, 'Tag', 'LDir3'), 'String', LDir(3));
    else
        error('NumLowerSeg ~= 1, 2, or 3')
    end
else
    EnterPQFile
    a = gcf;
    set(findobj(a, 'Tag', 'UANUM'), 'String', UANUM);
    set(findobj(a, 'Tag', 'LANUM'), 'String', LANUM);
end

set(findobj(a, 'Tag', 'AWP'), 'String', AWP);
set(findobj(a, 'Tag', 'AWQ'), 'String', AWQ);
set(findobj(a, 'Tag', 'LSP'), 'String', LSP);
set(findobj(a, 'Tag', 'LSQ'), 'String', LSQ);
set(findobj(a, 'Tag', 'URP'), 'String', URP);
set(findobj(a, 'Tag', 'URQ'), 'String', URQ);

```

```
set(findobj(a, 'Tag', 'LRP'), 'String', LRP);
set(findobj(a, 'Tag', 'LRQ'), 'String', LRQ);
```

14. SelectAirfoilType.m

```
function SelectAirfoilType

AirfoilType = get(findobj(gcf, 'Tag', 'AIRFOIL'), 'Value');
a = gcf;

if AirfoilType == 1
    set(findobj(a, 'Tag', 'AirfoilDesig'), 'String', '4-Digit NACA Airfoil
Designation');
    set(findobj(a, 'Tag', 'NACA'), 'String', '0014');
    set(findobj(a, 'Tag', 'NACA'), 'Position', [87.80000000000001 ...
35.38461538461539 8.200000000000001 1.6153846153846154]);
    set(findobj(a, 'Tag', 'AirfoilDesig'), 'Position', ...
[47.80000000000001 34.61538461538462 36.2 2.3846153846153846]);
    set(findobj(a, 'Tag', 'NACA'), 'HorizontalAlignment', 'center');
elseif AirfoilType == 2
    set(findobj(a, 'Tag', 'AirfoilDesig'), 'String', ...
'Flat Plate Percent Thickness');
    set(findobj(a, 'Tag', 'NACA'), 'String', '5');
    set(findobj(a, 'Tag', 'NACA'), 'Position', [87.80000000000001 ...
35.38461538461539 8.200000000000001 1.6153846153846154]);
    set(findobj(a, 'Tag', 'AirfoilDesig'), 'Position', ...
[47.80000000000001 34.61538461538462 36.2 2.3846153846153846]);
    set(findobj(a, 'Tag', 'NACA'), 'HorizontalAlignment', 'center');
else
    set(findobj(a, 'Tag', 'AirfoilDesig'), 'String', 'File Location');
    set(findobj(a, 'Tag', 'NACA'), 'String', ...
'C:\Documents and Settings\Justin M. Verville\Desktop\
AirfoilReadIn.txt');
    set(findobj(a, 'Tag', 'NACA'), 'Position', [63.800000000000004 ...
35.38461538461539 32.2 1.6153846153846154]);
    set(findobj(a, 'Tag', 'AirfoilDesig'), 'Position', ...
[47.80000000000001 35.38461538461539 16.2 1.6153846153846154]);
    set(findobj(a, 'Tag', 'NACA'), 'HorizontalAlignment', 'left');
end
```

15. SelectBoundary.m

```
function SelectBoundary

Type = get(findobj(gcf, 'Tag', 'BOUNDARY'), 'Value');
a = gcf;

if Type == 1
    set(findobj(a, 'Tag', 'RadORht'), 'String', 'Boundary Circle Radius');
else
    set(findobj(a, 'Tag', 'RadORht'), 'String', 'Boundary Semi-Height');
end
```

16. SelectxTE.m

```
function SelectxTE

load InitData

xTESelect = get(findobj(gcf,'Tag','xTESelect'),'Value');
a = gcf;

if xTESelect == 1
    xTE = xB(1) + CHORD + 0.5 * SRigid;
    set(findobj(a,'Tag','Xte'),'String',xTE);
else
    xTE = str2num(get(findobj(gcf,'Tag','Xte'),'String'));
end

save InitData xTE xTESelect -APPEND
```

17. UpdateAdjustRigid.m

```
function UpdateAdjustRigid

load InitData

AdjustRigid

a = gcf;

set(findobj(a,'Tag','SizeRigid'),'String',SRigid);
set(findobj(a,'Tag','SizeSmooth'),'String',sizesmooth);
set(findobj(a,'Tag','SizeSmoothL'),'String',sizesmoothL);
set(findobj(a,'Tag','BendSlopeBot'),'String',bendslopebot);
set(findobj(a,'Tag','BendSlopeTop'),'String',bendslopetop);
if AirfoilType == 1
    set(findobj(a,'Tag','Xte'),'String','-');
    set(findobj(a,'Tag','xTESelect'),'String','N/A');
else
    set(findobj(a,'Tag','Xte'),'String',xTE);
    if xTESelect == 1
        set(findobj(a,'Tag','xTESelect'),'Value',1);
    else
        set(findobj(a,'Tag','xTESelect'),'Value',2);
    end
end

end
```

18. UpdateOuterGrid.m

```
function UpdateOuterGrid

obendslopebot=str2num(get(findobj(gcf,'Tag','OBendSlopeBot'),'String'));

