Minimum Pressure Envelope Cavitation Analysis Using Two-Dimensional Panel Method

by

Christopher J. Peterson

B.S. Mechanical Engineering
University of Illinois (1998)

Master of Engineering Management
Old Dominion University (2006)

Submitted to the Department of Mechanical Engineering
in Partial Fulfillment of the Requirements for the Degrees of

Master of Science in Naval Architecture and Marine Engineering
and
Master of Science in Mechanical Engineering

at the
Massachusetts Institute of Technology
June 2008

Signature of Author_____
Department of Mechanical Engineering
May 9, 2008

Certified by_____
Patrick Keenan
Professor of Naval Architecture
Thesis Supervisor

Certified by_____
Richard W. Kimball
Thesis Supervisor

Accepted by_____
Lallit Anand
Professor of Mechanical Engineering
Chairman, Departmental Committee on Graduate Students

Minimum Pressure Envelope Cavitation Analysis Using Two-Dimensional Panel Method

by

Christopher J. Peterson

Submitted to the Department of Mechanical Engineering on May 9, 2007 in
Partial Fulfillment of the Requirements for the Degrees of
Master of Science in Naval Architecture and Marine Engineering
and
Master of Science in Mechanical Engineering

ABSTRACT

An analysis tool for calculating minimum pressure envelopes was developed using XFOIL. This thesis presents MATLAB® executables that interface with a modified version of XFOIL for determining the minimum pressure of a foil operating in an inviscid fluid. The code creates minimum pressure envelopes, similar to those published by Brockett (1965). XFOIL, developed by Mark Drela in 1986, is a design system for Low Reynolds Number Airfoils that combines the speed and accuracy of high-order panel methods with fully-coupled viscous/inviscid interaction. XFOIL was altered such that it reads in command line arguments that provide operating instructions, rather than operator interaction via menu options. In addition, all screen output and plotting functions were removed. These modifications removed XFOIL's user interface, and created a "black box" version of XFOIL that would perform the desired calculations and write the output to a file. These modifications allow rapid execution and interface by an external program, such as MATLAB®. In addition, XFOIL's algorithms provide a significant improvement in the accuracy of minimum pressure prediction over the method published by Brockett.

Development of the modified XFOIL and MATLAB® interface contained in this thesis is intended for future interface with *Open-source Propeller Design and Analysis Program* (OpenProp). OpenProp is an open source MATLAB®-based suite of propeller design tools. Currently, OpenProp performs parametric analysis and single propeller design, but does not perform cavitation analysis. Minimum pressure envelopes provide the propeller designer information about operating conditions encountered by propellers. The code developed in this thesis allows the designer to rapidly assess cavitation conditions while in the design phase, and make modifications to propeller blade design in order to optimize cavitation performance. A methodology for design is discussed outlining future integration with OpenProp.

Thesis Supervisor: Prof. Patrick Keenan
Title: Professor of Naval Architecture

Thesis Supervisor: Richard W. Kimball

Table of Contents

Table of Contents 3
List of Figures 4
List of Tables 4
1 Introduction 5
2 Conformal Transformations 8
2.1 History 8
2.2 Use of Conformal Transformations 8
3 Brockett's Analysis 12
3.1 Introduction to Brockett's Analysis 12
3.2 User Input to MATLAB Version of Brockett Code 12
3.3 Output from `Brockett.m` 13
3.4 Brockett Analysis Results 15
4 Introduction to XFOIL 18
4.1 XFOIL Functionality 18
4.2 XFOIL Formulation Summary 18
4.3 Adaptation of XFOIL 19
4.3.1 Executing XFOIL 19
4.4 Comparison of XFOIL Calculated Pressure Distributions 21
5 Minimum Pressure Envelope Analysis 24
5.1 Background 24
5.2 Description of Minimum Pressure Envelope Generation 24
5.2.1 Foil Shape Generation 24
5.2.2 User Specifications and Output from XBucket.m 26
5.3 Comparison of Brockett's Method to XFOIL Results 29
6 OpenProp Implementation Approach 32
6.1 Analysis of Existing Foils 33
6.2 Geometric Design to Prevent Cavitation 34
7 Conclusion 40
7.1 Recommendations for Future Work 40
7.1.1 Viscous Calculations 40
7.1.2 OpenProp Integration 41
Bibliography 42
Appendix A: MATLAB Script for Conformal Transformation of Karman-Trefftz Foil. 43
Appendix B: MATLAB Code of Brockett's Work (`Brockett.m`) 47
Appendix C: `Brockett.m` Variable Descriptions 62
Appendix D: Sample Input Scripts for `Brockett.m`. 65
Appendix E: `Brockett.m` Sample Output 69
Appendix F: Modified XFOIL User Guide 73
Appendix G: Instruction for compiling modified XFOIL Code 82
Appendix H: MATLAB Files for Calculation of Minimum Pressure Envelopes 89
Appendix I: Meanline and Camber Data File Format 98

List of Figures

Figure 1: Potential Flow Solution for 2-D Cylinder 9

Figure 2: Karman-Trefftz Foil and Streamlines 10

Figure 3: Karman-Trefftz Foil Pressure Distribution. 11

Figure 4: Comparison of Brockett Method to Exact Solution, $\alpha = 0°$ 16

Figure 5: Comparison of Brockett Method to Exact Solution, $\alpha = 5°$ 16

Figure 6: Comparison of Brockett Method to Exact Solution, $\alpha = 10°$ 17

Figure 7: Comparison of Brockett Method to Exact Solution, $\alpha = -5°$ 17

Figure 8: Comparison of XFOIL and Brockett Method to Exact Solution, $\alpha = 0°$ 21

Figure 9: Comparison of XFOIL and Brockett Method to Exact Solution, $\alpha = 5°$ 22

Figure 10: Comparison of XFOIL and Brockett Method to Exact Solution, $\alpha = 10°$ 22

Figure 11: Comparison of XFOIL and Brockett Method to Exact Solution, $\alpha = -5°$ 23

Figure 12: Minimum Pressure Envelopes for NACA 66 Section (TMB Mod. Nose and Tail) with Zero Camber at Various Thicknesses 26

Figure 13: Minimum Pressure Envelopes for NACA 66 Section (TMB Modified) with the NACA a=0.8 Camberline, Having a Maximum Camber Ratio of 0.04 at Various Thicknesses. 27

Figure 14: Sample Viscous and Inviscid Minimum Pressure Envelopes Calculated by XFOIL (Reynolds Number = $1*10^6$, 100 Maximum Iterations) 28

Figure 15: Sample Viscous and Inviscid Minimum Pressure Envelopes Calculated by XFOIL (Reynolds Number = $1*10^6$, 100 Maximum Iterations, 70 Panels) 29

Figure 16: Minimum Pressure Envelope Comparison 30

Figure 17: Example Flowchart for using Minimum Pressure Envelopes for Foil Design 35

Figure 18: Minimum Pressure Envelopes to Find Optimum Thickness 36

Figure 19: Minimum Pressure Envelopes to Find Envelope Width and Cavitation Margin 38

List of Tables

Table 1: Comparison of Brockett's published data to MATLAB Version of Calculations 14

Table 2: Comparison of Brockett's Method of Calculation of Minimum Pressure Coefficient to Exact Solution for a Karman-Trefftz Foil 15

1 Introduction

The study of propeller cavitation and its inception is an important aspect of propeller design. In order to accurately predict cavitation inception, it is necessary to be able to determine the actual pressure distribution in the fluid. By comparing the pressure coefficient to the local cavitation number, an estimate of the local cavitation conditions may be made. This is often accomplished by determining the fluid velocity distribution in the fluid, and then using the velocity to calculate local pressure conditions. Specifically, the pressure distribution is desired along the upper and lower surfaces of the foil in order to determine lift, drag, moment, and cavitation inception.

An early approach to this problem was to assume that the working fluid was inviscid. This assumption allowed the use of potential flow theory to calculate velocity as a function of location within the fluid. However, this method was limited to very simple shapes, such as a two-dimensional cylinder. Potential theory lacked the ability to directly calculate the fluid velocities around complex geometries such as foil surfaces.

Conformal mapping provided a method by which the exact velocity distribution could be calculated for certain types of foil shapes. However, exact conformal transformations for all foil shapes are not possible. Although various transformations have been introduced, this project uses the Karman-Trefftz foil as a comparison for numerical approaches. To further extend the use of conformal transformations, numerical approaches were developed to approximate the mapping function for foils of arbitrary shape.

An improved approach to obtain an accurate estimate of the actual pressure distribution on two-dimensional foils of arbitrary shape was developed by Brockett [1]. Brockett's work was based on the work of Moriya [2] which is an approximate conformal transformation of the circle to an airfoil profile and gives equations for the velocity distribution. Brockett published a FORTRAN computer program that would

accept foil ordinate input, in various formats, and calculate the velocity and pressure distributions at a specified angle of attack or lift coefficient.

The purpose of this report is to present a modified approach to the work of Brockett, and present an improved method for computing the minimum pressure envelopes of a foil. As a reference, Brockett's method of calculating the minimum pressure distribution was programmed into MATLAB. Sample input provided by Brockett was used to verify accuracy of the MATLAB version of Brockett's work.

A modified approach using the two-dimensional panel method of XFOIL is presented. Rather than using the approximate conformal transformation method presented by Brockett, a modified version of XFOIL was used to perform the calculations for pressure distribution. The modified XFOIL executable removed all interactive user interfaces, including menu driven options and interactive screen output. In addition, desired output is saved as a text file, rather than plotting to the screen. The development of the modified XFOIL executable allows the use of an external program, in this case MATLAB, to call XFOIL to perform the desired calculations. The results are then saved as a text file, and may be read in by MATLAB, which conducts the desired analysis and output.

A Karman-Trefftz foil was used as a reference for comparison. The analytic solution for the pressure distribution on the foil was used as the baseline to which the numerical methods were compared. The method presented using XFOIL to conduct calculations is nearly indistinguishable from the analytic solution, a significant improvement over the Brockett method that underestimated the minimum pressure by 28% at a high angle of attack (10°).

The intention of this project was to develop an improved method for computing minimum pressure envelopes for an arbitrary foil shape. In addition, it was developed such that this method would be integrated into the *Open-source Propeller Design and Analysis Program* (OpenProp). OpenProp is an open source MATLAB®-based suite of propeller design tools. OpenProp currently performs

parametric analysis and single propeller design, but does not perform cavitation analysis. The development of the MATLAB code in this project would aid the designer in the rapid design of propellers by providing a quick method to predict cavitation performance of a propeller, and allow the analysis of cavitation performance early in the design process. Conceptual implementation will be discussed later in this report.

2 Conformal Transformations

2.1 History

Prior to the development of the computer, obtaining an accurate solution for the flow around a complex shape was a challenging task. The development of conformal transformations was therefore of great benefit, as it provided an analytic solution for the exact inviscid flow solution to a select number of foil shapes. This method was developed by Joukowski in 1914. Karman and Trefftz then introduced a more general mapping function, which was a special case of the Joukowski transformation. Theodorsen [3] then built upon this work and developed an approximate numerical technique for obtaining the mapping function of an arbitrary foil shape. These developments ultimately led to the work of Brockett, and his development of the design charts published in 1966 [4].

2.2 Use of Conformal Transformations

Although a detailed explanation of conformal transformations is not warranted here, the motivation of this project deserves a brief description of the procedure of conformal transformations, in order to highlight the significant improvements of the work presented. The following is an adaptation of Kerwin's [5] derivation.

Potential flow solution for a two-dimensional cylinder is easily described and understood. It consists of a source-sink dipole, oriented by the direction of the uniform stream. This produces streamlines that define the two-dimensional shape shown in Figure 1.

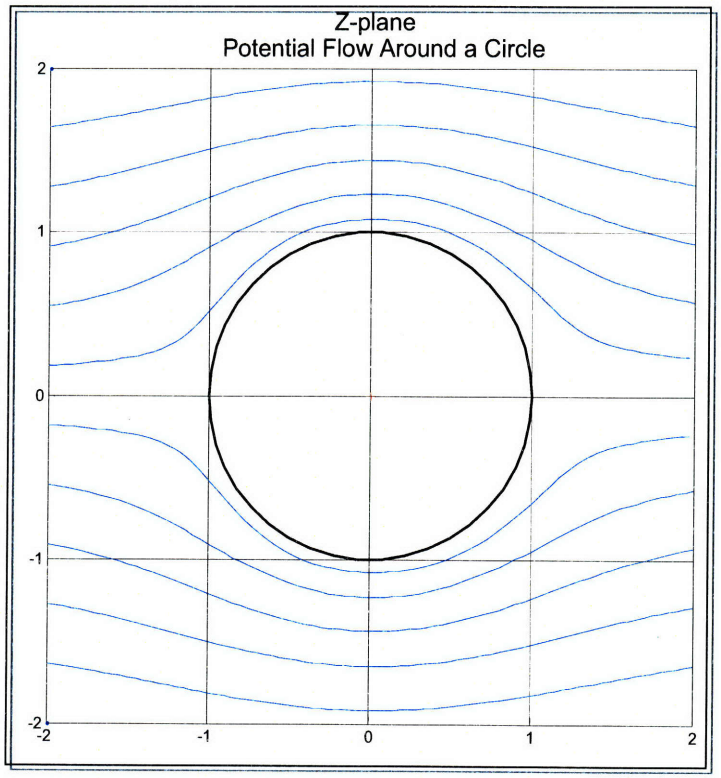


Figure 1: Potential Flow Solution for 2-D Cylinder

The Karman-Trefftz transformation maps a complex point *z*, where z = X + i*Y, from the Z-Plane to a point ζ using equation (2.1) below.

$$\zeta = \frac{\lambda a\left[(z+a)^{\lambda} + (z-a)^{\lambda}\right]}{\left[(z+a)^{\lambda} - (z-a)^{\lambda}\right]} \tag{2.1}$$

In order to evaluate the velocities in the ζ-Plane the derivative dζ /dz must be evaluated using equations (2.2) and (2.3) below.

$$\frac{d\zeta}{dz} = \frac{4\lambda^2 a^2\left[(z-a)^{\lambda-1}(z+a)^{\lambda-1}\right]}{\left[(z+a)^{\lambda} - (z-a)^{\lambda}\right]^2} \tag{2.2}$$

$$\left[u - iv\right]_{\zeta} = \frac{\left[u - iv\right]_{z}}{\frac{d\zeta}{dz}} \tag{2.3}$$

In order to map the circle into the ζ-plane, the Kaman-Trefftz method requires that the circle center be defined by x_C and y_C. Also, the trailing edge angle, τ, is defined in degrees and related by $\lambda = 2 - \tau/180$. Finally, a, which is the X-intercept, is set equal to unity. The foil in Figure 2 below was developed by defining x_C = -0.1 and y_C= 0.15, and τ = 10°. In addition, an angle of attack may be specified, however, in Figure 1 and Figure 2, the angle of attack is zero. Lastly, the circulation is set in order to meet the Kutta condition to ensure smooth flow leaving the trailing edge.

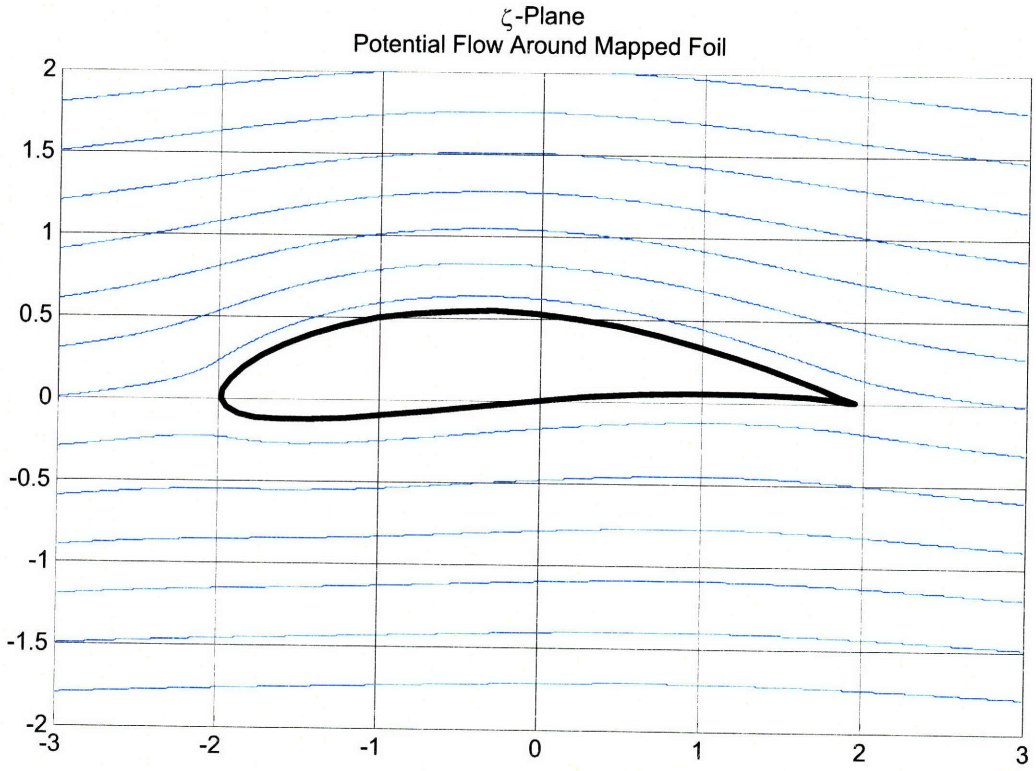


Figure 2: Karman-Trefftz Foil and Streamlines

To evaluate the velocity and calculate the pressure distribution for the ideal fluid flow over the foil surface, the velocities in the ζ-Plane were evaluated using Equation (2.3), and the pressure coefficient, Cp, was calculated using Equation (2.4), where q is the absolute velocity, and U is the free stream velocity.

$$C_p = \frac{p - p_\infty}{\frac{1}{2}\rho U^2} = 1 - \left(\frac{q}{U}\right)^2 \qquad (2.4)$$

Since the Karman-Trefftz foil provides an analytic solution for the potential flow around the mapped foil, it provides an exact solution to which numerical methods may be compared. Figure 3 shows the analytic pressure distribution for the Karman-Trefftz foil shown in Figure 2.

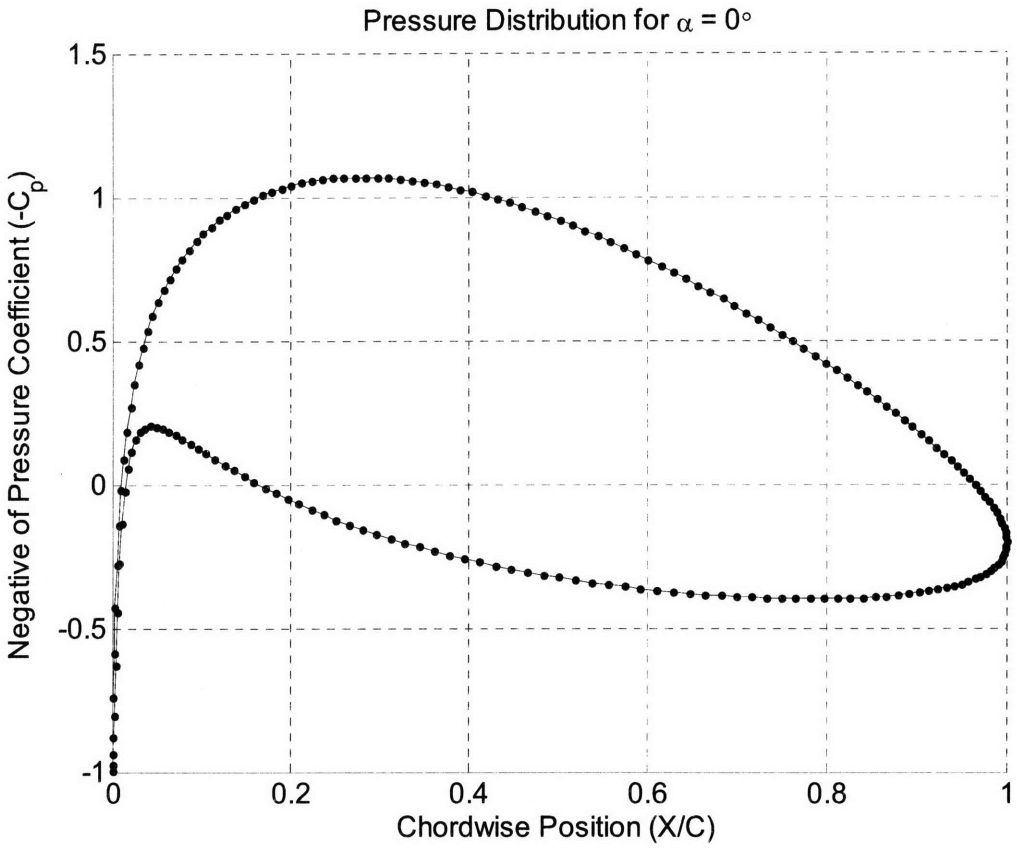


Figure 3: Karman-Trefftz Foil Pressure Distribution.

The analysis above is the result of the MATLAB script contained in Appendix A: MATLAB Script for Conformal Transformation of Karman-Trefftz Foil. This script also generates and exports data for further analysis.

3 Brockett's Analysis

3.1 Introduction to Brockett's Analysis

In reference [1], Brockett developed a computer code to evaluate the steady two-dimensional pressure distribution on arbitrary foils, and presented the results. This code is based on an approximate potential theory suggested by Moriya which is empirically modified in a manner suggested by Pinkerton to give an arbitrary lift for a set incidence, while satisfying the Kutta condition. Interpolation functions for the ordinates were used to reduce the calculations to a straight-forward numerical procedure. Brockett presents a FORTRAN computer program, which, as part of this thesis, was rewritten as a MATLAB script in order to facilitate simple input and output. The original code was left unaltered as much as possible, such that the algorithms were left intact. The only significant changes are the method in which data was fed to the program, and the programming structure as required by MATLAB. Also, MATLAB allows variables to be defined by the user, or data can be read in by MATLAB, rather than using FORTRAN control card format specified by Brockett. This functionality greatly enhances the interface capability of the program by the user, and allows more rapid analysis and comparison of results. The code is presented in Appendix B: MATLAB Code of Brockett's Work, and a description of the main variables used in the program are included in Appendix C: `Brockett.m` Variable Descriptions.

3.2 User Input to MATLAB Version of Brockett Code

FORTRAN used formatted control cards, which allowed the user to input data to be processed by the program. This method of data has been superceded by either direct input by the user or digital data saved as files on the computer. The MATLAB code developed maintains as much original structure as possible, while allowing the operator to specify which data will be used as input, which is accomplished by the use of MATLAB script files. These files allow the user to specify the data to be processed for each of the formats required by Brockett's code. In reference [1], Brockett provides sample input and output of the original code. The first script file in Appendix D: Sample Input Scripts for `Brockett.m.`, `REQD_IN.m`, duplicates the

sample input presented by Brockett, and was used as a validation case for the MATLAB code. Results matched the values presented in Brockett's sample output on pages 70 – 72A of reference [1]. This format of this script file is required when inputting data at the points at the required offset locations as required by `BROCKETT.m`.

Also included in Appendix D: , are `ARB_IN.m`, `KT_IN.m`, and `Brock_IN.m`. The `ARB_IN.m` is a sample file for the format required to input foil ordinates at arbitrary locations. `ARB_IN.m` is written to accept input similar to that presented by Brockett in Figure 4b of reference [1]. The format is for input of foil ordinates at arbitrary stations. The format allows multiple angles of attack to be input, resulting in the calculation of pressure distribution at each angle of attack. This format accepts input, as specified by Brockett, that specifies airfoil ordinates in X, Y format. Data points are required to be entered with the same number of points on the upper and lower surface, starting with the trailing edge along the upper surface to the nose location, continuing along the lower surface to the trailing edge. `KT_in.m` is a script file used to import the data generated by the conformal transformation of Appendix A: , used to compute the pressure distribution predicted by Brockett. `Brock_IN.m` is used to input the geometry for the NACA 66, a = 0.8 (TMB Modified) foil used by Brockett in reference [4].

Brockett's FORTRAN program required very specific format for inputting data. Since the original code was maintained similar to the original structure, several operational variables must be specified for the code to function properly. In addition, input data must be carefully structured in the proper format to be processed correctly. The script files of Appendix D: were used to accomplish the variable definitions required by `Brockett.m`.

3.3 Output from `Brockett.m`

To validate that the MATLAB version of Brockett's code was accurate, a test run was conducted that replicated the sample case included in reference [1]. The `REQD_IN.m` script file was used, and the output of `Brockett.m`, contained in

Appendix E: `Brockett.m` Sample Output, was verified using reference [1]. A summary of the output at sample chord position for the MATLAB version of Brockett's program, and the original data published by Brockett is shown in Table 1. As seen in Table 1, the data agrees within approximately 7 significant figures, which is the number of significant figures expected for the single-precision data type used by FORTRAN. The default for MATLAB is to use double-precision floating-point numbers, which would explain the slight differences between MATLAB calculations, and Brockett's published data.

	Pressure Coefficient		
Chord Positition (x/C)	**Brockett Ref [1]**	**MATLAB Version of Brockett**	**% Difference**
1	1.000000	1.000000	0.00000%
0	-0.495333	-0.495331	0.00040%
Upper Surface			
0.75	-0.099686	-0.099685	0.00100%
0.5	-0.329138	-0.329138	0.00000%
0.25	-0.616285	-0.616284	0.00016%
0.178606	-0.689939	-0.689939	0.00000%
0.030154	-1.226171	-1.226170	0.00008%
Lower Surface			
0.030154	0.540715	0.540716	-0.00018%
0.178606	0.033803	0.033804	-0.00296%
0.25	-0.037634	-0.037633	0.00266%
0.5	-0.038729	-0.038728	0.00258%
0.75	0.042112	0.042113	-0.00237%
Lift Coeff	0.430129	0.430129	0.00000%

Table 1: Comparison of Brockett's published data to MATLAB Version of Calculations

Output from the original and the MATLAB version of Brockett's work include multiple tables consisting of Profile Constants and Pressure Distribution information. In addition to the screen output of the data, the MATLAB version saves the data as text files labeled `Pressure.txt` and `Profile.txt`, which are saved in the "Data" folder. This data is later read into variables by MATLAB, or may be opened by the user.

3.4 Brockett Analysis Results

To compare the pressure distribution calculated by the Brockett method to the exact solution shown in Figure 3, `KT_IN.m` was written to read in the ordinates of the normalized Karman-Trefftz foil generated in Figure 2. The X-Y ordinates are cosine-spaced, and the number of points is specified by the '`out_pts`' variable in `ConfrmlTrans.m`. If `out_pts` is set to 37, this matches the required input location format of Brockett, but is not required. Otherwise, `BROCKETT.m` will accept format as arbitrary location input. Results obtained from the use of a Karman-Trefftz foil shape is shown in Figure 4.

Figure 4 through Figure 7 shows that Brockett's method predicts the general shape of the pressure distribution and gives an estimate of the minimum pressure for the foil, but does not accurately predict the magnitude. In Figure 4, the actual minimum pressure coefficient, $-Cp_{min}$, is 1.192 at x/C = 0.32. Brockett's analysis predicts that $-Cp_{min}$ = 1.079 at x/C = 0.33, which falls short by 9.5%. This error increases as angle of attack increases. Inviscid minimum pressure coefficients predicted by Brockett are compared to analytic results for the Karman-Trefftz foil for various angles of attack in Table 2.

	-Cp_min		
α	Brockett	Analytic	Error
-15	11.0498	10.7114	3.16%
-10	5.5078	5.2178	5.56%
-5	1.8012	1.7059	5.59%
-4	1.3052	1.2482	4.57%
-2	0.932	1.017	-8.36%
0	1.0788	1.1919	-9.49%
2	1.2366	1.3938	-11.28%
4	1.4281	1.6374	-12.78%
6	1.6749	2.0243	-17.26%
8	2.32	3.0368	-23.60%
10	3.1896	4.4451	-28.24%
15	6.8295	9.4474	-27.71%

Table 2: Comparison of Brockett's Method of Calculation of Minimum Pressure Coefficient to Exact Solution for a Karman-Trefftz Foil

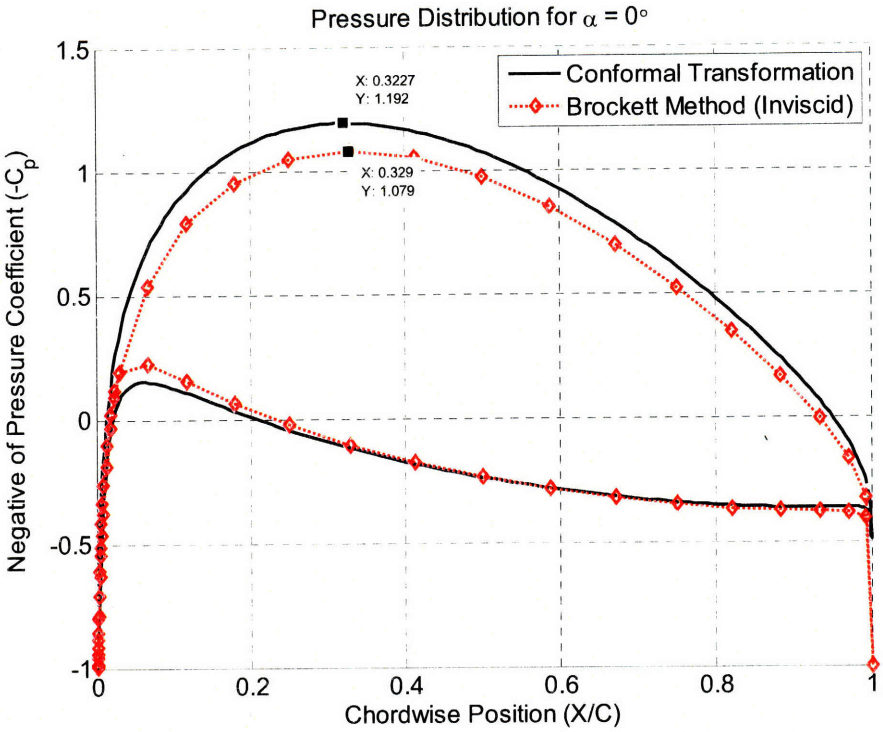


Figure 4: Comparison of Brockett Method to Exact Solution, α = 0°

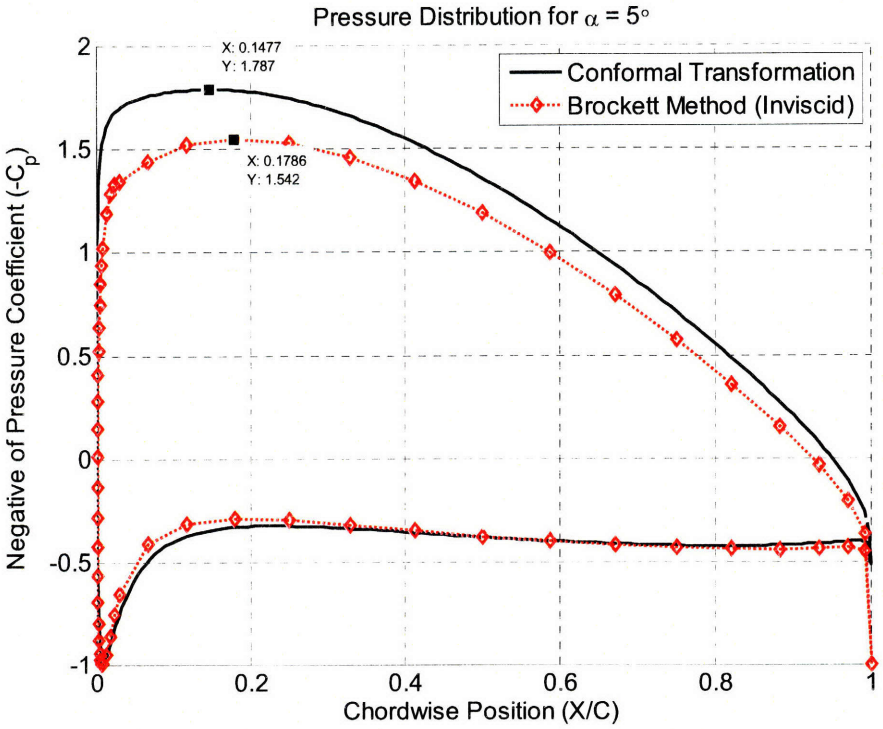


Figure 5: Comparison of Brockett Method to Exact Solution, α = 5°

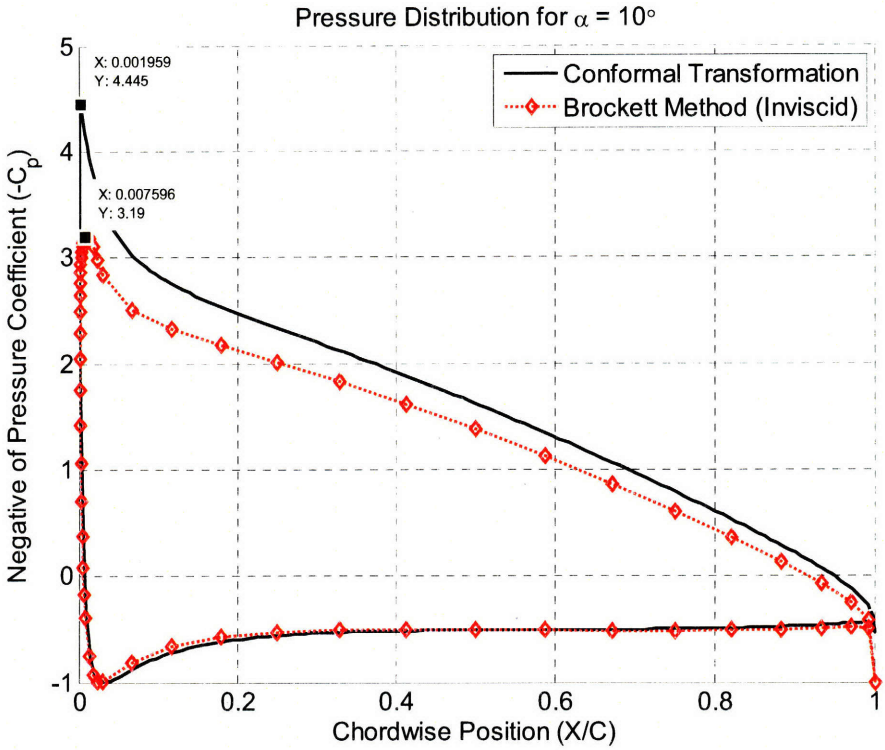


Figure 6: Comparison of Brockett Method to Exact Solution, α = 10°

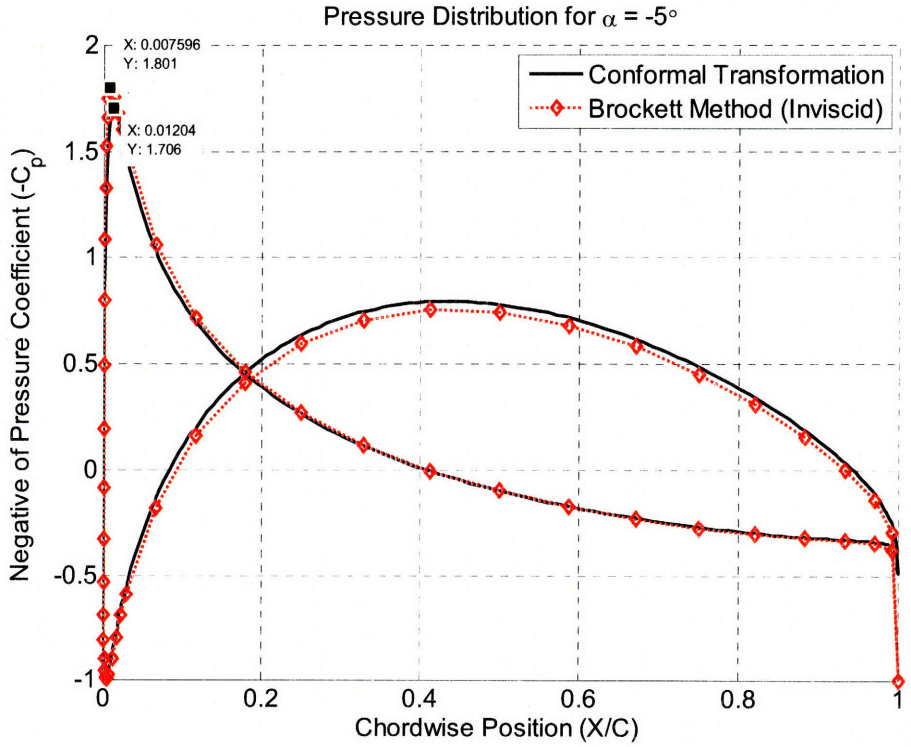


Figure 7: Comparison of Brockett Method to Exact Solution, α = -5°

4 Introduction to XFOIL

4.1 XFOIL Functionality

XFOIL 1.0 was written by Mark Drela in 1986. The main goal was to combine the speed and accuracy of high-order panel methods with the new fully-coupled viscous/inviscid interaction method used in the ISES code developed by Drela and Giles. A fully interactive interface was employed to make it much easier to use than the traditional batch-type CFD codes. Several inverse modes and a geometry manipulator were also incorporated early in XFOIL's development, making it a fairly general airfoil development system [6].