save InitData obendslopebot -APPEND
```

```
close
```

19. ViewEntire.m

```
function ViewEntire

load InitData
load PQData

if AirfoilType ~= 3
    if NumUpperSeg == 1
        UANUM(1) = str2num(get(findobj(gcf,'Tag','UANUM1'),'String'));
        UAP(1) = str2num(get(findobj(gcf,'Tag','UAP1'),'String'));
        UAQ(1) = str2num(get(findobj(gcf,'Tag','UAQ1'),'String'));
        Dir(1) = str2num(get(findobj(gcf,'Tag','Dir1'),'String'));
        EdgeUpperSeg(1) = 1.0;
    elseif NumUpperSeg == 2
        UANUM(1) = str2num(get(findobj(gcf,'Tag','UANUM1'),'String'));
        UAP(1) = str2num(get(findobj(gcf,'Tag','UAP1'),'String'));
        UAQ(1) = str2num(get(findobj(gcf,'Tag','UAQ1'),'String'));
        Dir(1) = str2num(get(findobj(gcf,'Tag','Dir1'),'String'));
        EdgeUpperSeg(1) = str2num(get(findobj(gcf,'Tag', ...
            'EdgeUpperSeg1'),'String'));
        UANUM(2) = str2num(get(findobj(gcf,'Tag','UANUM2'),'String'));
        UAP(2) = str2num(get(findobj(gcf,'Tag','UAP2'),'String'));
        UAQ(2) = str2num(get(findobj(gcf,'Tag','UAQ2'),'String'));
        Dir(2) = str2num(get(findobj(gcf,'Tag','Dir2'),'String'));
        EdgeUpperSeg(2) = 1.0;
    elseif NumUpperSeg == 3
        UANUM(1) = str2num(get(findobj(gcf,'Tag','UANUM1'),'String'));
        UAP(1) = str2num(get(findobj(gcf,'Tag','UAP1'),'String'));
        UAQ(1) = str2num(get(findobj(gcf,'Tag','UAQ1'),'String'));
        Dir(1) = str2num(get(findobj(gcf,'Tag','Dir1'),'String'));
        EdgeUpperSeg(1) = str2num(get(findobj(gcf,'Tag', ...
            'EdgeUpperSeg1'),'String'));
        UANUM(2) = str2num(get(findobj(gcf,'Tag','UANUM2'),'String'));
        UAP(2) = str2num(get(findobj(gcf,'Tag','UAP2'),'String'));
        UAQ(2) = str2num(get(findobj(gcf,'Tag','UAQ2'),'String'));
        Dir(2) = str2num(get(findobj(gcf,'Tag','Dir2'),'String'));
        EdgeUpperSeg(2) = str2num(get(findobj(gcf,'Tag', ...
            'EdgeUpperSeg2'),'String'));
        UANUM(3) = str2num(get(findobj(gcf,'Tag','UANUM3'),'String'));
        UAP(3) = str2num(get(findobj(gcf,'Tag','UAP3'),'String'));
        UAQ(3) = str2num(get(findobj(gcf,'Tag','UAQ3'),'String'));
        Dir(3) = str2num(get(findobj(gcf,'Tag','Dir3'),'String'));
        EdgeUpperSeg(3) = 1.0;
    else
        error('NumUpperSeg ~= 1, 2, or 3')
    end

    if NumLowerSeg == 1
        LANUM(1) = str2num(get(findobj(gcf,'Tag','LANUM1'),'String'));
        LAP(1) = str2num(get(findobj(gcf,'Tag','LAP1'),'String'));
```

```

    LAQ(1) = str2num(get(findobj(gcbf, 'Tag', 'LAQ1'), 'String'));
    LDir(1) = str2num(get(findobj(gcbf, 'Tag', 'LDir1'), 'String'));
    EdgeLowerSeg(1) = 1.0;
elseif NumLowerSeg == 2
    LANUM(1) = str2num(get(findobj(gcbf, 'Tag', 'LANUM1'), 'String'));
    LAP(1) = str2num(get(findobj(gcbf, 'Tag', 'LAP1'), 'String'));
    LAQ(1) = str2num(get(findobj(gcbf, 'Tag', 'LAQ1'), 'String'));
    LDir(1) = str2num(get(findobj(gcbf, 'Tag', 'LDir1'), 'String'));
    EdgeLowerSeg(1) = str2num(get(findobj(gcbf, 'Tag', ...
        'EdgeLowerSeg1'), 'String'));
    LANUM(2) = str2num(get(findobj(gcbf, 'Tag', 'LANUM2'), 'String'));
    LAP(2) = str2num(get(findobj(gcbf, 'Tag', 'LAP2'), 'String'));
    LAQ(2) = str2num(get(findobj(gcbf, 'Tag', 'LAQ2'), 'String'));
    LDir(2) = str2num(get(findobj(gcbf, 'Tag', 'LDir2'), 'String'));
    EdgeLowerSeg(2) = 1.0;
elseif NumLowerSeg == 3
    LANUM(1) = str2num(get(findobj(gcbf, 'Tag', 'LANUM1'), 'String'));
    LAP(1) = str2num(get(findobj(gcbf, 'Tag', 'LAP1'), 'String'));
    LAQ(1) = str2num(get(findobj(gcbf, 'Tag', 'LAQ1'), 'String'));
    LDir(1) = str2num(get(findobj(gcbf, 'Tag', 'LDir1'), 'String'));
    EdgeLowerSeg(1) = str2num(get(findobj(gcbf, 'Tag', ...
        'EdgeLowerSeg1'), 'String'));
    LANUM(2) = str2num(get(findobj(gcbf, 'Tag', 'LANUM2'), 'String'));
    LAP(2) = str2num(get(findobj(gcbf, 'Tag', 'LAP2'), 'String'));
    LAQ(2) = str2num(get(findobj(gcbf, 'Tag', 'LAQ2'), 'String'));
    LDir(2) = str2num(get(findobj(gcbf, 'Tag', 'LDir2'), 'String'));
    EdgeLowerSeg(2) = str2num(get(findobj(gcbf, 'Tag', ...
        'EdgeLowerSeg2'), 'String'));
    LANUM(3) = str2num(get(findobj(gcbf, 'Tag', 'LANUM3'), 'String'));
    LAP(3) = str2num(get(findobj(gcbf, 'Tag', 'LAP3'), 'String'));
    LAQ(3) = str2num(get(findobj(gcbf, 'Tag', 'LAQ3'), 'String'));
    LDir(3) = str2num(get(findobj(gcbf, 'Tag', 'LDir3'), 'String'));
    EdgeLowerSeg(3) = 1.0;
else
    error('NumLowerSeg ~= 1, 2, or 3')
end