XFOIL is an analysis and design system for Low Reynolds Number Airfoils. XFOIL uses an inviscid linear-vorticity panel method with a Karman-Tsien compressibility correction for direct and mixed-inverse modes. Source distributions are superimposed on the airfoil and wake permitting modeling of viscous layer influence on the potential flow. Both laminar and turbulent layers are treated with an e^9-type amplification formulation determining the transition point. The boundary layer and transition equations are solved simultaneously with the inviscid flow field by a global Newton method [7].

4.2 XFOIL Formulation Summary

Details of XFOIL's formulation are presented in reference [7], and will only be summarized here. XFOIL uses a general inviscid airfoil flow field, constructed by the superposition of a free stream flow, a vortex sheet of strength γ on the airfoil surface, and source sheet strength, σ, on the airfoil surface and wake. The airfoils contour and wake trajectory is discretized into flat panels, with panel nodes on the airfoil and wake. Each airfoil panel has a linear vorticity distribution defined by the node value. Each airfoil and wake panel has a constant source strength, which is later related to viscous layer quantities. Requiring the stream function to be equal to a constant value at each of the nodes on the airfoil surface results in a system of linear equations that could be solved in combination with the Kutta condition.

XFOIL's viscous formulation was not used extensively in this research, and will not be discussed in detail.

4.3 Adaptation of XFOIL

XFOIL has been in use for many years, and has become a highly regarded analysis tool. This project adapts the improved functionally of XFOIL for use in propeller design by using the algorithms contained within XFOIL to conduct the analysis of a given foil. In particular, XFOIL is used to develop minimum pressure envelopes, or cavitation buckets, as presented by Brockett [4].

XFOIL in its current release (XFOIL 6.94) is a menu driven program, which requires interactive user input and manipulation. In addition, XFOIL generates various output plots to allow graphical display and interface by the user. In order to adapt XFOIL's functionality, XFOIL was converted into a "black-box" calculation tool. XFOIL's menu driven functionality was removed by altering the source code such that all desired operational instructions would be input as command line arguments, rather than menu driven items and direct user input. All of XFOIL's plot and screen output utilities were also disabled. By disabling the plot functionality, calculation speed was improved. Also, removal of XFOIL's plot functionality, allowed simpler compilation of the source code, since no graphical interface was required for the operating system. Finally, XFOIL was altered such that any desired output was written to and saved as a data file, which could be read by MATLAB or opened directly by a text editing program.

4.3.1 Executing XFOIL

XFOIL was altered such that it reads in command line arguments that provide instructions, rather than direct input from the operator via menu options. This allows rapid execution by an external program, such as MATLAB. From the DOS prompt, or by executing a system command, XFOIL can be instructed how to process input data, and which results to save. A simple example of how the modified XFOIL program may be executed as follows. At the DOS command prompt, in a directory containing the xfoil.exe program, the user may type:

```
"xfoil NACA 4415 OPER ALFA 5 OPER CPWR output"
```

The above command instructs XFOIL to use internal definition for a NACA 4415 foil, at an angle of attack of 5°, and write the pressure coefficient (Cp) distribution to a file named "output". When executed, the command line above results in a DOS output of the following:

```
START of XFOIL
START of Menu Loop. Command is: NACA
Using NACA  4415
 Max thickness =     0.150043  at x =   0.301
 Max camber    =     0.039999  at x =   0.398
START of Menu Loop. Command is: OPER
OPER loop command:  A
Angle of Attack:      5.000
 Calculating unit vorticity distributions ...
 OPER loop complete.
START of Menu Loop. Command is: OPER
OPER loop command:  CPWR
OPER loop complete.
```

The DOS output above represents informational items intentionally left in the XFOIL program to allow the user to verify that commands were executed properly. The result of the XFOIL calculations are written to a user specified file. Sample format is as follows. The first column is the X-location, starting at the trailing edge, continues along the upper surface around the nose back to the trailing edge. The second column is the calculated pressure coefficients at the corresponding locations.

```
#     x         Cp
   1.00000   0.48832
   0.99329   0.28542
   0.98206   0.19383
   0.96938   0.11911
      ...       ...
   0.95217   0.26832
   0.96743   0.28589
   0.98105   0.31188
   0.99296   0.35546
   1.00000   0.48832
```

Details of the structure and format of commands are included in Appendix F: Modified XFOIL User Guide. In addition, Appendix G: Instruction for compiling modified XFOIL Code contains additional instructions for obtaining and compiling the source code for the modified version of XFOIL.

4.4 Comparison of XFOIL Calculated Pressure Distributions

In section 3.5, the exact solution to the Karman-Trefftz foil pressure distribution was compared to the method presented by Brockett. Figure 8 to Figure 11 compare the XFOIL calculated pressure distribution, the Karman-Trefftz solution and the Brockett solution. In each instance, it can be seen that the XFOIL solution is nearly identical to the analytic solution. The XFOIL calculations for Figure 8 through Figure 11 were performed by instructing XFOIL to repanel the foil using 50 panels. Although higher panel resolution could be specified (160 is default), 50 was specified to prevent an excessive number of data points.

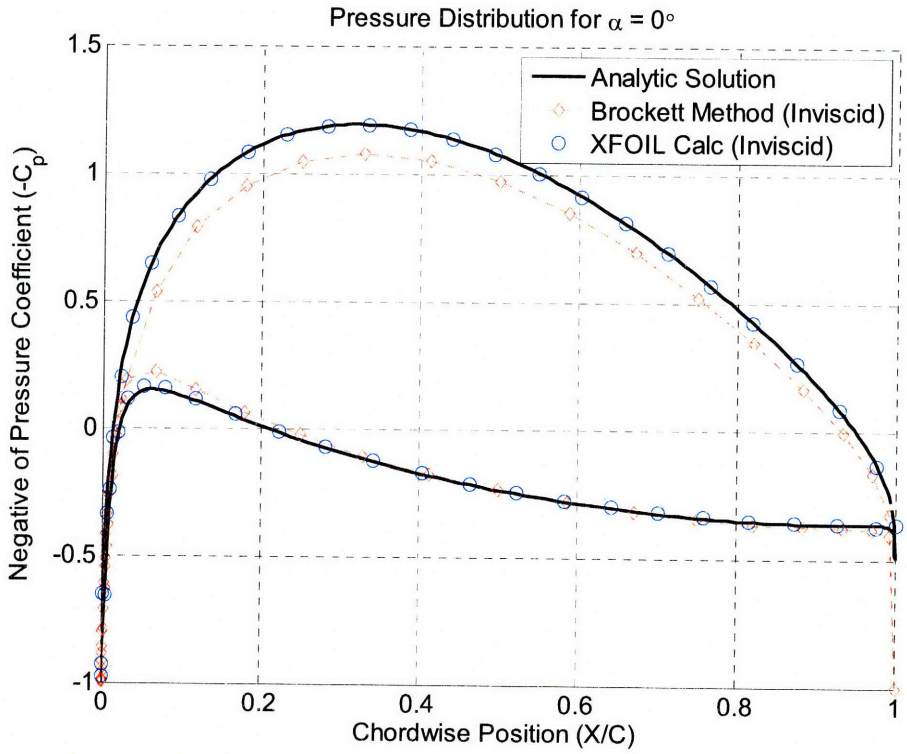


Figure 8: Comparison of XFOIL and Brockett Method to Exact Solution, α =0°

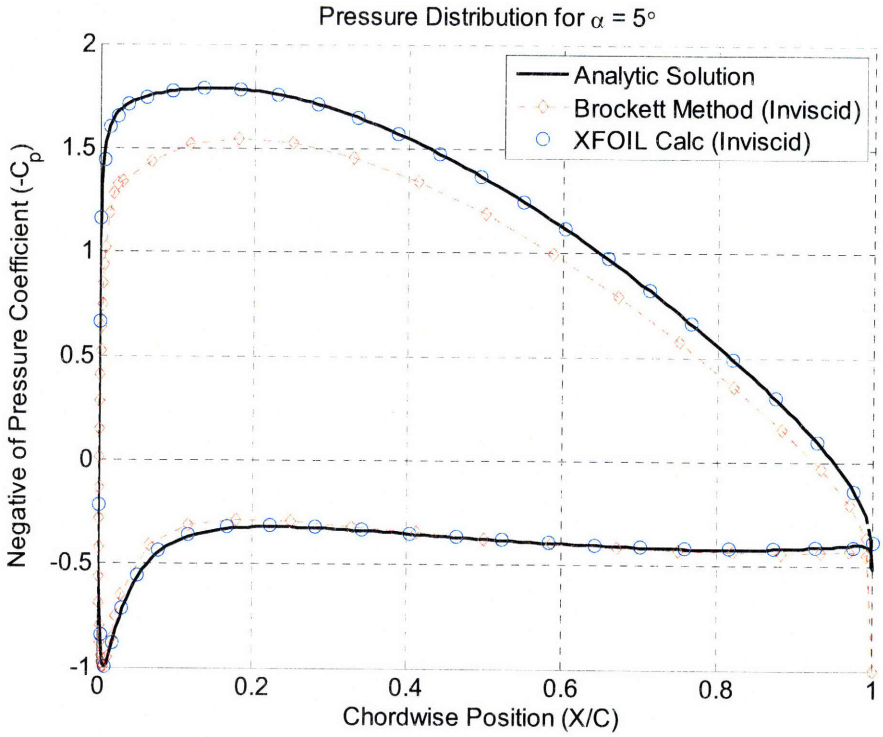


Figure 9: Comparison of XFOIL and Brockett Method to Exact Solution, α =5°

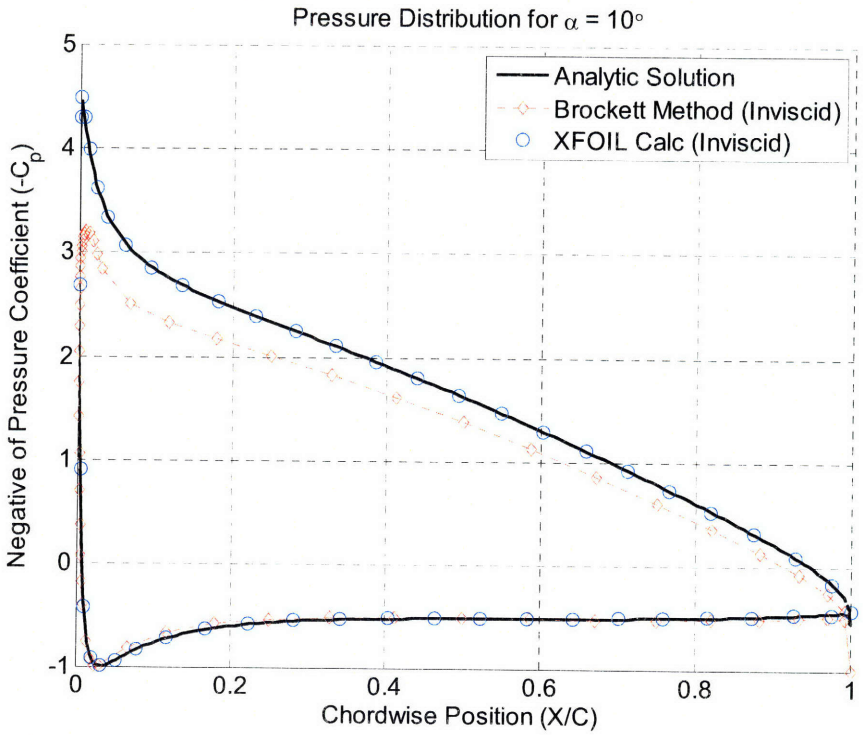


Figure 10: Comparison of XFOIL and Brockett Method to Exact Solution, α =10°

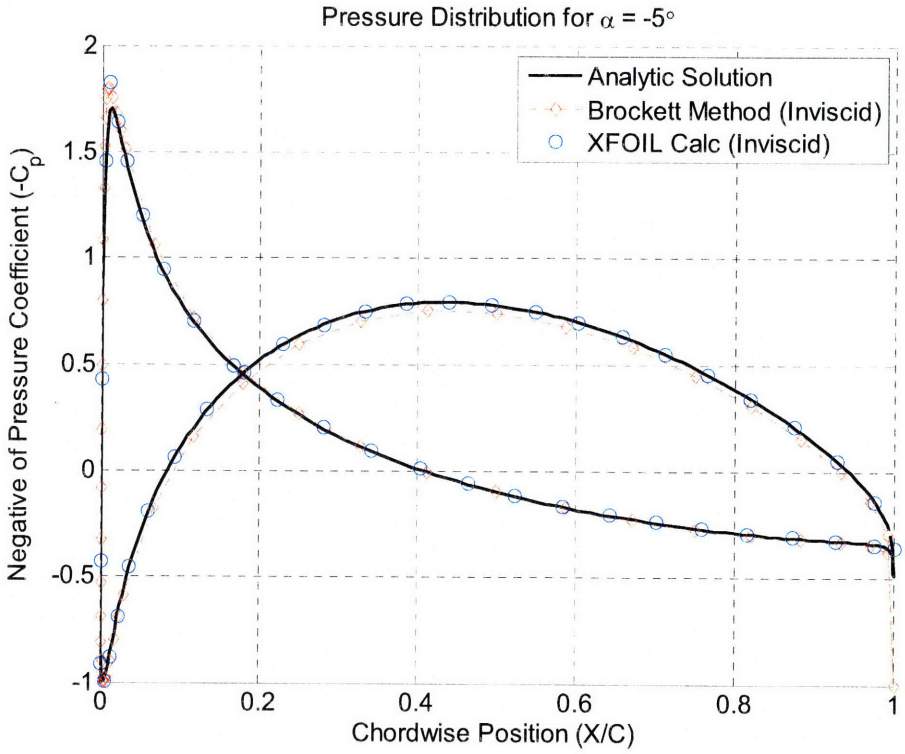


Figure 11: Comparison of XFOIL and Brockett Method to Exact Solution, α =-5°

5 Minimum Pressure Envelope Analysis

5.1 Background

In reference [4], Brockett published minimum pressure envelops for modified NACA-66 sections with NACA A=0.8 camber and BUSHIPS Type I and Type II sections using the calculation method described in section 3. These minimum pressure envelopes were computed for steady two-dimensional flow, with an empirical correction for viscosity. In addition, design charts for selecting "optimum" foils were included.

The work presented here includes a similar analysis method, with calculations performed by XFOIL, allowing the generation of minimum pressure envelopes for an arbitrary foil shape. This was accomplished using MATLAB integrated with the modified version of XFOIL described in section 4.3. Based on the improved accuracy of XFOIL over the method proposed by Brockett as shown in section 4.4, it is ascertained that this method provides a more accurate calculation of the pressure distribution, and location and magnitude of the minimum pressure for the inviscid solution

5.2 Description of Minimum Pressure Envelope Generation

Appendix H: MATLAB Files for Calculation of Minimum Pressure Envelopes contains the MATLAB files that were used to generate the minimum pressure envelopes using the modified XFOIL executable. This script performs various functions described in the following sections.

5.2.1 Foil Shape Generation

Foils may be defined in either of two methods. XFOIL contains built in functions defining NACA 4 and 5-digit series foils. If the user desires to use these NACA foils, then the `foil_type` variable should be set to "NACA". If NACA series foil shape is desired, the user must also set the variable `foil_name` to either "FOUR" or "FIVE", depending on which NACA series is desired, and the chordwise position of maximum camber must be specified by the `fo_loc` variable.

If profile data will be read in from a data file, then the `foil_type` variable should be set to "LOAD". Foil shape is defined by meanline and thickness information. The data files containing the meanline and thickness information are specified by the user using the `mean_type` and `thick_type` variables, which are set to the name of the files containing the meanline and thickness offset values. The data files need to be located in the corresponding "Meanline" and "Thickness" folders. Sample format for these files is contained and described in Appendix I: Meanline and Camber Data File Format.

The `makefoil.m` function inputs the meanline and camber data from the files specified, and combines the meanline and thickness distributions in the standard method as described by Abbott and Von Doenhoff [8]. In addition, the camber and thickness distributions are scaled if required. The `makefoil.m` function utilizes MATLAB's spline function to interpolate the required points to define the foil surface. XFOIL requires that foil geometry is specified by defining the X-Y locations along the foil surface from trailing edge, along the upper surface, around the leading edge, and back to the trailing edge along the lower surface. This is accomplished within the `makefoil.m` function.

The user may specify the number of desired output points to export to XFOIL by specifying the `N_parab_eval` and `N_surf_pts` in the `makefoil.m` script. Care should be used to specify a reasonable number of points, especially along the leading edge. Too many points may cause errors in XFOIL due to excessively small spacing. However, a sufficient number of points to adequately define the foils should be used, provided they are adequately spaced, with more points in regions of higher curvature. `N_parab_def` specifies the number of points used for creating the spline that defines the nose radius. If less than approximately 20 points are specified, the spline utility fails to produce a smooth output curve. Input and output may be plotted to verify proper definition of surface locations by setting the `make_plot` variable to 'yes', otherwise it should be set to 'no' to prevent excess plot generation.

5.2.2 User Specifications and Output from XBucket.m

The output from the MATLAB script, `XBucket.m`, may be specified by the user. The purpose of the script is to produce minimum pressure envelopes for the foil geometry specified by the user. Output plots are a similar format to that of Brockett [4]. Sample output plots show in Figure 12 and Figure 13 below.

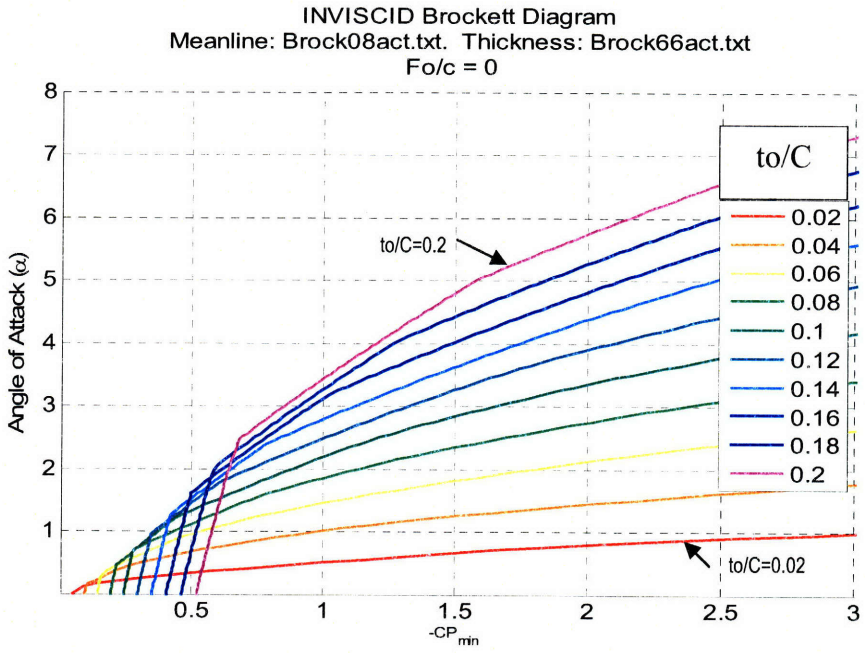


Figure 12: Minimum Pressure Envelopes for NACA 66 Section (TMB Mod. Nose and Tail) with Zero Camber at Various Thicknesses

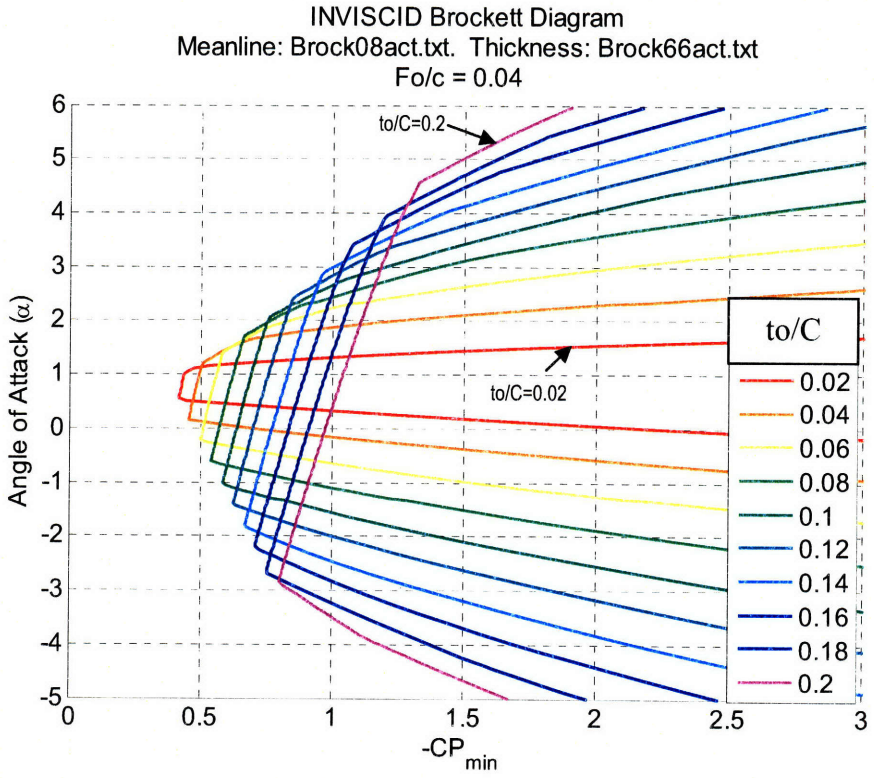


Figure 13: Minimum Pressure Envelopes for NACA 66 Section (TMB Modified) with the NACA a=0.8 Camberline, Having a Maximum Camber Ratio of 0.04 at Various Thicknesses.

`XBucket.m` generates output as in Figure 12 and Figure 13 based on user specified ranges. The upper and lower bounds of the angle of attack for which calculations and plotting are performed is specified by `Alpha_lim`. `Alpha_delta` specifies the resolution, or increment in angle of attack, for which each minimum pressure coefficient is determined. Larger values of `Alpha_delta` save calculation time, but produce less accurate plots.

Each plot produced is for a specified camber ratio. The desired range and camber ratio increment are specified by the `foc_rng and foc_step` variable. A separate plot will be produced for each camber ratio from the lower `foc_rng` value to the upper `foc_rng` value, in increments of `foc_step`.

Similarly, on each plot are minimum pressure envelopes for each thickness ratio. The range of values for thickness is specified by `toc_rng`, in increments of `toc_step`. A separate curve is plotted for each thickness value.

Also, although not a specific concentration of this project, the user may specify that XFOIL's viscous calculation mode be used. In order to conduct viscous calculations, the user must set `visc_tog` to 1, and specify the desired Reynolds number for calculation. This function has been incorporated for further research. Initial results are not reliable, as XFOIL does not converge consistently. The effect of convergence failure is shown for a typical case in Figure 14. The jagged curve is a result of XFOIL's viscous calculations failing to converge when calculating the minimum pressure coefficient for a given angle of attack.

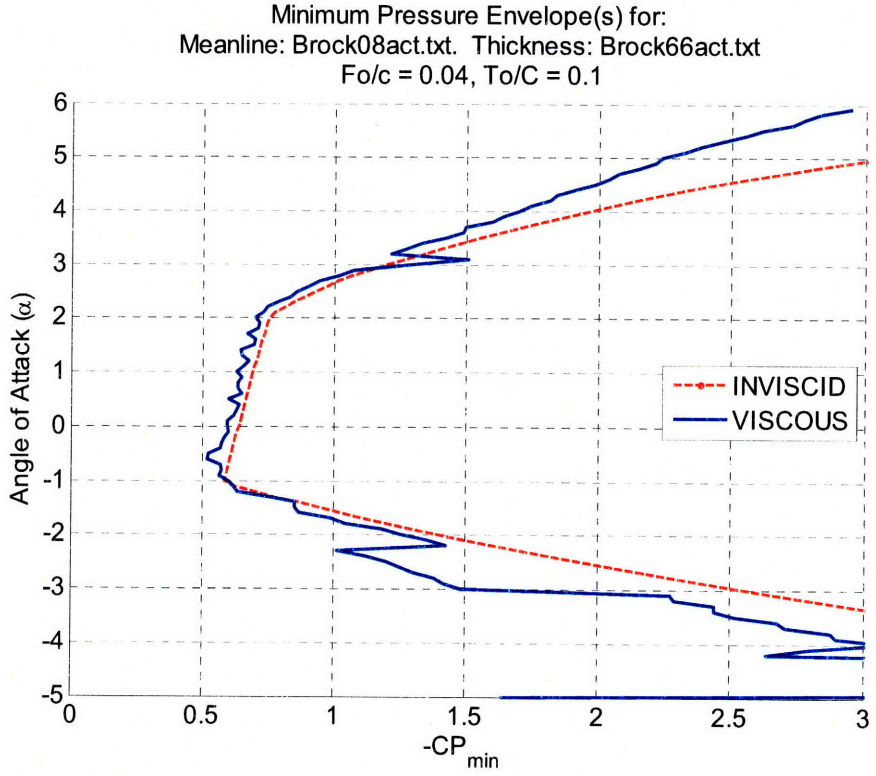


Figure 14: Sample Viscous and Inviscid Minimum Pressure Envelopes Calculated by XFOIL (Reynolds Number = $1*10^6$, 100 Maximum Iterations)

It is believed that small panel size (or excessive number of panels), angle of attack, Reynolds Number, maximum number of iterations and viscous solution acceleration parameter (VACC) are all factors that affect XFOIL convergence. Various combinations were tried to improve convergence. Results were improved when adding the command to repanel the foil with 70 panels, vice the previous value of

140. Results are shown in Figure 15. Additional research should be conducted to evaluate the viscous calculation capability of XFOIL, and determine how to most effectively set parameters that result in smooth, consistent, convergent results.

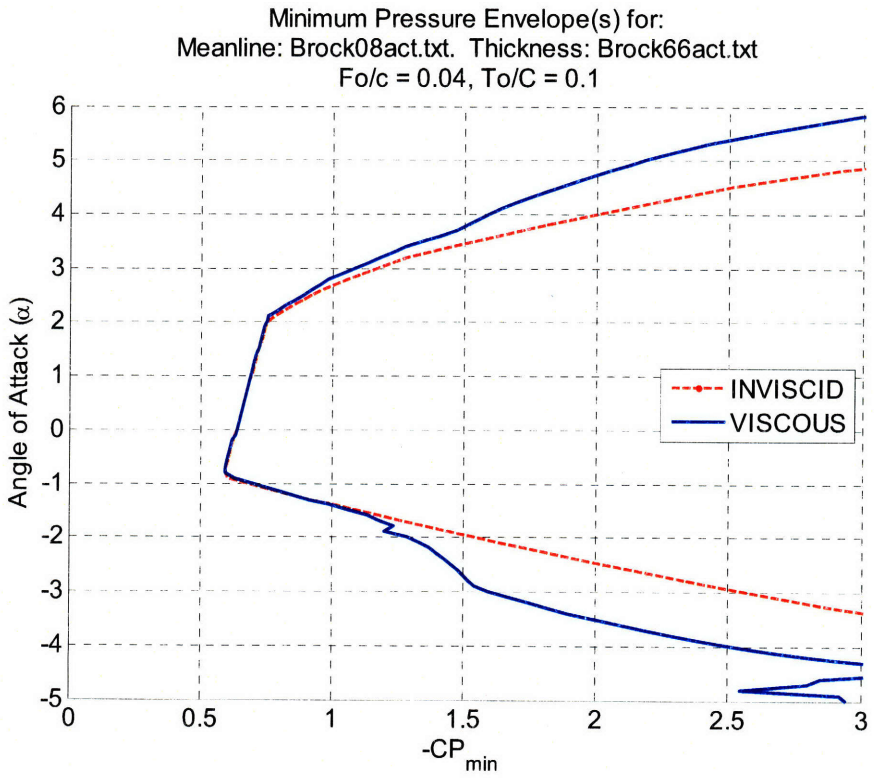


Figure 15: Sample Viscous and Inviscid Minimum Pressure Envelopes Calculated by XFOIL (Reynolds Number = $1*10^6$, 100 Maximum Iterations, 70 Panels)

5.3 Comparison of Brockett's Method to XFOIL Results

This intent of this research was to create a MATLAB based utility that would reproduce the minimum pressure diagrams published by Brockett [4], which could be later integrated into OpenProp for propeller design. Initial attempts using a simple two-dimensional panel method did not closely match Brockett's published results. As a result, XFOIL was implemented in order to conduct the pressure distribution calculations. XFOIL was chosen due to its highly regarded reputation as an accurate tool for conducting foil analysis and design. Results using XFOIL were still not able to reproduce the data as expected. Finally, the program as published by Brockett in reference [1] was reprogrammed in MATLAB in order to conduct further comparison. It was this comparison that revealed the noticeable differences

between Brockett's results and exact theory for potential flow, as previously shown in Figure 4 through Figure 7.

Figure 16 below illustrates the differences between Brockett's published minimum pressure envelopes, and the XFOIL calculated results. Figure 16 presents minimum pressure envelopes for the NACA 66 (TMB Modified), a = 0.8 meanline. Each individual curve was developed for a camber ratio of 0.06, and a thickness ratio of 0.12.

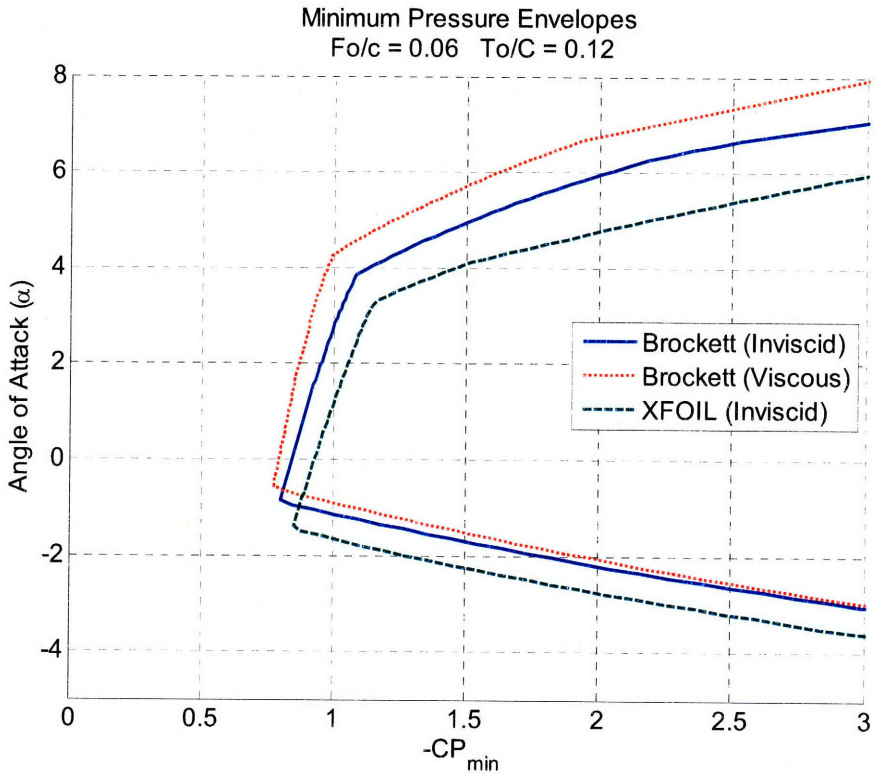


Figure 16: Minimum Pressure Envelope Comparison

Figure 16 illustrates the difference between Brockett (Inviscid) and the XFOIL (Inviscid) solutions. The difference is a result of inaccuracies of the Brockett method to predict the minimum pressure coefficient, as previously discussed in section 4.4, and summarized in Table 2. Specifically, the Brockett method underestimates the magnitude of the minimum pressure coefficient for intermediate and positive angles of attack, which corresponds to the near vertical and upper portions of the minimum pressure envelope, and overestimates the magnitude of

minimum pressure at negative angles of attack. The near vertical portion of the graph represents the region of operation when the minimum pressure occurs in the vicinity of the mid-chord. The upper and lower portions of the envelope correspond to nose cavitation, when the minimum pressure occurs near the leading edge of the foil due to elevated angles of attack.

The trend shown in Figure 16 is typical for all thicknesses and camber ratios. As a result, the overall minimum pressure envelopes as shown in Figure 12 and Figure 13 do not exactly match the published results of Brockett [4]. In addition, Brockett's published minimum pressure envelops for modified NACA-66 sections with NACA a=0.8 camber include an empirical correction for viscosity. The difference between Brockett's potential theory calculation and empirical correction for viscosity is also shown in Figure 16. The magnitude of the difference between Brockett's viscous and inviscid calculations is approximately equal to the magnitude difference between Brockett's inviscid calculation and XFOIL. As a result, it is believed that further investigation should be conducted to account for the viscous effects, and how viscous effect could be accounted for using XFOIL. As previously noted, XFOIL is capable of performing viscous calculations, and that ability was retained in the modified version XFOIL used for this work. However, accurate results were not reliably obtained, and were not evaluated. Further research in this area is recommended, which would greatly enhance the capabilities generated as for this project.

6 OpenProp Implementation Approach

Open-source Propeller Design and Analysis Program (OpenProp) is an open source MATLAB®-based suite of propeller numerical design tools. This program is an enhanced version of the *MIT Propeller Vortex Lattice Lifting Line Program* (PVL) developed by Professor Justin Kerwin at MIT in 2001. OpenProp v1.0, originally titled MPVL, was written in 2007 by Hsin-Lung Chung and is described in detail in [9]. Two of its main improvements versus PVL are its intuitive graphical user interfaces (GUIs) and greatly improved data visualization which includes graphic output and three-dimensional renderings.