save PQData UANUM UAP UAQ Dir EdgeUpperSeg ...
    LANUM LAP LAQ LDir EdgeLowerSeg -APPEND

end

AWP = str2num(get(findobj(gcbf, 'Tag', 'AWP'), 'String'));
AWQ = str2num(get(findobj(gcbf, 'Tag', 'AWQ'), 'String'));
LSP = str2num(get(findobj(gcbf, 'Tag', 'LSP'), 'String'));
LSQ = str2num(get(findobj(gcbf, 'Tag', 'LSQ'), 'String'));
URP = str2num(get(findobj(gcbf, 'Tag', 'URP'), 'String'));
URQ = str2num(get(findobj(gcbf, 'Tag', 'URQ'), 'String'));
LRP = str2num(get(findobj(gcbf, 'Tag', 'LRP'), 'String'));
LRQ = str2num(get(findobj(gcbf, 'Tag', 'LRQ'), 'String'));

save PQData AWP AWQ LSP LSQ URP URQ LRP LRQ -APPEND

if AirfoilType ~= 3
    for i = 1:NumUpperSeg

```

```

        if AirfoilType == 1
            UpperAirfoil(1,i)
        elseif AirfoilType == 2
            UpperAirfoilRTE(1,i)
        end
    end
end

AirfoilWakeInit(1)
LeftSurfaceInit(1)
UpperRightVerticalInit(1)
LowerRightVerticalInit(1)

if AirfoilType ~= 3
    for i = 1:NumLowerSeg
        if AirfoilType == 1
            LowerAirfoil(1,i)
        elseif AirfoilType == 2
            LowerAirfoilRTE(1,i)
        end
    end
end

SECT=2;

load Intermediate

if m(2*SECT-1) - NUM ~= m(2*(SECT-1)-1) - NUM2
    error('The correct number of points must be allotted for the lower
        surface of the airfoil wake')
end

Duplicate(0)
UpperRigid
LowerRigid
close
OuterGrid

```

20. ViewEntireGrid.m

```

function ViewEntireGrid

load InitData

WakeUpper(0)
WakeLower(0)
if Type == 1
    UpperTop(0)
    ArcLengthTop
    LowerBottom(0)
    ArcLengthBottom
else
    RectUpperTop(0)
    RectArcLengthTop

```

```

    RectLowerBottom(0)
    RectArcLengthBottom
end
UpperHorizontal(0)
LowerHorizontal(0)

COMPLETE = 0;
save check COMPLETE

PlotEntire(0)

```

21. ViewLeadingEdgeLine.m

```

function ViewLeadingEdgeLine

load PQData
load InitData

if AirfoilType ~=3
    if NumUpperSeg == 1
        UANUM(1) = str2num(get(findobj(gcf,'Tag','UANUM1'),'String'));
        UAP(1) = str2num(get(findobj(gcf,'Tag','UAP1'),'String'));
        UAQ(1) = str2num(get(findobj(gcf,'Tag','UAQ1'),'String'));
        Dir(1) = str2num(get(findobj(gcf,'Tag','Dir1'),'String'));
        EdgeUpperSeg(1) = 1.0;
    elseif NumUpperSeg == 2
        UANUM(1) = str2num(get(findobj(gcf,'Tag','UANUM1'),'String'));
        UAP(1) = str2num(get(findobj(gcf,'Tag','UAP1'),'String'));
        UAQ(1) = str2num(get(findobj(gcf,'Tag','UAQ1'),'String'));
        Dir(1) = str2num(get(findobj(gcf,'Tag','Dir1'),'String'));
        EdgeUpperSeg(1) = str2num(get(findobj(gcf,'Tag', ...
            'EdgeUpperSeg1'),'String'));
        UANUM(2) = str2num(get(findobj(gcf,'Tag','UANUM2'),'String'));
        UAP(2) = str2num(get(findobj(gcf,'Tag','UAP2'),'String'));
        UAQ(2) = str2num(get(findobj(gcf,'Tag','UAQ2'),'String'));
        Dir(2) = str2num(get(findobj(gcf,'Tag','Dir2'),'String'));
        EdgeUpperSeg(2) = 1.0;
    elseif NumUpperSeg == 3
        UANUM(1) = str2num(get(findobj(gcf,'Tag','UANUM1'),'String'));
        UAP(1) = str2num(get(findobj(gcf,'Tag','UAP1'),'String'));
        UAQ(1) = str2num(get(findobj(gcf,'Tag','UAQ1'),'String'));
        Dir(1) = str2num(get(findobj(gcf,'Tag','Dir1'),'String'));
        EdgeUpperSeg(1) = str2num(get(findobj(gcf,'Tag', ...
            'EdgeUpperSeg1'),'String'));
        UANUM(2) = str2num(get(findobj(gcf,'Tag','UANUM2'),'String'));
        UAP(2) = str2num(get(findobj(gcf,'Tag','UAP2'),'String'));
        UAQ(2) = str2num(get(findobj(gcf,'Tag','UAQ2'),'String'));
        Dir(2) = str2num(get(findobj(gcf,'Tag','Dir2'),'String'));
        EdgeUpperSeg(2) = str2num(get(findobj(gcf,'Tag', ...
            'EdgeUpperSeg2'),'String'));
        UANUM(3) = str2num(get(findobj(gcf,'Tag','UANUM3'),'String'));
        UAP(3) = str2num(get(findobj(gcf,'Tag','UAP3'),'String'));
        UAQ(3) = str2num(get(findobj(gcf,'Tag','UAQ3'),'String'));
    end
end

```