OpenProp was designed to perform two primary tasks: parametric analysis and single propeller design. Both tasks begin with a desired operating condition defined primarily by the required thrust, ship speed, and inflow profile. The parametric analysis produces efficiency diagrams for all possible combinations of number of blades, propeller speed, and propeller diameter for ranges and increments entered by the user. Efficiency diagrams are then used to determine the optimum propeller parameters for the desired operating conditions given any constraints (e.g. propeller speed or diameter) specified by the user.

OpenProp was developed to serve as an open source code for propeller design. While it is currently a tool used in the initial design phase, it is a base program that can be continually expanded to perform detailed design and analysis of sophisticated marine propulsors and turbines. Development of a method of cavitation analysis that could be integrated into OpenProp was a primary motivation for this thesis.

The use of MATLAB provides for integration into the propeller design suite, OpenProp. Integration of cavitation analysis into OpenProp would provide the designer information about cavitation conditions while early in the design process, allowing adjustments to blade geometry to correct deficiencies. Following the design recommendations of Brockett [4], design charts or internal data feedback could provide adjustments to blade geometry. Cavitation prediction can either be

conducted for existing foils, or foil design could be selected in order to avoid cavitation for a given set of operating conditions.

6.1 Analysis of Existing Foils

For an existing propeller, where blade geometry is known, the code presented here could be used to conduct cavitation analysis for the foil. The geometry for the foil can be formatted as required, and may be used as input. To predict cavitation on existing foils, the minimum pressure curve for the propeller geometry at the radial position under investigation should first be generated. Then, based on operating conditions (angle of attack and local cavitation number, $\sigma = [p_\infty - p_{vapor}] / [\frac{1}{2}\rho U^2]$), the operating point may be compared to the calculated minimum pressure envelope. By setting the cavitation number equal to the negative of the minimum pressure coefficient, the operating point may be determined. If the operating point falls within the region bound by the minimum pressure envelope, cavitation is assumed not to occur. Cavitation is assumed to occur in the region outside of the minimum pressure envelope.

To analyze a complete propeller blade, it is recommended that a routine be created that analyzes the propeller blade at various radial positions from the hub to the tip at user specified intervals. At each radial position, the geometry must be determined as input. In addition to the minimum pressure coefficient, the pressure distribution along the chord may be calculated and compared to the cavitation number. By determining where the negative of the pressure coefficient is greater than the cavitation number, regions along the propeller where cavitation is predicted could be determined. These regions could then be used to produce a color coded plot of the surface of the propeller blade, indicating regions were cavitation is predicted to occur.

Margin to cavitation may also be determined. For propellers that are predicted not to cavitate, the operating angle of attack can be compared to the angles of attack at the upper and lower bounds of the minimum pressure envelope for the cavitation number. The difference between the operating angle of attack and the angles of

attack at the envelope boundaries gives an indication of how close the propeller is to cavitation based on expected operating conditions. This information can also be used to predict how far from design conditions the propeller may be operated before the onset of cavitation. Varying inflow would be an example of off design conditions that could be analyzed using the margin to cavitations. For example, if the inflow is known to vary by 2° around the circumference, then as long as the margin to cavitation is greater than 2°, cavitation would not be expected to occur due to varying inflow.

6.2 Geometric Design to Prevent Cavitation

Rather than analyzing an existing propeller, minimum pressure envelopes may be used as an aid to the designer in producing propeller blade geometry that is optimized to prevent cavitation. 'Optimum' foil geometry, as described by Brockett, allows the greatest total angle change without occurrence of cavitation for a given cavitation number. The optimum foil is the one for which the minimum pressure envelope is the widest at the given $-Cp_{min}$. In other words, it is the thickness which provides the greatest envelope width for the given operating conditions. Figure 17 outlines a basic procedure that could be implemented into OpenProp, utilizing the information provided from the minimum pressure envelope data to assist in propeller design.

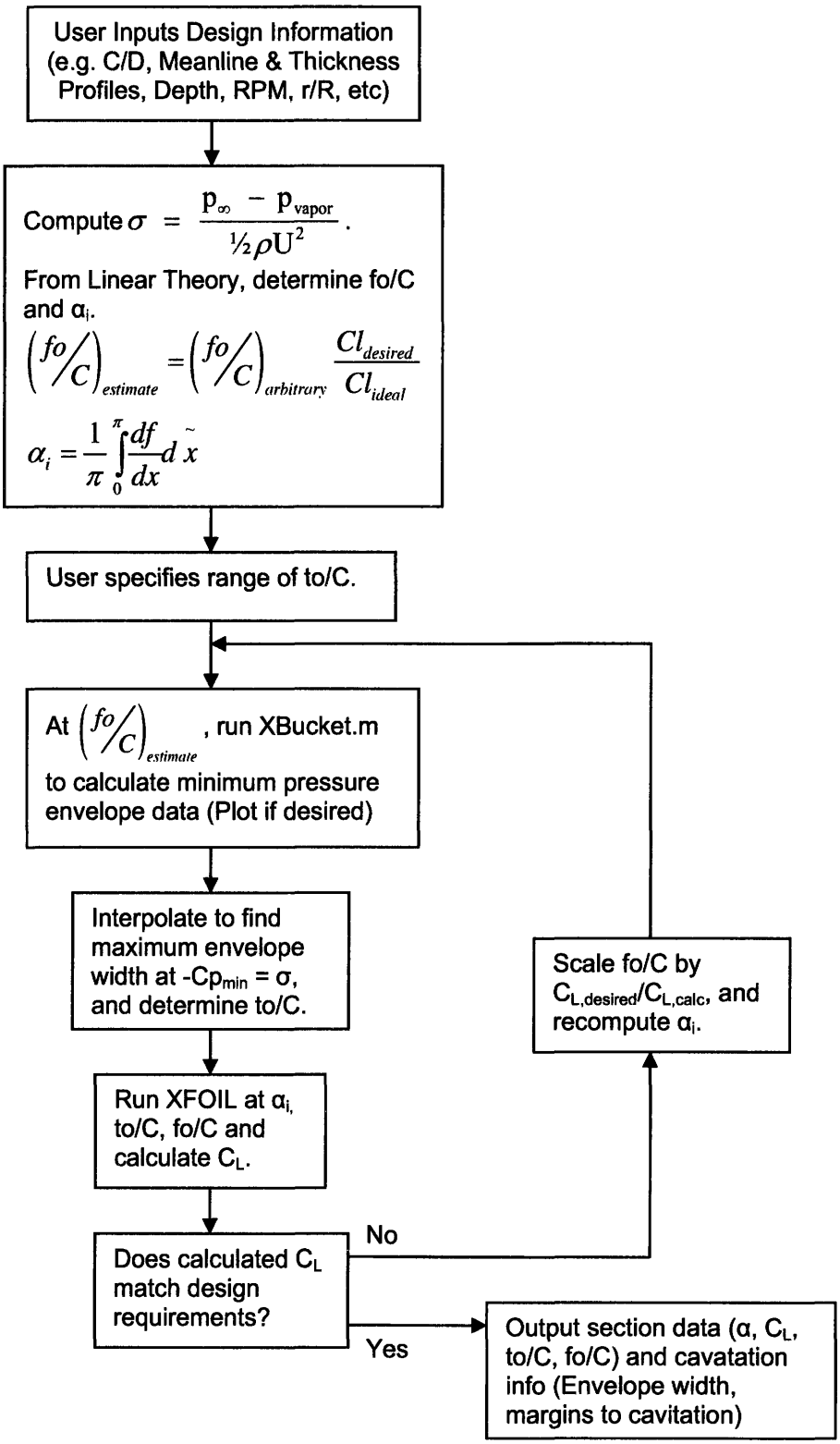


Figure 17: Example Flowchart for using Minimum Pressure Envelopes for Design.

For cambered foils, there are two separate curves that bound the minimum pressure envelopes, one for the upper portion, and one for the lower. The curve bounding the upper portion is the limit for the maximum envelope width, and is shown in Figure 18. This boundary will be used to determine the optimal foil thickness.

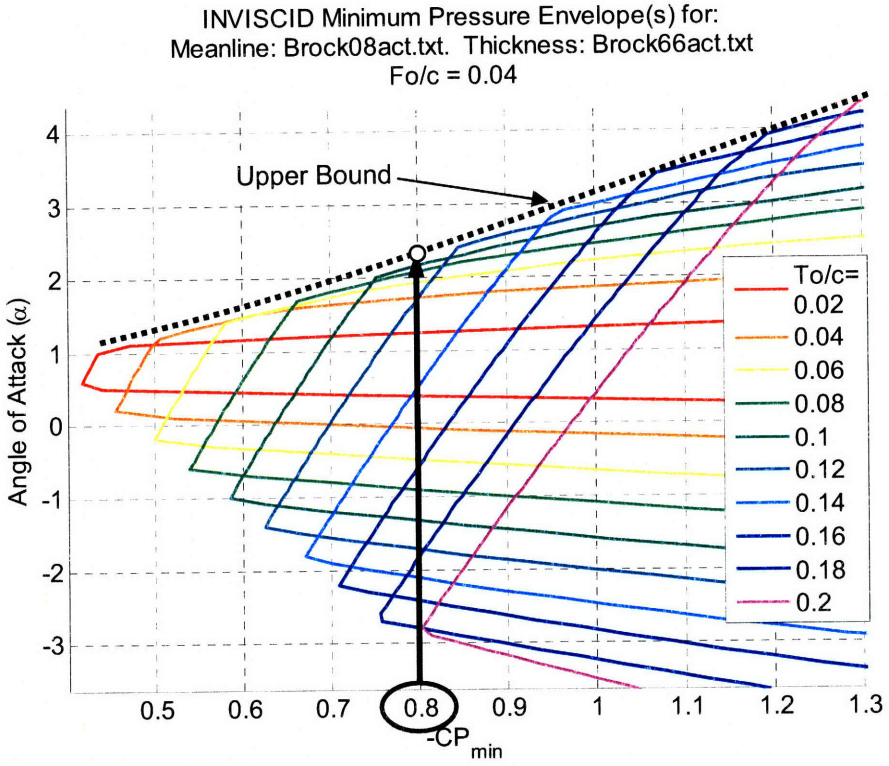


Figure 18: Minimum Pressure Envelopes to Find Optimum Thickness

The designer must first specify a desired meanline and thickness distribution, howver, the values of fo/C and to/C are not yet known. To determine the angle of attack, an arbitrary fo/C must be specified. From thin airfoil theory, the idea angle of attack, α_i, is defined as the angle of attack for which the coefficient $A_o = 0$. A_o is the angle of attack dependent coefficient in the Fourier series expansion of df/dx (camberline slope) [5]. Equations (6.1) and (6.2) below provide details.

$$\alpha_i = \frac{1}{\pi}\int_0^{\pi}\frac{df}{dx}d\tilde{x} \qquad (6.1)$$

$$x = -\frac{c}{2}\cos(\tilde{x}) \tag{6.2}$$

Tabulated camberline data may be available that includes α_i and $C_{L,ideal}$ for a given fo/C, and may be used instead of specifying an arbitrary fo/C and calculating α_i and $C_{L,ideal}$. Once determined, the ideal lift coefficient, which is the lift coefficient for the foil at the ideal angle of attack, should be evaluated, and may be obtained from XFOIL. Once the ideal lift coefficient is determined, the initial estimate for camber ratio of the design foil may be determined by scaling the camber by the ratio of the desired lift coefficient to ideal lift coefficient.

$$\left(fo/C\right)_{estimate} = \left(fo/C\right)_{arbitrary} \frac{Cl_{desired}}{Cl_{ideal}} \tag{6.3}$$

Using the minimum pressure envelopes for the estimated camber, the thickness ratio can be determined based on the upper bounding curve of Figure 18 by entering the graph at the design cavitation number. For example, if the value of the cavitation number is assumed to be 0.8, the optimum foil thickness from Figure 18 is between 0.1 and 0.12, and could be interpolated as approximately to/C = 0.11. Numerically, this procedure could be accomplished by one of two methods. If the location of the "knuckles" of the envelopes can be identified, the bounding curve for the minimum pressure envelopes can be determined, as in Figure 18. Defining the bounding surface as a function of thickness ratio would allow the designer to directly calculate the thickness that results in the widest envelope for the given cavitation number. This method was attempted, but determining the location of the "knuckles" was difficult and a reliable method was not found. Rather than determining a function that describes the boundary of the minimum pressure envelopes, it is recommended that the envelope widths be calculated for each of the thicknesses, and a maximum determined for the specified cavitation number.

Based on design specifications (lift coefficient and cavitation number), the designer has the option of two methods for determining an angle of attack for which to place the foil. The angle of attack may be specified as the ideal angle of attack, or the

angle that maximizes the margin to cavitation, located halfway between the upper and lower portion of the minimum pressure envelope curve for the given thickness at the specified cavitation number. By definition, the ideal angle of attack will minimize the local pressure spike at the leading edge, but may not provide equal margins to caviation about the operating point, particularly for higher to/C. Howerver, the ideal angle of attack should provide a useful starting point for the design.

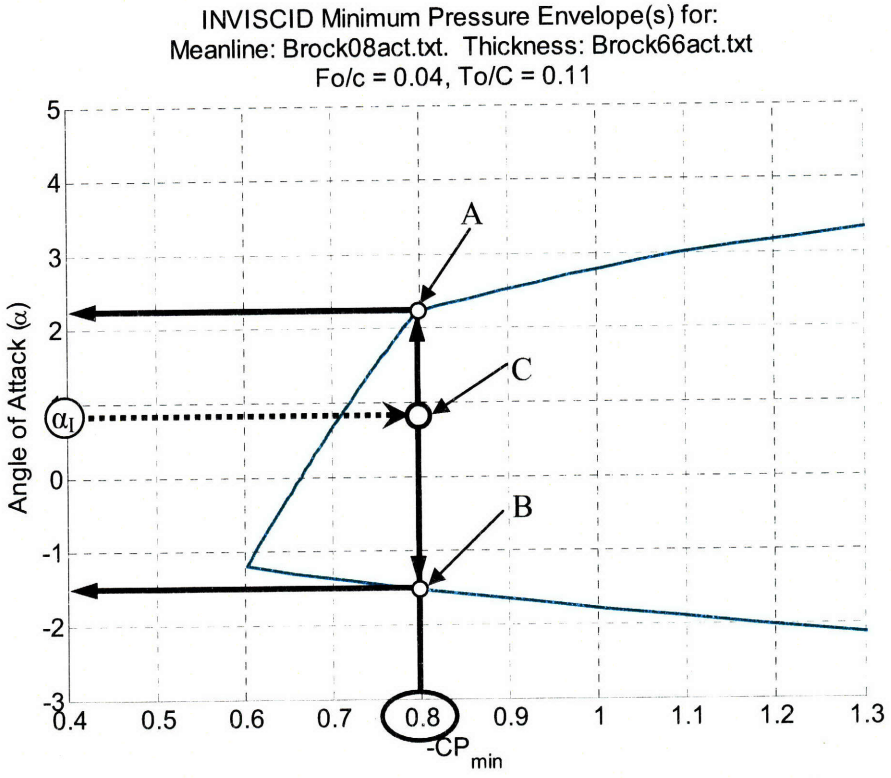


Figure 19: Minimum Pressure Envelopes to Find Envelope Width and Cavitation Margin

Figure 19 shows the minimum pressure envelope for to/C = 0.11. From this graph, the cavitation envelope width may be determined. Points A and B correspond to the values of the minimum pressure envelope at $-Cp_{min} = \sigma$, and provide the values for the angle of attack where the onset of cavitation is expected. Cavitation is not expected between -1.5° and 2.2°. Point C is the operating point if the foil is placed at the ideal angle of attack of 0.9°. The width of the cavitation envelope is 3.7°, which is +1.3° and -2.4° from design angle of attack. XFOIL should be use to

recalculate lift coefficient at the design parameters to verify required lift coefficient is achieved.

Rather than assuming the foil be operated at the ideal angle of attack, a method that maximizes that margin tocavitation could be used instead. Cavitation margin is defined as the magnitude of the difference between the operating angle of attack, and the angle at which cavitation is predicted. In order to maximize the cavitation margin, the operating angle of attack should be exactly in the middle of the upper and lower portions of the minimum pressure envelope. For this method, the operating angle of attack would be specified as (2.2° - 1.5°)/2 = 0.35°. However, by changing the angle of attack, the lift coefficient would be reduced and may not provide the required lift. As a result, the camber would need to be increased, and design process would be repeated until design criteria were met.

7 Conclusion

A method of generating minimum pressure envelopes using XFOIL was created. By modifying the source code, a version of XFOIL that does not require user interaction was created. Using MATLAB to interface with XFOIL, minimum pressure envelopes for an arbitrary foil shape can be generated, provided offset data is available for foil geometry.

The minimum pressure envelopes created as a result of the XFOIL calculations were compared to published work by Brockett. It was found that the two-dimensional panel method of XFOIL could more accurately estimate the potential flow solution for a Karman-Trefftz foil than the approximate conformal transformation method used by Brockett. Although XFOIL includes the ability to conduct calculations for a viscous fluid, additional work is required in order to evaluate the limits for which XFOIL's viscous mode will reliably converge.

The code developed as part of this thesis is intended to be used for further integration into OpenProp. Integration into OpenProp will allow the user to both conduct cavitation analysis and prediction for existing foils, as well as allow the designer to consider cavitation in the design process, and select foil geometry that will prevent cavitation.

7.1 Recommendations for Future Work

7.1.1 Viscous Calculations

In order to utilize the benefits of the highly accurate potential flow solution available from XFOIL, the effects of viscosity must be reevaluated, and accounted for. The empirical modification used by Brockett depends upon specifying an experimental lift coefficient for each angle of incidence, and can be determined from the lift-curve slope (η) and angle of zero lift (α_{0e}), using the following equation:

$$C_L = 2\pi\eta\left(\alpha - \alpha_{0_e}\right) \tag{7.1}$$

It has been experimentally shown [8] that η and α_{0e} are independent for high Reynolds numbers (>$6x10^6$). This method should be compared to the manner by which viscous calculations are conducted in XFOIL. XFOIL's viscous mode should be integrated into the methods presented here for calculating minimum pressure envelopes, if found to be more accurate. In addition, the parameters affecting convergence of XFOIL's viscous calculation should be evaluated to ensure proper problem formulation and evaluation.

7.1.2 OpenProp Integration

The design approaches outlined in section 6 should be developed and integrated into OpenProp. Once developed, these methods would provide great benefit and enhance the utility of the OpenProp design suite. Program code that performs the basic functions required to integrate cavitation design were developed in this thesis. In particular, the development of the modified version of XFOIL allows rapid calculation of complex foil geometries, and simplifies the method of retrieving data from XFOIL and entering that data into MATLAB. In addition, the MATLAB code generated in this project provide a starting point from which to develop the functionality and usefulness of OpenProp.

Bibliography

[1] Brockett, Terry E. *Steady Two-Dimensional Pressure Distributions on Arbitrary Profiles.* MS Thesis, Cornell University (1965).

[2] Moriya, T. *On the Aerodynamic Theory of an Arbitrary Wing Section.* Journal of the Society of Aeronautical Science, Japan, Vol. 8, no. 78 (1941), p. 1054.

[3] Theodorsen, Theodore. *Theory of Wing Sections of Arbitrary Shape.* NACA Rep. 411, 1931.

[4] Brockett, Terry. *Minimum Pressure Envelopes for Modified NACA-66 Sections with NACA a = 0.8 Camber and BUSHIPS Type I and Type II Sections.* David Taylor Model Basin, Report 1780. February 1966.

[5] Kerwin, Justin E. 13.04 Lecture Notes, *Hydrofoils and Propellers*, Massachusetts Institute of Technology (2001).

[6] Drela, Mark and Youngren, Harold. XFOIL 6.9 User Primer (30 Nov 2001). Retrieved Dec 15, 2007, from http://web.mit.edu/drela/Public/web/xfoil/xfoil_doc.txt

[7] Drela, Mark. *XFOIL: An Analysis and Design System for Low Reynolds Number Arifoils.* In: T.J. Mueller, editor. *Low Reynolds Number Aerodynamics: Proceedings for the Conference,* Notre Dame, Indiana, USA, 5-7 June 1989. Springer-Verlag, p. 1-12.

[8] Abbott, Ira H. and Von Doenhoff, Albert E. *Theory of Wing Sections.* Dover Publications, Inc., New York (1959).

[9] Chung, H. *An Enhanced Propeller Design Program Based on Propeller Vortex Lattice Lifting Line Theory,* MS Thesis, Massachusetts Institute of Technology, Department of Mechanical Engineering (2007).

Appendix A: MATLAB Script for Conformal Transformation of Karman-Trefftz Foil.

```
%   Code Developed by Chris Peterson to calculate and display Conformal
%   Transformation of a Karman-Trefftz foil. Intended to be used to
%   compare values for different methods of calculating surface velocites
%   for airfoils.
clc; clear all;close all;
%User defined Data
U           = 1;            %Free Stream Velocity
alpha_deg   = 0;            %Angle of Attack (Degrees)
xc          = -0.10;        %Circle Center Location (<0)
yc          = 0.150;        %Circle Center (>0 adds + camber)
tau         = 10;           %Tail Angle
n_pts       = 201;          %Number of points along mapped foil surface
out_pts     = 36;           %Total number of X-Y output points (ODD)
a           =  1;           %X-intercept
alpha       = deg2rad(alpha_deg);
%Display parameters
No_strm     = 21;           %Number of Streamlines to plot
range       = 3;            %Z-plane X-Y Range
strm_strt   = -3;           %X-Location for streamline start
div         = 0.1;          %Grid spacing for velocity vectors on Z-plane
%Calculation of properties
beta        = atan(-yc/(1-xc));             %Angle to rear stagnation point
beta_deg    = rad2deg(beta);                %Beta in degrees
rc          = sqrt((a-xc)^2 + yc^2);        %Calculate radius of circle
Gamma_calc = 4*pi*rc*U*sin(beta-alpha);     %Kutta condition requirement
Gamma = Gamma_calc;                         %Set circulation to required
lam = 2-tau/180;                           %Trailing egde to lamba calculation
%Generate Z-plane Plot with streamlines and velocity vectors
[X,Y]   = meshgrid(-range:div:range,-range:div:range);  %Create location mesh
r       = sqrt((X-xc).^2 + (Y-yc).^2);              %Radius at mesh locations
%Calculate angle theta to mesh locations, 0 <= theta < 2*pi
for j=1:length(X)
    for k=1:length(X)
        if X(j,k) >= xc
            theta(j,k) = atan((Y(j,k)-yc)/(X(j,k)-xc));
        elseif X(j,k) < xc
            theta(j,k) = atan((Y(j,k)-yc)/(X(j,k)-xc)) + pi;
        end
    end
end
%Calculate velocity components u, v based on potential theory
u = U*cos(alpha) - (U.*((rc./r).^2).*cos(2.*theta - alpha))...
    - Gamma.*sin(theta)./(2.*pi.*r);
v = U*sin(alpha) - (U.*((rc./r).^2).*sin(2.*theta - alpha))...
    + Gamma.*cos(theta)./(2.*pi.*r);
%Calculate location of stagnation points
theta_s1    = asin(Gamma/(4*pi*rc*U)) + alpha;
theta_s2    = asin(-Gamma/(4*pi*rc*U)) + alpha - pi;
x_st1       = rc*cos(theta_s1)+xc;
y_st1       = rc*sin(theta_s1)+yc;
x_st2       = rc*cos(theta_s2)+xc;
y_st2       = rc*sin(theta_s2)+yc;
%Define point on circle
```

```
x_circ = xc + rc*cos(0:pi/21:2*pi);
y_circ = yc + rc*sin(0:pi/21:2*pi);
z_circ = x_circ + i*y_circ;       %z is complex coordinates of circle
%Eliminates points inside circle for vector plot (large values near
singularities)
u_mod = u;
v_mod = v;
for j = 1:length(X)
    for k = 1:length(X)
        if (X(j,k)-xc)^2 + (Y(j,k)-yc)^2 < rc^2
            u_mod(j,k) = 0;
            v_mod(j,k) = 0;
        end
    end
end
%   Plot Z-plane, with circle, stagnation points, velocity vectors and
%   streamlines.
figure()
orient landscape;
axis equal;hold on;grid on;ylim([-range range]);xlim([-range range]);
title({'Z-plane';'Potential Flow Around a Circle'});
set(gca,'YTick',-range:range);set(gca,'XTick',-range:range);
streamline(stream2(X,Y,u,v,strm_strt*ones(No_strm,1)...
    ,-range:2*range/(No_strm-1):range));    %Plots streamlines
plot(x_circ, y_circ, 'k')                   %Plots circle
plot(x_circ, y_circ, 'k.')                  %Plots circle points
plot(xc, yc, 'r+')                          %Plots circle center
plot(x_st1, y_st1, 'ko')                    %Plots stagnation point 1
plot(x_st2, y_st2, 'ko')                    %Plots Stagnation point 2
% quiver(X,Y,u_mod,v_mod, 'g');               %Plots Vectors
%Map surface of ccircle to Zeta-plane
Zeta_circ = lam*a*((z_circ+a).^lam + (z_circ-a).^lam)...
    ./((z_circ+a).^lam - (z_circ-a).^lam);
%Routine to find velocities and -Cp on foil surface
theta   = 0:2*pi/n_pts:2*pi-pi/n_pts;   %Defines theta incremented 0->2*pi
x_z     = xc + rc*cos(theta);           %X location in Z-plane
y_z     = yc + rc*sin(theta);           %Y location in Z-plane
u_z     = U*cos(alpha)...               %X velocity in Z-plane
          - (U.*cos(2.*theta - alpha)) - Gamma.*sin(theta)./(2.*pi.*rc);
v_z     = U*sin(alpha)...               %Y velocity in Z-plane
          - (U.*sin(2.*theta - alpha)) + Gamma.*cos(theta)./(2.*pi.*rc);
z_z     = x_z + i.*y_z;                 %Complex velocoity in Z-plane
%Transform Surface Locations & Velocities to Zeta Plane
Zeta    = lam*a.*...                    %Complex coords Zeta = f(z)
          ((z_z+a).^lam + (z_z-a).^lam)./((z_z+a).^lam - (z_z-a).^lam);
x_zeta  = real(Zeta);                   %X location in Zeta-plane
y_zeta  = imag(Zeta);                   %Y location in Zeta-plane
dzeta_dz = (4*(lam*a)^2)...             %D(Zeta)/Dz
          *( ( (z_z-a).^(lam-1)) .* ((z_z+a).^(lam-1)) )...
          ./(( ((z_z+a).^lam) - ((z_z-a).^lam)).^2);
vel_zeta = (u_z - i.*v_z)./dzeta_dz; %[u-iv]_zeta = [u-iv]_x/Dzeta/Dz
u_zeta  = real(vel_zeta);               %X velocity in Zeta-plane
v_zeta  = -imag(vel_zeta);              %Y velocity in Zeta-plane
q_zeta  = sqrt(u_zeta.^2 + v_zeta.^2);%Zeta-Velocity Magnitude
cp_zeta = 1-(q_zeta./U).^2;             %Zeta pressure coefficient

%Create plot of Zeta plane
```

```
figure(); grid on;hold on;axis equal;orient landscape;
xlim([-range range]);ylim([-range+1 range-1]);
title({'\zeta-Plane';'Potential Flow Around Mapped Foil'});
plot(Zeta_circ, 'k')
%plot(Zeta_circ, 'k.')
%Find Z coodinates in Z-Plane based on required spacing in Zeta-plane
%in order to calculate U-V components in mesh spacing for streamline plot
[Xgrd_zeta,Ygrd_zeta] = ...              %Create location mesh
    meshgrid(-range:div:range,-range:div:range);
Zetagrd = Xgrd_zeta + i*Ygrd_zeta;
Z_grd = -a.*((((Zetagrd-lam)./(Zetagrd+lam)).^(1/lam))+1)...
    ./((((Zetagrd-lam)./(Zetagrd+lam)).^(1/lam))-1);
X_grd = real(Z_grd);
Y_grd = imag(Z_grd);
r_grd       = sqrt((X_grd-xc).^2 + (Y_grd-yc).^2);%Radius at mesh locations
%Calculate angle theta to mesh locations, 0 <= theta < 2*pi
for j=1:length(X_grd)
    for k=1:length(X_grd)
        if X_grd(j,k) >= xc
            theta_grd(j,k) = atan((Y_grd(j,k)-yc)/(X_grd(j,k)-xc));
        elseif X(j,k) < xc
            theta_grd(j,k) = atan((Y_grd(j,k)-yc)/(X_grd(j,k)-xc)) + pi;
        end
    end
end
%Calculate velocity components u, v based on potential theory
u_grd_z = U*cos(alpha) - (U.*((rc./r_grd).^2).*cos(2.*theta_grd - alpha))...
    - Gamma.*sin(theta_grd)./(2.*pi.*r_grd);
v_grd_z = U*sin(alpha) - (U.*((rc./r_grd).^2).*sin(2.*theta_grd - alpha))...
    + Gamma.*cos(theta_grd)./(2.*pi.*r_grd);
for j = 1:length(X_grd)
    for k = 1:length(X_grd)
        if (X_grd(j,k)-xc)^2 + (Y_grd(j,k)-yc)^2 < rc^2
            u_grd_z(j,k) = 0;
            v_grd_z(j,k) = 0;
        end
    end
end
dzeta_dz_grd = (4*(lam*a)^2)...              %D(Zeta)/Dz
           *( ( (Z_grd-a).^(lam-1)) .* ((Z_grd+a).^(lam-1)) )...
           ./(( ((Z_grd+a).^lam) - ((Z_grd-a).^lam)).^2);
vel_grd_zeta = (u_grd_z - i.*v_grd_z)./dzeta_dz_grd;
u_zeta_grd  = real(vel_grd_zeta);            %X velocity in Zeta-plane
v_zeta_grd  = -imag(vel_grd_zeta);           %Y velocity in Zeta-plane
%quiver(Xgrd_zeta ,Ygrd_zeta, u_zeta_grd, v_zeta_grd, 'g')%Plots Vectors
streamline(stream2(Xgrd_zeta ,Ygrd_zeta, u_zeta_grd, v_zeta_grd,...
    strm_strt*ones(No_strm,1),-range:2*range/(No_strm-1):range));%Plots
streamlines
%Create and save plot of minimum pressure distribution
figure(); hold on; grid on; xlim([0 1])
xlabel('Chordwise Position (X/C)')
ylabel('Negative of Pressure Coefficient (-C_p)')
title(['Pressure Distribution for \alpha = ', num2str(alpha_deg),'\circ'])
plot((x_zeta-min(x_zeta))./(max(x_zeta)-min(x_zeta)), -cp_zeta, 'k.-')
saveas(gcf,'Trefftz.fig')
close();
%Scales foil to Chord lenght of 1.
```

```
chord = (max(x_zeta)-min(x_zeta));
x_zeta_scl = (x_zeta-min(x_zeta))./(max(x_zeta)-min(x_zeta));
y_zeta_scl = y_zeta./(max(x_zeta)-min(x_zeta));
%Locates nose location and index X and Y.
[x_nose, i_nose] = min(x_zeta_scl);
[x_tail, i_tail] = max(x_zeta_scl);
%Breaks scaled locations into upper and lower surfaces
x_US = [x_zeta_scl(i_tail:end) x_zeta_scl(1:i_nose)];
y_US = [y_zeta_scl(i_tail:end) y_zeta_scl(1:i_nose)];
x_LS = x_zeta_scl(i_nose:i_tail);
y_LS = y_zeta_scl(i_nose:i_tail);
%Defines X locations, cos-spaced, to be used for output
x_spl = (1+cos(0:2*pi/(out_pts-1):2*pi))/2;
%Splines Upper and Lower surfaces and evaluates at x_spl locations
spl_US = spline(x_US, y_US);
spl_LS = spline(x_LS, y_LS);
y_spl = [ppval(spl_US, x_spl(1:ceil(out_pts/2))) ...
    ppval(spl_LS, x_spl(ceil(out_pts/2)+1:end))];
%Summary plot to compare input, output, and spline functions
figure();hold on; axis equal;
fnplt(spl_US, 'r')
fnplt(spl_LS, 'g')
plot(x_US, y_US, 'k.')
plot(x_LS, y_LS, 'k.')
plot(x_spl, y_spl, 'bo')
legend('US Spline', 'LS Spline', 'US Data', 'LS Data', 'Output Points')
%Saves splined output point and angle of attack to file x_output
save('x_output', 'x_spl', 'y_spl', 'alpha_deg')
%Writes splined output X and Y locations to data file trefxy
fid = fopen('trefxy', 'w');
for i =1:length(x_spl)
    fprintf(fid, '%12.8f %12.8f\n', x_spl(i), y_spl(i));
end
fclose(fid);
%run brockthesis    %Starts Brockett's thesis for comparison of data.
```