```

    Dir(3) = str2num(get(findobj(gcbf, 'Tag', 'Dir3'), 'String'));
    EdgeUpperSeg(3) = 1.0;
else
    error('NumUpperSeg ~= 1, 2, or 3')
end

save PQData UANUM UAP UAQ Dir EdgeUpperSeg -APPEND

end

AWP = str2num(get(findobj(gcbf, 'Tag', 'AWP'), 'String'));
AWQ = str2num(get(findobj(gcbf, 'Tag', 'AWQ'), 'String'));
LSP = str2num(get(findobj(gcbf, 'Tag', 'LSP'), 'String'));
LSQ = str2num(get(findobj(gcbf, 'Tag', 'LSQ'), 'String'));

save PQData AWP AWQ LSP LSQ -APPEND

LeftSurfaceInit(0)

```

22. ViewLowerAirfoil.m

```

function ViewLowerAirfoil(Seg)

load PQData
load InitData

if AirfoilType ~= 3
    if NumLowerSeg == 1
        LANUM(1) = str2num(get(findobj(gcbf, 'Tag', 'LANUM1'), 'String'));
        LAP(1) = str2num(get(findobj(gcbf, 'Tag', 'LAP1'), 'String'));
        LAQ(1) = str2num(get(findobj(gcbf, 'Tag', 'LAQ1'), 'String'));
        LDir(1) = str2num(get(findobj(gcbf, 'Tag', 'LDir1'), 'String'));
        EdgeLowerSeg(1) = 1.0;
    elseif NumLowerSeg == 2
        LANUM(1) = str2num(get(findobj(gcbf, 'Tag', 'LANUM1'), 'String'));
        LAP(1) = str2num(get(findobj(gcbf, 'Tag', 'LAP1'), 'String'));
        LAQ(1) = str2num(get(findobj(gcbf, 'Tag', 'LAQ1'), 'String'));
        LDir(1) = str2num(get(findobj(gcbf, 'Tag', 'LDir1'), 'String'));
        EdgeLowerSeg(1) = str2num(get(findobj(gcbf, 'Tag', ...
            'EdgeLowerSeg1'), 'String'));
        LANUM(2) = str2num(get(findobj(gcbf, 'Tag', 'LANUM2'), 'String'));
        LAP(2) = str2num(get(findobj(gcbf, 'Tag', 'LAP2'), 'String'));
        LAQ(2) = str2num(get(findobj(gcbf, 'Tag', 'LAQ2'), 'String'));
        LDir(2) = str2num(get(findobj(gcbf, 'Tag', 'LDir2'), 'String'));
        EdgeLowerSeg(2) = 1.0;
    elseif NumLowerSeg == 3
        LANUM(1) = str2num(get(findobj(gcbf, 'Tag', 'LANUM1'), 'String'));
        LAP(1) = str2num(get(findobj(gcbf, 'Tag', 'LAP1'), 'String'));
        LAQ(1) = str2num(get(findobj(gcbf, 'Tag', 'LAQ1'), 'String'));
        LDir(1) = str2num(get(findobj(gcbf, 'Tag', 'LDir1'), 'String'));
        EdgeLowerSeg(1) = str2num(get(findobj(gcbf, 'Tag', ...
            'EdgeLowerSeg1'), 'String'));
        LANUM(2) = str2num(get(findobj(gcbf, 'Tag', 'LANUM2'), 'String'));

```

```

LAP(2) = str2num(get(findobj(gcf,'Tag','LAP2'),'String'));
LAQ(2) = str2num(get(findobj(gcf,'Tag','LAQ2'),'String'));
LDir(2) = str2num(get(findobj(gcf,'Tag','LDir2'),'String'));
EdgeLowerSeg(2) = str2num(get(findobj(gcf,'Tag', ...
    'EdgeLowerSeg2'),'String'));
LANUM(3) = str2num(get(findobj(gcf,'Tag','LANUM3'),'String'));
LAP(3) = str2num(get(findobj(gcf,'Tag','LAP3'),'String'));
LAQ(3) = str2num(get(findobj(gcf,'Tag','LAQ3'),'String'));
LDir(3) = str2num(get(findobj(gcf,'Tag','LDir3'),'String'));
EdgeLowerSeg(3) = 1.0;
else
    error('NumLowerSeg ~= 1, 2, or 3')
end

save PQData LANUM LAP LAQ LDir EdgeLowerSeg -APPEND

end

if AirfoilType == 1
    LowerAirfoil(0,Seg)
elseif AirfoilType == 2
    LowerAirfoilRTE(0,Seg)
else
    load Results
    SECT = 2;
    figure
    plot(X(1:LANUM+1,m(2*SECT),SECT),Y(1:LANUM+1,m(2*SECT),SECT), ...
        X(1:LANUM+1,m(2*SECT),SECT),Y(1:LANUM+1,m(2*SECT),SECT),'o')
    axis equal
end

```

23. ViewLR.m

```

function ViewLR

load InitData
load PQData

if AirfoilType ~= 3
    if NumUpperSeg == 1
        UANUM(1) = str2num(get(findobj(gcf,'Tag','UANUM1'),'String'));
        UAP(1) = str2num(get(findobj(gcf,'Tag','UAP1'),'String'));
        UAQ(1) = str2num(get(findobj(gcf,'Tag','UAQ1'),'String'));
        Dir(1) = str2num(get(findobj(gcf,'Tag','Dir1'),'String'));
        EdgeUpperSeg(1) = 1.0;
    elseif NumUpperSeg == 2
        UANUM(1) = str2num(get(findobj(gcf,'Tag','UANUM1'),'String'));
        UAP(1) = str2num(get(findobj(gcf,'Tag','UAP1'),'String'));
        UAQ(1) = str2num(get(findobj(gcf,'Tag','UAQ1'),'String'));
        Dir(1) = str2num(get(findobj(gcf,'Tag','Dir1'),'String'));
        EdgeUpperSeg(1) = str2num(get(findobj(gcf,'Tag', ...
            'EdgeUpperSeg1'),'String'));
        UANUM(2) = str2num(get(findobj(gcf,'Tag','UANUM2'),'String'));
        UAP(2) = str2num(get(findobj(gcf,'Tag','UAP2'),'String'));
    end
end

```

```

    UAQ(2) = str2num(get(findobj(gcbf, 'Tag', 'UAQ2'), 'String'));
    Dir(2) = str2num(get(findobj(gcbf, 'Tag', 'Dir2'), 'String'));
    EdgeUpperSeg(2) = 1.0;
elseif NumUpperSeg == 3
    UANUM(1) = str2num(get(findobj(gcbf, 'Tag', 'UANUM1'), 'String'));
    UAP(1) = str2num(get(findobj(gcbf, 'Tag', 'UAP1'), 'String'));
    UAQ(1) = str2num(get(findobj(gcbf, 'Tag', 'UAQ1'), 'String'));
    Dir(1) = str2num(get(findobj(gcbf, 'Tag', 'Dir1'), 'String'));
    EdgeUpperSeg(1) = str2num(get(findobj(gcbf, 'Tag', ...
        'EdgeUpperSeg1'), 'String'));
    UANUM(2) = str2num(get(findobj(gcbf, 'Tag', 'UANUM2'), 'String'));
    UAP(2) = str2num(get(findobj(gcbf, 'Tag', 'UAP2'), 'String'));
    UAQ(2) = str2num(get(findobj(gcbf, 'Tag', 'UAQ2'), 'String'));
    Dir(2) = str2num(get(findobj(gcbf, 'Tag', 'Dir2'), 'String'));
    EdgeUpperSeg(2) = str2num(get(findobj(gcbf, 'Tag', ...
        'EdgeUpperSeg2'), 'String'));
    UANUM(3) = str2num(get(findobj(gcbf, 'Tag', 'UANUM3'), 'String'));
    UAP(3) = str2num(get(findobj(gcbf, 'Tag', 'UAP3'), 'String'));
    UAQ(3) = str2num(get(findobj(gcbf, 'Tag', 'UAQ3'), 'String'));
    Dir(3) = str2num(get(findobj(gcbf, 'Tag', 'Dir3'), 'String'));
    EdgeUpperSeg(3) = 1.0;
else
    error('NumUpperSeg ~= 1, 2, or 3')
end

save PQData UANUM UAP UAQ Dir EdgeUpperSeg -APPEND

end

AWP = str2num(get(findobj(gcbf, 'Tag', 'AWP'), 'String'));
AWQ = str2num(get(findobj(gcbf, 'Tag', 'AWQ'), 'String'));
LRP = str2num(get(findobj(gcbf, 'Tag', 'LRP'), 'String'));
LRQ = str2num(get(findobj(gcbf, 'Tag', 'LRQ'), 'String'));

save PQData AWP AWQ LRP LRQ -APPEND

LowerRightVerticalInit(0)

```

24. ViewRigid.m

```

function ViewRigid

load InitData
load PQData

if AirfoilType ~= 3
    if NumUpperSeg == 1
        UANUM(1) = str2num(get(findobj(gcbf, 'Tag', 'UANUM1'), 'String'));
        UAP(1) = str2num(get(findobj(gcbf, 'Tag', 'UAP1'), 'String'));
        UAQ(1) = str2num(get(findobj(gcbf, 'Tag', 'UAQ1'), 'String'));
        Dir(1) = str2num(get(findobj(gcbf, 'Tag', 'Dir1'), 'String'));
        EdgeUpperSeg(1) = 1.0;
    elseif NumUpperSeg == 2
        UANUM(1) = str2num(get(findobj(gcbf, 'Tag', 'UANUM1'), 'String'));

```

```

    UAP(1) = str2num(get(findobj(gcf,'Tag','UAP1'),'String'));
    UAQ(1) = str2num(get(findobj(gcf,'Tag','UAQ1'),'String'));
    Dir(1) = str2num(get(findobj(gcf,'Tag','Dir1'),'String'));
    EdgeUpperSeg(1) = str2num(get(findobj(gcf,'Tag', ...
        'EdgeUpperSeg1'),'String'));
    UANUM(2) = str2num(get(findobj(gcf,'Tag','UANUM2'),'String'));
    UAP(2) = str2num(get(findobj(gcf,'Tag','UAP2'),'String'));
    UAQ(2) = str2num(get(findobj(gcf,'Tag','UAQ2'),'String'));
    Dir(2) = str2num(get(findobj(gcf,'Tag','Dir2'),'String'));
    EdgeUpperSeg(2) = 1.0;
elseif NumUpperSeg == 3
    UANUM(1) = str2num(get(findobj(gcf,'Tag','UANUM1'),'String'));
    UAP(1) = str2num(get(findobj(gcf,'Tag','UAP1'),'String'));
    UAQ(1) = str2num(get(findobj(gcf,'Tag','UAQ1'),'String'));
    Dir(1) = str2num(get(findobj(gcf,'Tag','Dir1'),'String'));
    EdgeUpperSeg(1) = str2num(get(findobj(gcf,'Tag', ...
        'EdgeUpperSeg1'),'String'));
    UANUM(2) = str2num(get(findobj(gcf,'Tag','UANUM2'),'String'));
    UAP(2) = str2num(get(findobj(gcf,'Tag','UAP2'),'String'));
    UAQ(2) = str2num(get(findobj(gcf,'Tag','UAQ2'),'String'));
    Dir(2) = str2num(get(findobj(gcf,'Tag','Dir2'),'String'));
    EdgeUpperSeg(2) = str2num(get(findobj(gcf,'Tag', ...
        'EdgeUpperSeg2'),'String'));
    UANUM(3) = str2num(get(findobj(gcf,'Tag','UANUM3'),'String'));
    UAP(3) = str2num(get(findobj(gcf,'Tag','UAP3'),'String'));
    UAQ(3) = str2num(get(findobj(gcf,'Tag','UAQ3'),'String'));
    Dir(3) = str2num(get(findobj(gcf,'Tag','Dir3'),'String'));
    EdgeUpperSeg(3) = 1.0;
else
    error('NumUpperSeg ~= 1, 2, or 3')
end