Appendix B: MATLAB Code of Brockett's Work (Brockett.m)

```
%ADAPTATION OF BROCKETT'S THESIS WORK.  Code Modified by Chris Peterson.
%Code allows user specified input, and plots pressure distribution for
%given input.

clear all; clc;
prnt2scrn    = 1;          %Turn on (1) or off (0) screen output

run CfmlInput              %Allows user specified setting and data input.

%Preallocate memory for Improved Speed
CO = zeros(1,18);SO = zeros(1,18);X = zeros(1,18);ANTRP=zeros(1,12);
CNT=zeros(1,12);XA=zeros(1,12);SNT=zeros(1,12);COL=zeros(1,17);
COT=zeros(1,17);Z1=zeros(12,17);Z2=zeros(12,17);Z3=zeros(12,17);
Z4=zeros(12,17);EE=zeros(1,NX);DD=zeros(1,37);

if IDEN == 0
    SY=zeros(1,19);
    VL=zeros(1,19);
elseif IDEN > 0
    SY=zeros(1,36);
    VL=zeros(1,36);
end

%
%   CALCULATION OF CONSTANTS
%
AN=18.0;
for I=1:18
    TA=(I-1)*.17453293;
    CO(I)=cos(TA);
    SO(I)=sin(TA);
    X(I)=.5*(1.+CO(I));
end
SO(19)=0.;
CO(19)=-1.;
X(19)=0.;
for I=20:37;
    IA=38-I;
    X(I)=X(IA);
    CO(I)=CO(IA);
    SO(I)=-SO(IA);
end

%   INTERMEDIATE POINTS AND CORRESPOINDING X VALUES

for I=1:9
    ANTRP(I)=(I)*.017453293;
end
ANTRP(10)=12.5*.017453293;
ANTRP(11)=15.0*.017453293;
```

```
ANTRP(12)=17.5*.017453293;
for I=1:12
    CNT(I)=cos(ANTRP(I));
    XA(I)=.5*(1-CNT(I));
    SNT(I)=sin(ANTRP(I));
end

%   CALCULATION OF VECTORS USED TO OBTAIN SLOPE AND VELOCITY

for I=1:2:17
    COL(I) = -1/(AN*(1-CO(I+1)));
end

COEF   = 1;
for I=1:17
    COEF   = -COEF;
    COT(I)  = COEF*SO(I+1)/(1-CO(I+1))*0.5;
end

for I=1:12
    COEF    = 1;
    CNNT    = cos(18*ANTRP(I));
    SNNT    = sin(18*ANTRP(I));
    for J=1:17
        COEF=-COEF;
        TA  = (-CNT(I)-CO(J+1));
        TB  = (COEF*CNNT-1)/36;
        TC  = COEF*SNNT*0.5;
        TD  = TA*TA;
        TE  = (1+CO(J+1)*CNT(I))/TD;
        TF  = SO(J+1)*SNT(I)/TD;
        TD  = COEF*CNNT*0.5/TA;
        Z1(I,J)  = TB*TF+TC*SO(J+1)/TA;
        Z2(I,J)  = TB*TE+TC*SNT(I)/TA;
        Z3(I,J)  = (TC/18.)*TF-TD*SO(J+1);
        Z4(I,J)  = (TC/18.)*TE-TD*SNT(I);
    end
end
%   READ INPUT  (REPLACED WITH FUNCTION ARGS)
%
%C     ARBITRARY INPUT SUBROUTINE
%
if IPM > 0
    if prnt2scrn == 1
        fprintf('    INPUT AT ARBITRARY X VALUES\n')
    end
    if IDEN > 0    %(Not symmetric)
        if ILK > 0 %(:INPUT TAU, RHO, RHO)
            if prnt2scrn == 1
                fprintf('   THICKNESS  CAMBER   NOSE RADIUS\n')
                fprintf('%12.6f',TAO,F,RHO)
                fprintf('\n\n')
                fprintf('       X          YT          YC        DYC/DX\n') %
PRINT 31
            end
            RHO = RHO*TAO^2;
```

```
    for I=1:NX
        AT = AT_in(I);
        YT = YT_in(I);
        YC = YC_in(I);
        YCP = YCP_in(I);
        IA=2*NX-I;
        if AT > 0
            if YCP ~= 0
                THT     = atan(YCP*F);
                SA      = sin(THT)*YT*TAO;
                CA      = cos(THT)*YT*TAO;
                CC(I)   = AT-SA;
                Y(I)    = YC*F+CA;
                CC(IA)  = AT+SA;
                Y(IA)   = YC*F-CA;
            elseif YCP == 0
                Y(I)    = YC*F+YT*TAO;
                Y(IA)   = YC*F-YT*TAO;
                CC(I)   = AT;
                CC(IA)  = AT;
            end
        elseif AT == 0
            THT = atan(YCP*F);
            Y(I)= RHO*sin(THT);
            YN  = Y(I);
            CC(I)= -RHO*(1.-cos(THT));
            XN  = CC(I);
        end
        if prnt2scrn == 1
            fprintf('%12.6f',AT,YT,YC,YCP)
            fprintf('\n')
        end
    end
    NX = 2*NX-1;
elseif ILK == 0
    if prnt2scrn == 1
        fprintf('      X              Y\n')
        fprintf('%12.6f', XN, YN)
        fprintf('\n')
        for I = 1:NX
            CC(I)=CC_in(I);
            Y(I) =Y_in(I);
            fprintf('%12.6f',CC(I),Y(I))
            fprintf('\n')
        end
    else
        for I = 1:NX
            CC(I)=CC_in(I);
            Y(I) =Y_in(I);
        end
    end
end
IMS     = 37;
B       = 1-XN;
AWK     = atan(YN/B);
SA      = sin(AWK);
CA      = cos(AWK);
```

```
        AWK     = AWK*180/pi;
        if prnt2scrn == 1
            fprintf('\n\nROTATED AND SHRUNK INPUT\n')
            fprintf('ANGLE OF ROTATION= %9.6f DEG,\n', AWK)
            fprintf('  NOTE: THIS ANGLE WILL BE ADDED TO EACH OF THE INPUT
ANGLES\n')
            fprintf('      X           Y        PHI,DEG\n')
        end
        for I = 1:NX
            CC(I)   = (CC(I)-XN)/B;
            Y(I)    = (Y(I)-YN)/B;
            ALTER   = CC(I);
            CC(I)   = (CC(I)*CA-Y(I)*SA)*CA;
            Y(I)    = (Y(I)*CA+ALTER*SA)*CA;
        end
        ND  = NX-1;
        A   = (NX+1)/2;
        for I=2:ND
            B       = I;
            EE(I)   = 2*CC(I)-1;
            if EE(I) ~= 0
                EE(I)=atan(sqrt(abs(1-EE(I)^2))/EE(I));
                if (B-A) <= 0
                    if (CC(I)-.5) < 0
                        EE(I) = EE(I)+pi;
                    end
                elseif (B-A) > 0
                    if (CC(I)-0.5) < 0
                        EE(I) = pi+abs(EE(I));
                    elseif (CC(I)-0.5) > 0
                        EE(I)=2*pi - EE(I);
                    elseif (CC(I)-0.5) == 0
                        EE(I)=1.5*pi;
                    end
                end
            elseif EE(I) == 0
                if (B-A) == 0
                    error('ERRONEOUS INPUT')
                elseif (B-A) < 0;
                    EE(I)=pi/2;
                elseif (B-A) > 0;
                    EE(I)=1.5*pi;
                end
            end
        end
        EE(1)=0;
        EE(NX)=2*pi;
        for I=1:NX
            A=EE(I)*180/pi;
            if prnt2scrn == 1
                fprintf('%12.6f',CC(I),Y(I),A)
                fprintf('\n')
            end
        end
    elseif IDEN == 0
        if prnt2scrn == 1
            fprintf('      X           Y        PHI,DEG\n')
```

```
    end
    ND=NX-1;
    for I=1:NX
        CC(I)=CC_in(I);
        Y(I)=Y_in(I);
    end
    EE(1)   =0;
    if prnt2scrn == 1
        fprintf('      X           Y        PHI,DEG\n')
        fprintf('%12.6f',CC(1),Y(1),EE(1))
        fprintf('\n')
    end
    for I = 2:ND
        EE(I)   = 2*CC(I)-1;
        if EE(I) ~= 0
            EE(I) = atan(sqrt(abs(1-(EE(I))^2))/EE(I));
            if (CC(I)-0.5) < 0
                EE(I) = EE(I)+pi;

            elseif (CC(I)-0.5) == 0
                EE(I) = pi/2;
            end
        elseif EE(I)==0
            EE(I) = pi/2;
        end
        A   = EE(I)*180./pi;
        if prnt2scrn == 1
            fprintf('%12.6f',CC(I),Y(I),A)
            fprintf('\n')
        end
    end
    EE(NX)  = pi;
    A       = 180;
    if prnt2scrn == 1
        fprintf('%12.6f',CC(NX),Y(NX),A)
        fprintf('\n')
    end
    IMS     = 19;
    AWK     = 0;
end

I       = 1;
CC(1)   = Y(1);
R       = 0;
Y1      = Y(2)-Y(1);
Y2      = Y(3)-Y(1);
Y3      = Y(4)-Y(1);
A       = (Y1*EE(3)-Y2*EE(2))/(EE(2)*EE(3)*(EE(2)-EE(3)));
B       = (Y2*EE(4)-Y3*EE(3))/(EE(4)*EE(3)*(EE(3)-EE(4)));
A3      = (A-B)/(EE(2)-EE(4));
A2      = A-A3*(EE(2)+EE(3));
A1      = Y1/EE(2)-EE(2)*(A2+A3+EE(2));
R       = R + pi/18;
I       = I + 1;
CC(I)   = Y(1)+R*(A1+R*(A2+R*A3));
if (R-EE(2)) <= 0
```

```
    while (R-EE(2)) <= 0
        R       = R + pi/18;
        I       = I + 1;
        CC(I)   = Y(1)+R*(A1+R*(A2+R*A3));
    end
end
NP  = NX-2;
Y1  = Y(ND)-Y(NX);
Y2  = Y(NP)-Y(NX);
N3  = NX-3;
Y3  = Y(N3)-Y(NX);
R   =  0;
X1  = EE(ND)-EE(NX);
X2  = EE(NP)-EE(NX);
X3  = EE(N3)-EE(NX);
A   = (Y1*X2-Y2*X1)/(X1*X2*(X1-X2));
B   = (Y2*X3-Y3*X2)/(X3*X2*(X2-X3));
A3  = (A-B)/(X1-X3);
A2  = A-A3*(X1+X2);
A1  = Y1/X1-X1*(A2+A3*X1);
I   = IMS;
R   = R-pi/18;
I   = I-1;
CC(I) = Y(NX)+R*(A1+R*(A2+R*A3));
A   = R+EE(NX);
if (A-EE(ND)) > 0
    while (A-EE(ND)) > 0
        R   = R-pi/18;
        I   = I-1;
        CC(I) = Y(NX)+R*(A1+R*(A2+R*A3));
        A   = R+EE(NX);
    end
end
for I = 2:IMS
    R = (I-1)*pi/18;
    for J = 2:NP
        if (R-EE(J)) > 0
            if (R-EE(J+1)) < 0
                JP  = J-1;
                X1  = EE(J)-EE(JP);
                X2  = EE(J+1)-EE(JP);
                X3  = EE(J+2)-EE(JP);
                Y1  = Y(J)-Y(JP);
                Y2  = Y(J+1)-Y(JP);
                Y3  = Y(J+2)-Y(JP);
                A   = (Y1*X2-Y2*X1)/(X1*X2*(X1-X2));
                B   = (Y2*X3-Y3*X2)/(X3*X2*(X2-X3));
                A3  = (A-B)/(X1-X3);
                A2  = A-A3*(X1+X2);
                A1  = Y1/X1-X1*(A2+A3*X1);
                R   = R-EE(JP);
                CC(I)   = Y(JP)+R*(A1+R*(A2+R*A3));
            end
        elseif (R-EE(J)) == 0
            CC(I)   = Y(J);
        end
    end
```

```
    end
    if IDEN == 0
        for I = 1:18
            IA      = 38-I;
            CC(IA)  = -CC(I);
        end
    end
    CC(IMS)     = Y(NX);
    if prnt2scrn == 1
        fprintf('\n\nINPUT AT REQUIRED X VALUES\n')
        fprintf('      INDEX        X          YU          YL\n')
    end
    for I = 1:19
        IA      = 38-I;
        Y(I)    = CC(I);
        Y(IA)   = CC(IA);
        J       = I-1;
        Input_at_reqd_x(I,:) = [J X(I) Y(I) Y(IA)];
        if prnt2scrn == 1
            fprintf('%12.6f',J,X(I),Y(I),Y(IA))
            fprintf('\n');
        end
    end
    Y(19)       = 0;
    if prnt2scrn == 1
        fprintf('  NOTE: LE AND TE ORDINATES SET=0\n')
    end
    ABA     = Y(1);
    Y(1)    = 0;
elseif IPM == 0
    AWK = 0;
    if IDEN == 0
        Y(19)   = 0;
        ABA     = Y(1);
        Y(1)    = 0;
        for I = 1:18
            IA      = 38-I;
            Y(IA)   = -Y(I);
        end
    elseif IDEN > 0
        Y(37)   = -Y(1);
        ABA     = Y(1);
        Y(1)    = 0;
    end
end
%
%   CALCULATION OF PROFILE SLOPE AND COTANGENT INTEGRAL
%
if IDEN < 0
    ERROR('ERRONEOUS INPUT');
elseif IDEN == 0
    MAD = 19;
elseif IDEN > 0
    MAD = 36;
end

for I=1:MAD
```

```
    SY(I)   = 0;
    VL(I)   = Y(I)*9;
    for J=1:17
        LA = I-J;
        if LA <= 0;
            LA = 36+LA;
        end
        KB=I+J;
        if (KB-36) > 0;
            KB = KB - 36;
        end
        SY(I) = SY(I) + (Y(LA) - Y(KB))*COT(J);
        VL(I) = VL(I) + (Y(LA) + Y(KB))*COL(J);
    end
end

VL(37) = VL(1);
SY(37) = SY(1);
if IDEN == 0
    for I       = 2:18
        IA      = 38-I;
        VL(IA)  = -VL(I);
        SY(IA)  = SY(I);
    end
end
Y(1)=ABA;
%
%   CALCULATION OF BASE PROFILE VELOCITY AND INCREMENT DUE TO ANGLE OF
%   ATTACK
if prnt2scrn == 1
    fprintf('\n\n                                PROFILE CONSTANTS\n\n');
    fprintf('\t   X\t\t\tY\t\t\tC\t\t\tD\t\t\tE\t\t\t  DY/DPHI\n');
end
for I=1:37
    D2      = sqrt(SY(I)^2 + (SO(I)^2)/4.);
    CC(I)   = (VL(1)-VL(I)-SO(I)/2.)/D2;
    DD(I)   = (SY(I)-SY(1)+(CO(I)-1)/2)/D2;
    AAAAA   = (X(I)-1)/(2*D2)+X(I)*DD(I);
    TA      = -CC(I);
    TB      = -DD(I);
    EE(I)   = D2;
    Profile_Const(I,:) = [X(I) Y(I) TA TB AAAAA SY(I)];
    if prnt2scrn == 1
        fprintf('%12.6f %12.6f %12.6f %12.6f %12.6f %12.6f\n',...
            X(I),Y(I),TA,TB,AAAAA,SY(I));
    end
end
if prnt2scrn == 1
    fprintf('\n\n NON-DIMENSIONAL VELOCITY, V=(C*cos(ALFA)+D*sin(ALFA))*(1-
/+DELTA*sqrt(X-X^2))+DELTA*E )\n')
end
%
%   CALCULATION OF LIFT CURVE SLOPE AND ANGLE OF ZERO LIFT,THEORY
%
P1  = (1+2*SY(1))^2+4*VL(1)^2;
P   = sqrt(P1);
AOL = atan(2*VL(1)/(1+2*SY(1)));
```

```
AXL = AOL*180/pi;
if prnt2scrn == 1
    fprintf(' DCL/D(ALPHA)/2PI (THEORY)=%10.6f\n ANGLE,CL=0 (THEORY) =%10.6f
DEG \n\n\n\n'...
        ,P, AXL)
end
%
%   CALCULATION OF BASE PROFILE VELOCITY AND INCREMENT DUE TO ANGLE OF
%   ATTACK AT INTERMEDIATE POINTS
%
if prnt2scrn == 1
    fprintf('                              PROFILE CONSTANTS\n\n')
    fprintf('                             INTERMEDIATE VALUES\n')
    fprintf('                               UPPER SURFACE\n')
    fprintf('\t\tX\t\t\tC \t\t\tD \t\t\tE \t\tDY/DPHI  \n')
end

for I=1:12
    CD1 = 0.0;
    CD2 = 0.0;
    CD3 = 0.0;
    CD4 = 0.0;
    for J=1:17
        JC = 37-J;
        YT = Y(J+1)-Y(JC);
        YC = Y(J+1)+Y(JC);
        CD1 = CD1+YT*Z1(I,J);
        CD2 = CD2+YC*Z2(I,J);
        CD3 = CD3+YT*Z3(I,J);
        CD4 = CD4+YC*Z4(I,J);
    end
    DYU     = CD3+CD4;
    DYL     = CD3-CD4;
    CTU     = CD1+CD2;
    CTL     = -CD1+CD2;
    TA      = SNT(I)*SNT(I)*.25;
    TB      = .5*(1.+CNT(I));
    ANTRP(I)= DYL;
    E1(1,I) = sqrt(DYU*DYU+TA);
    E1(2,I) = sqrt(DYL*DYL+TA);
    C1(1,I) = (VL(1)-CTU-SNT(I)*.5)/E1(1,I);
    C1(2,I) = (VL(1)-CTL+SNT(I)*.5)/E1(2,I);
    D1(1,I) = (DYU-SY(1)-TB)/E1(1,I);
    D1(2,I) = (DYL-SY(1)-TB)/E1(2,I);
    AAAAA   = (XA(I)-1.)/(2.*E1(1,I))+XA(I)*D1(1,I);
    TA      = -C1(1,I);
    TB      = -D1(1,I);
    Profile_Const_INT_U(I,:) = [XA(I) TA TB AAAAA DYU];
    if prnt2scrn == 1
        fprintf('%12.6f%12.6f%12.6f%12.6f%12.6f\n',...
            XA(I),TA,TB,AAAAA,DYU)
    end
end
if prnt2scrn == 1
    fprintf('\n\n                                  LOWER SURFACE\n')
    fprintf('\t\tX\t\t\tC \t\t\tD \t\t\tE \t\tDY/DPHI  \n')
end
```

```
for I=1:12
    AAAAA   = (XA(I)-1.)/(2.*E1(2,I))+XA(I)*D1(2,I);
    TA      = -C1(2,I);
    TB      = -D1(2,I);
    Profile_Const_INT_L(I,:) = [XA(I) TA TB AAAAA ANTRP(I)];
    if prnt2scrn == 1
        fprintf('%12.6f',XA(I),TA,TB,AAAAA,ANTRP(I))
        fprintf('\n')
    end
end

if ICL == 0
    AOLE    = AOLE+AWK;
    AXL     = AOLE*.017453293;
end
%
%   CALCULATION OF THEORETICAL AND DISTORTED PRESSURE DISTRIBUTION
%
if IPMIN > 0
    if prnt2scrn == 1
        fprintf('\n\n                                MINIMUM PRESSURES\n')
        fprintf('                ETA = %8.6f,',ETA)
        fprintf('   ALFA(CL=0) = %10.6f\n\n',AOLE)
        fprintf('    ALFA        CL          CP MIN      MAX VELOC       X
CM(X=0.25)\n')
    end
end
for I=1:JA
    if IPMIN == 0
        if prnt2scrn==1
            fprintf('\n\n                          PRESSURE DISTRIBUTION\n')
        end
        if IDEN == 0
            if prnt2scrn == 1
                fprintf('                          SYMMETRICAL PROFILE\n\n')
            end
        elseif IDEN > 0
            if prnt2scrn == 1
                fprintf('                         NON-SYMMETRICAL PROFILE\n\n')
            end
        end
        if prnt2scrn == 1
            fprintf('    ALFA        CL          DELTA      sin(ALFA)  LIFT
SLOPE   ALFA,CL=0\n')
        end
    end
    DMALFA  = ALFA(I)+AWK;
    ANG     = DMALFA*pi/180;
    if ICL ~= 0
        CL  = CLE(I);
    elseif ICL == 0
        CL  = 2*pi*ETA*(ANG-AXL);
    end
    DEL     = ANG - AOL - atan(CL/sqrt(39.478418*P1 - CL^2) );
    SA      = sin(ANG);
    CA      = cos(ANG);
    if IPMIN == 0
```

```
        if prnt2scrn == 1
            fprintf('%12.6f',DMALFA,CL,DEL,SA,ETA,AOLE)
            fprintf('\n')
            fprintf('\n\n       X      POTNL VELOC  VISC INCRM  VISC VELOC
POTNL P/Q    VISC P/Q\n')
        end
    end
    APG      = ANG-DEL;
    CAP      = cos(APG);
    SAP      = sin(APG);
    CLINT    = 0.0;
    CDINT    = 0.0;
    CMXINT   = 0.0;
    CMYINT   = 0.0;
    SIGMA    = 0.0;
    for J=1:37
        APG      = ANG-DEL*X(J);
        CAV      = cos(APG);
        SAV      = sin(APG);
        VELP     = abs(CC(J)*CA+DD(J)*SA);
        VELV     = (1-DEL*SO(J)/2.)*((CC(J)-VL(1)/EE(J))*CAV+(DD(J)...
            +(.5+SY(1))/EE(J))*SAV+VL(1)/EE(J)*CAP-(.5+SY(1))/EE(J)*SAP);
        VELV     = abs(VELV);
        ANCR     = VELV-VELP;
        PRESP    = 1-(VELP)^2;
        PRESV    = 1-(VELV)^2;
        ABD      = pi/AN*PRESV;
        ABC      = ABD*SO(J)/2;
        CMXINT   = CMXINT+ABC*(X(J)-.25);
        if IPMIN == 0
            CLINT    = CLINT-ABC;
            if (J-1) > 0
                ABD      = ABD*SY(J);
                CDINT    = CDINT-ABD;
                if (J-37) < 0
                    CMYINT   = CMYINT-ABD*Y(J);
                end
            end
            Press_Dist(J,:) =  [X(J) VELP ANCR VELV PRESP PRESV];
            if prnt2scrn == 1
                fprintf('%12.6f',X(J),VELP,ANCR,VELV,PRESP,PRESV)
                fprintf('\n')
            end
        elseif IPMIN > 0
            if (SIGMA-PRESV) >=0
                SIGMA    = PRESV;
                XMIN     = X(J);
                VMAX     = VELV;
            end
        end
        if (37-J) == 0
            if IPMIN == 0
                if prnt2scrn == 1
                    fprintf('\nINTEGRATED CN=%10.6f\n',CLINT)
%  4508 PRINT 17,CLINT
                    fprintf('INTEGRATED CC=%10.6f\n',CDINT)
%       PRINT 18,CDINT
```

```
                    fprintf('INTEGRATED CM(X)=%10.6f, CW ABT
X=0.25\n',CMXINT)       %       PRINT 19,CMXINT
                    fprintf('INTEGRATED CM(Y)=%10.6f, CW ABT Y=0\n',CMYINT)
%       PRINT 20,CMYINT
%
%   CALCULATION AT INTERMEDIATE POINTS
%
                    fprintf('\n\n                              PRESSURE
DISTRIBUTION\n\n')      %       PRINT 8
                    fprintf('    ALFA         CL           DELTA       sin(ALFA)
LIFT SLOPE   ALFA,CL=0\n')    %       PRINT 12
                    fprintf('%12.6f', DMALFA,CL,DEL,SA,ETA,AOLE)
%       PRINT 2, DMALFA,CL,DEL,SA,ETA,AOLE
                    fprintf('\n\n                              INTERMEDIATE
VALUES\n')          %       PRINT 14
% C
% C     UPPER SURFACE NOSE VELOCITY
% C
                    fprintf('\n                               UPPER SURFACE \n')
%       PRINT 15
                    fprintf('      X     POTNL VELOC  VISC INCRM  VISC VELOC
POTNL P/Q    VISC P/Q   \n') %       PRINT 13
                end
            end
            for K=1:12
                APG     = ANG-DEL*XA(K);
                CAV     = cos(APG);
                SAV     = sin(APG);
                VELP    = abs(C1(1,K)*CA+D1(1,K)*SA);
                VELV    = abs((1.-DEL*SNT(K)/2.)*((C1(1,K)-VL(1)...
                    /E1(1,K))*CAV+(D1(1,K)+(.5+SY(1))/E1(1,K))*SAV+...
                    (VL(1)*CAP-(.5+SY(1))*SAP)/E1(1,K)));
                ANCR    = VELV-VELP;
                PRESP   = 1 -(VELP)^2;
                PRESV   = 1 -(VELV)^2;
                if IPMIN > 0
                    if (SIGMA-PRESV) >= 0
                        SIGMA   = PRESV;
                        XMIN    = XA(K);
                        VMAX    = VELV;
                    end
                elseif IPMIN == 0
                    Press_Dist_INT_U(K,:) = [XA(K) VELP ANCR VELV PRESP
PRESV];
                    if prnt2scrn == 1
                        fprintf('%12.6f', XA(K),VELP,ANCR,VELV,PRESP,PRESV)
                        fprintf('\n')
                    end
                end
            end
%
%     LOWER SURFACE NOSE VELOCITY
%
            if IPMIN == 0
                if prnt2scrn == 1
                    fprintf('\n                               LOWER SURFACE\n')
```

```
                    fprintf('      X      POTNL VELOC  VISC INCRM  VISC VELOC
 POTNL P/Q     VISC P/Q\n')
                end
            end
            for K=1:12
                APG      = ANG-DEL*XA(K);
                CAV      = cos(APG);
                SAV      = sin(APG);
                VELP     = abs(C1(2,K)*CA+D1(2,K)*SA);
                VELV     = abs((1.+DEL*SNT(K)/2.)*((C1(2,K)-VL(1)...
                    /E1(2,K))*CAV+(D1(2,K)+(.5+SY(1))/E1(2,K))...
                    *SAV+(VL(1)*CAP-(.5+SY(1))*SAP)/E1(2,K)));
                ANCR     = VELV - VELP;
                PRESP    = 1-VELP^2;
                PRESV    = 1-VELV^2;

                if IPMIN > 0
                    if (SIGMA-PRESV) >= 0
                        SIGMA    = PRESV;
                        XMIN     = XA(K);
                        VMAX     = VELV;
                    end
                elseif  IPMIN == 0
                    Press_Dist_INT_L(K,:) = [XA(K) VELP ANCR VELV PRESP
PRESV];
                    if prnt2scrn == 1
                        fprintf('%12.6f',XA(K),VELP,ANCR,VELV,PRESP,PRESV)
                        fprintf('\n')
                    end
                end
            end
            if IPMIN > 0
                Min_Press(I,:) = [DMALFA CL SIGMA VMAX XMIN CMXINT];
                if prnt2scrn == 1
                    fprintf('%12.6f',DMALFA,CL,SIGMA,VMAX,XMIN,CMXINT)
                    fprintf('\n')
                end
            end
        end
    end
end
%   END OF BROCKETT'S CODE

%   Combine Profile Constant Data in Order
i = 1;
j = length(Profile_Const_INT_U);
step = 1;
while j>0 && i<length(Profile_Const)
    A = Profile_Const(i,1);
    B = Profile_Const_INT_U(j,1);
    if A > B
        Profile_Comb(step,:) = Profile_Const(i,:);
        i = i + 1;
        step = step + 1;
    elseif B > A
```

```
        Profile_Comb(step,:) = [Profile_Const_INT_U(j,1) 0
Profile_Const_INT_U(j,2:end)];
        j = j -1;
        step = step + 1;
    end
end
j = 1;
while j <= length(Profile_Const_INT_L) || i <= length(Profile_Const)
    if B < A && j <= length(Profile_Const_INT_L)
        Profile_Comb(step,:) = [Profile_Const_INT_L(j,1) 0
Profile_Const_INT_L(j,2:end)];
        j = j + 1;
        if j <= length(Profile_Const_INT_L)
            B = Profile_Const_INT_L(j,1);
        end
        step = step + 1;
    elseif A < B || j > length(Profile_Const_INT_L)
        Profile_Comb(step,:) = Profile_Const(i,:);
        i = i + 1;
        if i <= length(Profile_Const)
            A = Profile_Const(i,1);
        end
        step = step + 1;
    end
end

%   Combine Pressure Distribution Data in Order
if IPMIN == 0
    i = 1;
    j = length(Press_Dist_INT_U);
    step = 1;
    while j>0 && i<length(Press_Dist)
        A = Press_Dist(i,1);
        B = Press_Dist_INT_U(j,1);
        if A > B
            Press_Comb(step,:) = Press_Dist(i,:);
            i = i + 1;
            step = step + 1;
        elseif B > A
            Press_Comb(step,:) = Press_Dist_INT_U(j,:);
            j = j -1;
            step = step + 1;
        end
    end
    j = 1;
    while j <= length(Press_Dist_INT_L) || i <= length(Press_Dist)
        if B < A && j <= length(Press_Dist_INT_L)
            Press_Comb(step,:) = Press_Dist_INT_L(j,:);
            j = j + 1;
            if j <= length(Press_Dist_INT_L)
                B = Press_Dist_INT_L(j,1);
            end
            step = step + 1;
        elseif A < B || j > length(Press_Dist_INT_L)
            Press_Comb(step,:) = Press_Dist(i,:);
            i = i + 1;
            if i <= length(Press_Dist)
```

```
                A = Press_Dist(i,1);
            end
            step = step + 1;
        end
    end

end

%   Calls XFOIL calculate pressure distribution and imports data
cmd = ['xfoil.exe LOAD trefxy OPER ALFA ', num2str(ALFA),...
    ' OPER CPWR CPX'];
system(cmd);
fid = fopen('CPX');
xfoil_data_in = textscan(fid, '%f64  %f64', 'headerlines', 1);
fclose(fid);
xfoil_x = xfoil_data_in{1};
xfoil_cp = xfoil_data_in{2};

%   Plots Data from Brockett and Xfoil on same graph as Trefftz.
open('Trefftz.fig')
plot(Press_Comb(:,1), -Press_Comb(:,5), 'ro-')
plot(Press_Comb(:,1), -Press_Comb(:,6), 'r.-')
plot(xfoil_x, -xfoil_cp, 'bx-')
legend('Conformal Transformation','Brockett Method (Inviscid)',...
    'Brockett Method (Viscid)','XFOIL Calc')
savefile = ['K-T,alfa=',num2str(ALFA),];
saveas(gcf, savefile);
```