if NumLowerSeg == 1
    LANUM(1) = str2num(get(findobj(gcf,'Tag','LANUM1'),'String'));
    LAP(1) = str2num(get(findobj(gcf,'Tag','LAP1'),'String'));
    LAQ(1) = str2num(get(findobj(gcf,'Tag','LAQ1'),'String'));
    LDir(1) = str2num(get(findobj(gcf,'Tag','LDir1'),'String'));
    EdgeLowerSeg(1) = 1.0;
elseif NumLowerSeg == 2
    LANUM(1) = str2num(get(findobj(gcf,'Tag','LANUM1'),'String'));
    LAP(1) = str2num(get(findobj(gcf,'Tag','LAP1'),'String'));
    LAQ(1) = str2num(get(findobj(gcf,'Tag','LAQ1'),'String'));
    LDir(1) = str2num(get(findobj(gcf,'Tag','LDir1'),'String'));
    EdgeLowerSeg(1) = str2num(get(findobj(gcf,'Tag', ...
        'EdgeLowerSeg1'),'String'));
    LANUM(2) = str2num(get(findobj(gcf,'Tag','LANUM2'),'String'));
    LAP(2) = str2num(get(findobj(gcf,'Tag','LAP2'),'String'));
    LAQ(2) = str2num(get(findobj(gcf,'Tag','LAQ2'),'String'));
    LDir(2) = str2num(get(findobj(gcf,'Tag','LDir2'),'String'));
    EdgeLowerSeg(2) = 1.0;
elseif NumLowerSeg == 3
    LANUM(1) = str2num(get(findobj(gcf,'Tag','LANUM1'),'String'));
    LAP(1) = str2num(get(findobj(gcf,'Tag','LAP1'),'String'));
    LAQ(1) = str2num(get(findobj(gcf,'Tag','LAQ1'),'String'));

```

```

    LDir(1) = str2num(get(findobj(gcf,'Tag','LDir1'),'String'));
    EdgeLowerSeg(1) = str2num(get(findobj(gcf,'Tag', ...
        'EdgeLowerSeg1'),'String'));
    LANUM(2) = str2num(get(findobj(gcf,'Tag','LANUM2'),'String'));
    LAP(2) = str2num(get(findobj(gcf,'Tag','LAP2'),'String'));
    LAQ(2) = str2num(get(findobj(gcf,'Tag','LAQ2'),'String'));
    LDir(2) = str2num(get(findobj(gcf,'Tag','LDir2'),'String'));
    EdgeLowerSeg(2) = str2num(get(findobj(gcf,'Tag', ...
        'EdgeLowerSeg2'),'String'));
    LANUM(3) = str2num(get(findobj(gcf,'Tag','LANUM3'),'String'));
    LAP(3) = str2num(get(findobj(gcf,'Tag','LAP3'),'String'));
    LAQ(3) = str2num(get(findobj(gcf,'Tag','LAQ3'),'String'));
    LDir(3) = str2num(get(findobj(gcf,'Tag','LDir3'),'String'));
    EdgeLowerSeg(3) = 1.0;
else
    error('NumLowerSeg ~= 1, 2, or 3')
end

save PQData UANUM UAP UAQ Dir EdgeUpperSeg ...
    LANUM LAP LAQ LDir EdgeLowerSeg -APPEND

end

AWP = str2num(get(findobj(gcf,'Tag','AWP'),'String'));
AWQ = str2num(get(findobj(gcf,'Tag','AWQ'),'String'));
LSP = str2num(get(findobj(gcf,'Tag','LSP'),'String'));
LSQ = str2num(get(findobj(gcf,'Tag','LSQ'),'String'));
URP = str2num(get(findobj(gcf,'Tag','URP'),'String'));
URQ = str2num(get(findobj(gcf,'Tag','URQ'),'String'));
LRP = str2num(get(findobj(gcf,'Tag','LRP'),'String'));
LRQ = str2num(get(findobj(gcf,'Tag','LRQ'),'String'));

save PQData AWP AWQ LSP LSQ URP URQ LRP LRQ -APPEND

if AirfoilType ~= 3
    for i = 1:NumUpperSeg
        if AirfoilType == 1
            UpperAirfoil(1,i)
        elseif AirfoilType == 2
            UpperAirfoilRTE(1,i)
        end
    end
end

AirfoilWakeInit(1)
LeftSurfaceInit(1)
UpperRightVerticalInit(1)
LowerRightVerticalInit(1)

if AirfoilType ~= 3
    for i = 1:NumLowerSeg
        if AirfoilType == 1
            LowerAirfoil(1,i)
        elseif AirfoilType == 2
            LowerAirfoilRTE(1,i)

```

```

        end
    end
end

SECT=2;

load Intermediate

if m(2*SECT-1) - NUM ~= m(2*(SECT-1)-1) - NUM2
    error('The correct number of points must be allotted for the lower
        surface of the airfoil wake')
end

Duplicate(0)
UpperRigid
LowerRigid
PlotRigid

```

25. ViewTrailingEdgeLine.m

```

function ViewTrailingEdgeLine

load PQData
load InitData

if AirfoilType ~= 3
    if NumUpperSeg == 1
        UANUM(1) = str2num(get(findobj(gcf,'Tag','UANUM1'),'String'));
        UAP(1) = str2num(get(findobj(gcf,'Tag','UAP1'),'String'));
        UAQ(1) = str2num(get(findobj(gcf,'Tag','UAQ1'),'String'));
        Dir(1) = str2num(get(findobj(gcf,'Tag','Dir1'),'String'));
        EdgeUpperSeg(1) = 1.0;
    elseif NumUpperSeg == 2
        UANUM(1) = str2num(get(findobj(gcf,'Tag','UANUM1'),'String'));
        UAP(1) = str2num(get(findobj(gcf,'Tag','UAP1'),'String'));
        UAQ(1) = str2num(get(findobj(gcf,'Tag','UAQ1'),'String'));
        Dir(1) = str2num(get(findobj(gcf,'Tag','Dir1'),'String'));
        EdgeUpperSeg(1) = str2num(get(findobj(gcf,'Tag', ...
            'EdgeUpperSeg1'),'String'));
        UANUM(2) = str2num(get(findobj(gcf,'Tag','UANUM2'),'String'));
        UAP(2) = str2num(get(findobj(gcf,'Tag','UAP2'),'String'));
        UAQ(2) = str2num(get(findobj(gcf,'Tag','UAQ2'),'String'));
        Dir(2) = str2num(get(findobj(gcf,'Tag','Dir2'),'String'));
        EdgeUpperSeg(2) = 1.0;
    elseif NumUpperSeg == 3
        UANUM(1) = str2num(get(findobj(gcf,'Tag','UANUM1'),'String'));
        UAP(1) = str2num(get(findobj(gcf,'Tag','UAP1'),'String'));
        UAQ(1) = str2num(get(findobj(gcf,'Tag','UAQ1'),'String'));
        Dir(1) = str2num(get(findobj(gcf,'Tag','Dir1'),'String'));
        EdgeUpperSeg(1) = str2num(get(findobj(gcf,'Tag', ...
            'EdgeUpperSeg1'),'String'));
        UANUM(2) = str2num(get(findobj(gcf,'Tag','UANUM2'),'String'));
        UAP(2) = str2num(get(findobj(gcf,'Tag','UAP2'),'String'));
        UAQ(2) = str2num(get(findobj(gcf,'Tag','UAQ2'),'String'));
    end
end

```

```

Dir(2) = str2num(get(findobj(gcf,'Tag','Dir2'),'String'));
EdgeUpperSeg(2) = str2num(get(findobj(gcf,'Tag', ...
    'EdgeUpperSeg2'),'String'));
UANUM(3) = str2num(get(findobj(gcf,'Tag','UANUM3'),'String'));
UAP(3) = str2num(get(findobj(gcf,'Tag','UAP3'),'String'));
UAQ(3) = str2num(get(findobj(gcf,'Tag','UAQ3'),'String'));
Dir(3) = str2num(get(findobj(gcf,'Tag','Dir3'),'String'));
EdgeUpperSeg(3) = 1.0;
else
    error('NumUpperSeg ~= 1, 2, or 3')
end

save PQData UANUM UAP UAQ Dir EdgeUpperSeg -APPEND

end

AWP = str2num(get(findobj(gcf,'Tag','AWP'),'String'));
AWQ = str2num(get(findobj(gcf,'Tag','AWQ'),'String'));

save PQData AWP AWQ -APPEND

AirfoilWakeInit(0)

```

26. ViewUpperAirfoil.m

```

function ViewUpperAirfoil(Seg)

load PQData
load InitData

if AirfoilType ~= 3
    if NumUpperSeg == 1
        UANUM(1) = str2num(get(findobj(gcf,'Tag','UANUM1'),'String'));
        UAP(1) = str2num(get(findobj(gcf,'Tag','UAP1'),'String'));
        UAQ(1) = str2num(get(findobj(gcf,'Tag','UAQ1'),'String'));
        Dir(1) = str2num(get(findobj(gcf,'Tag','Dir1'),'String'));
        EdgeUpperSeg(1) = 1.0;
    elseif NumUpperSeg == 2
        UANUM(1) = str2num(get(findobj(gcf,'Tag','UANUM1'),'String'));
        UAP(1) = str2num(get(findobj(gcf,'Tag','UAP1'),'String'));
        UAQ(1) = str2num(get(findobj(gcf,'Tag','UAQ1'),'String'));
        Dir(1) = str2num(get(findobj(gcf,'Tag','Dir1'),'String'));
        EdgeUpperSeg(1) = str2num(get(findobj(gcf,'Tag', ...
            'EdgeUpperSeg1'),'String'));
        UANUM(2) = str2num(get(findobj(gcf,'Tag','UANUM2'),'String'));
        UAP(2) = str2num(get(findobj(gcf,'Tag','UAP2'),'String'));
        UAQ(2) = str2num(get(findobj(gcf,'Tag','UAQ2'),'String'));
        Dir(2) = str2num(get(findobj(gcf,'Tag','Dir2'),'String'));
        EdgeUpperSeg(2) = 1.0;
    elseif NumUpperSeg == 3
        UANUM(1) = str2num(get(findobj(gcf,'Tag','UANUM1'),'String'));
        UAP(1) = str2num(get(findobj(gcf,'Tag','UAP1'),'String'));
        UAQ(1) = str2num(get(findobj(gcf,'Tag','UAQ1'),'String'));
        Dir(1) = str2num(get(findobj(gcf,'Tag','Dir1'),'String'));
    end
end

```

```

EdgeUpperSeg(1) = str2num(get(findobj(gcf,'Tag', ...
    'EdgeUpperSeg1'),'String'));
UANUM(2) = str2num(get(findobj(gcf,'Tag','UANUM2'),'String'));
UAP(2) = str2num(get(findobj(gcf,'Tag','UAP2'),'String'));
UAQ(2) = str2num(get(findobj(gcf,'Tag','UAQ2'),'String'));
Dir(2) = str2num(get(findobj(gcf,'Tag','Dir2'),'String'));
EdgeUpperSeg(2) = str2num(get(findobj(gcf,'Tag', ...
    'EdgeUpperSeg2'),'String'));
UANUM(3) = str2num(get(findobj(gcf,'Tag','UANUM3'),'String'));
UAP(3) = str2num(get(findobj(gcf,'Tag','UAP3'),'String'));
UAQ(3) = str2num(get(findobj(gcf,'Tag','UAQ3'),'String'));
Dir(3) = str2num(get(findobj(gcf,'Tag','Dir3'),'String'));
EdgeUpperSeg(3) = 1.0;
else
    error('NumUpperSeg ~= 1, 2, or 3')
end

save PQData UANUM UAP UAQ Dir EdgeUpperSeg -APPEND

end

if AirfoilType == 1
    UpperAirfoil(0,Seg)
elseif AirfoilType == 2
    UpperAirfoilRTE(0,Seg)
else
    load Results
    SECT = 1;
    figure
    plot(X(1:UANUM+1,1,SECT),Y(1:UANUM+1,1,SECT), ...
        X(1:UANUM+1,1,SECT), Y(1:UANUM+1,1,SECT),'o')
    axis equal
end

```

27. ViewUR.m

```

function ViewUR

load PQData
load InitData

if AirfoilType ~= 3
    if NumUpperSeg == 1
        UANUM(1) = str2num(get(findobj(gcf,'Tag','UANUM1'),'String'));
        UAP(1) = str2num(get(findobj(gcf,'Tag','UAP1'),'String'));
        UAQ(1) = str2num(get(findobj(gcf,'Tag','UAQ1'),'String'));
        Dir(1) = str2num(get(findobj(gcf,'Tag','Dir1'),'String'));
        EdgeUpperSeg(1) = 1.0;
    elseif NumUpperSeg == 2
        UANUM(1) = str2num(get(findobj(gcf,'Tag','UANUM1'),'String'));
        UAP(1) = str2num(get(findobj(gcf,'Tag','UAP1'),'String'));
        UAQ(1) = str2num(get(findobj(gcf,'Tag','UAQ1'),'String'));
        Dir(1) = str2num(get(findobj(gcf,'Tag','Dir1'),'String'));
        EdgeUpperSeg(1) = str2num(get(findobj(gcf,'Tag', ...

```

```

        'EdgeUpperSeg1'),'String'));
UANUM(2) = str2num(get(findobj(gcf,'Tag','UANUM2'),'String'));
UAP(2) = str2num(get(findobj(gcf,'Tag','UAP2'),'String'));
UAQ(2) = str2num(get(findobj(gcf,'Tag','UAQ2'),'String'));
Dir(2) = str2num(get(findobj(gcf,'Tag','Dir2'),'String'));
EdgeUpperSeg(2) = 1.0;
elseif NumUpperSeg == 3
    UANUM(1) = str2num(get(findobj(gcf,'Tag','UANUM1'),'String'));
    UAP(1) = str2num(get(findobj(gcf,'Tag','UAP1'),'String'));
    UAQ(1) = str2num(get(findobj(gcf,'Tag','UAQ1'),'String'));
    Dir(1) = str2num(get(findobj(gcf,'Tag','Dir1'),'String'));
    EdgeUpperSeg(1) = str2num(get(findobj(gcf,'Tag', ...
        'EdgeUpperSeg1'),'String'));
    UANUM(2) = str2num(get(findobj(gcf,'Tag','UANUM2'),'String'));
    UAP(2) = str2num(get(findobj(gcf,'Tag','UAP2'),'String'));
    UAQ(2) = str2num(get(findobj(gcf,'Tag','UAQ2'),'String'));
    Dir(2) = str2num(get(findobj(gcf,'Tag','Dir2'),'String'));
    EdgeUpperSeg(2) = str2num(get(findobj(gcf,'Tag', ...
        'EdgeUpperSeg2'),'String'));
    UANUM(3) = str2num(get(findobj(gcf,'Tag','UANUM3'),'String'));
    UAP(3) = str2num(get(findobj(gcf,'Tag','UAP3'),'String'));
    UAQ(3) = str2num(get(findobj(gcf,'Tag','UAQ3'),'String'));
    Dir(3) = str2num(get(findobj(gcf,'Tag','Dir3'),'String'));
    EdgeUpperSeg(3) = 1.0;
else
    error('NumUpperSeg ~= 1, 2, or 3')
end

save PQData UANUM UAP UAQ Dir EdgeUpperSeg -APPEND

end

AWP = str2num(get(findobj(gcf,'Tag','AWP'),'String'));
AWQ = str2num(get(findobj(gcf,'Tag','AWQ'),'String'));
URP = str2num(get(findobj(gcf,'Tag','URP'),'String'));
URQ = str2num(get(findobj(gcf,'Tag','URQ'),'String'));

save PQData AWP AWQ URP URQ -APPEND

UpperRightVerticalInit(0)

```

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF REFERENCES

Fletcher, C.A.J. *Computational Techniques for Fluid Dynamics*, Volume II. Berlin: Springer-Verlag, 1991.

THIS PAGE INTENTIONALLY LEFT BLANK

INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center
Ft. Belvoir, Virginia
2. Dudley Knox Library
Naval Postgraduate School
Monterey, California
3. ENS Natalie Chouinard, USNR
USS Seattle (AOE-3)
Norfolk, Virginia
4. Max Platzer
Naval Postgraduate School
Monterey, California
5. Kevin Jones
Naval Postgraduate School
Monterey, California
6. Isaac Kaminer
Naval Postgraduate School
Monterey, California
7. Russ Duren
Naval Postgraduate School
Monterey, California
8. Beny Neta
Naval Postgraduate School
Monterey, California
9. CDR Mark Couch, USN
Naval Postgraduate School
Monterey, California
9. CAPT Scott Stewart, USN
Air-to-Air Missile PMA
Patuxent River, Maryland

10. Martin W. Verville
Wausau, Wisconsin