Appendix C: `Brockett.m` Variable Descriptions

Function variables are specified as those variable that instruct the MATLAB version of Brockett's work how to process the data, which data will be input, and how the data is formatted. The following paragraph provide a brief description of these variables, and how they are used in Brockett.m

JA: Specifies the number of angles of attack that will be calculated. Although not specified directly by the user, the scripts in Appendix C: calculate this value based on the number of inputs for the ALFA vector. It is recommended that multiple angles of attack be processed individually, since output for multiple angles is only printed to the screen.

KA: This variable was used originally by Brockett for processing multiple jobs. This functionality is not used in the MATLAB version. Separate jobs are specified by the appropriate script file (i.e. ARB_IN.m, REQ_IN.m, etc), in which the use specifies the job parameters. The job specification script is specified by the "run" command in the beginning of Brockett.m.

IPMIN: Specifies whether or not minimum pressure distribution data will be reported. Should normally be set to 0. Plotting and screen output will not be available if minimum pressure data is not calculated. If screen output is not desired, use the 'print2scr' variable below.

ALFA: Vector of angles of attack to be calculated. Normally a single value. Note: If multiple angles of attack are specified, output for each angle of attack will be only to screen in tabular format. Plots for each angle of attack will not be generated.

IDEN: Specifies whether input data points are for a symmetric foil shape. Symmetric data is designated by IDEN = 0, or non-symmetric IDEN = 1. Symmetric data can either be in the format of offsets, or camber and thickness data.

IPM: IPM = 0 specifies that ordinate information will be input at required locations. Required locations are specified by:

$$x_m = \frac{1}{2}\left(1+\cos\frac{m\cdot\pi}{18}\right), m = 0...18$$

$$x_{36-m} = x_m$$

When input at required station is specified, only the upper surface, Y_0 through Y_{17}, are specified for symmetric foils. All others, Y_0 through Y_{35} are specified. If IPM = 1 (arbitrary input locations), user must define the number of locations (NX) that will be input.

ICL: ICL = 1 if experimental lift coefficient is specified rather that angle of zero lift (AOLE) and lift-slope curve coefficient (ETA). If ICL = 0, AOLE and ETA must be set to zero. Otherwise, if ICL = 0, AOLE and ETA must be specified. AOLE and ETA values do not affect inviscid calculation, and are only used for the empirical modification to account for viscous effects. If unknown, may be set to 0 for inviscid calculations.

CLE: If ICL = 1, user must specify experimental lift coefficients corresponding to input angles of attack (ALFA). ALFA and CLE vector must be of equal length.

ILK: ILK = 0 specifies that offsets will be input in X, Y format. CC is vector of x-values, and Y is vector of corresponding y-offset values. For symmetric foils, give only upper surface from trailing edge to nose. Last point must be (0, 0). For non-symmetric foils, must specify XN and YN which are x and y ordinates of nose location. Sample scripts locate this point automatically from input vectors CC and Y. Order for non-symmetric foils must be from trailing edge (1, X.X), along upper surface to (XN, YN), and back to trailing edge (1.0, X.X).

ILK = 1 specifies that foil surface locations will be specified by thickness ratio (TAO), camber (F), and leading edge radius (RO). Input required is x-location (AT_in), thickness value (YT_in), camber value (YC_in), and camberline slope

(YCP_in). AT_in, YT_in, YC_in, and YCP_in start at trailing edge(x=1) and go to leading edge (x=0), and are of length NX.

foil_name: User specified foil designation. Used for plot legend and/or titles.

Appendix D: Sample Input Scripts for `Brockett.m`.

REQD_IN.m

```
%   Written by Chris Peterson

%   This file inputs data to be processed by MATLABs version of
%   Brockett's Thesis program.  The data is from fig 3b, pg 67.
%   Format is for input at required stations, angle of attack specified

ALFA     = 4.09;          %ANGLE OF ATTACK
JA       = length(ALFA);  %NUMBER OF ANGLES
IPMIN    = 0;             %0:Report Data, 1: No Data
IDEN     = 0;             %0:SYMM, 1:NONSYMM
IPM      = 0;             %0:STD INPUT LOCATIONS, 1:ARBITRARY STATIONS
ICL      = 0;             %0:USE ALPHA, 1: USE INPUT CLE
AOLE     = 0;             %Experimental angle of zero lift
ETA      = 0.959;         %Lift curve-slope coeff
foil_name = 'RAE-101,00-10';

%   Y_in is array of ordinates, trailing edge to leading edge along upper
%   surface, then leading edge to trailing edge along lower surface.  Only
%   Yn n=0->17 for symmetric foils.  All others n=0->35
Y_in = [0  .00068    .0027    .00599    .01046    .01597    .02236...
    .029345    .03636    .04267    .047445    .04985    .04885...
    .04475    .038405    .03034    .02093    .01071 ];

%   Formats Y for both surfaces
Y = [Y_in 0 fliplr(-Y_in)];

%Plot input
figure(1)
axis equal;
hold on;
plot((1+cos(0:2*pi/36:2*pi))/2, Y, 'g.')
plot((1+cos(0:2*pi/36:2*pi))/2, Y, 'g')
legend('Input to Brockett')
title(foil_name)

NX = 0;              %Req'd to set variable EE(used in Arb Input)
```

ARB_IN.m

```
%   Written by Chris Peterson

%   This file inputs data to be processed by the MATLAB version of
%   Brockett's Thesis program.  The data is from fig 4b, pg 69.
%   Format is for arbitrary input stations, non-symmetrical,
%   lift coefficient specified

IPMIN    = 0;            %0:Report Data, 1: No Data
IDEN     = 1;            %0:SYMM, 1:NONSYMM
IPM      = 1;            %0:STD INPUT LOCATIONS, 1:ARBITRARY STATIONS
ICL      = 1;            %0:USE ALPHA, 1: USE INPUT CLE
  CLE    = [-0.14 .15 .44 .73 .97 1.16 1.26 1.34 1.11];          %Lift Coeff
  ETA    = 0;
  AOLE   = 0;
ILK      = 0;            %0:INPUT STA X,Y  1:INPUT TAU, RHO, RHO
ALFA     = [-7.6 -4.5 -1.5 1.5 4.7 8.0 9.7 11.4 16.2];    %ANGLE OF ATTACK
JA       = length(ALFA); %NUMBER OF ANGLES

foil_name = 'CLARK Y, NACA RPT 460';

%   Coordinates.  Must have same number on upper surface as lower surface.
%   For symmetrical foil, give only upper surface (last point 0,0 for
%   symmetrical foils)
CC = [1 .992404 .95 .9 .8 .7 .6 .5 .4 .3 .2 .15 .1 .075 .05 .025 .0125...
    0 .0125 .025 .05 .075 .1 .15 .2 .3 .4 .5 .6 .7 .8 .9 .95 .992404 1];
Y  = [.0006    .0027    .0144    .0273    .0515    .0728    .0907...
    .1043    .1131    .1162    .1126    .1057    .0950    .0873    .0777...
    .0637    .0532    .0354    .0180    .0136    .0085    .0053...
    .0033    .0008    -.0005    -.0006    -.0006    -.0006    -.0006...
    -.0006  -.0006    -.0006    -.0006    -.0006    -.0006];

[XN, indx] = min(CC);
YN = Y(indx);
NX = length(CC);

%Plot input
figure()
axis equal;
hold on;
plot(CC,Y, 'k.', CC,Y, 'k');
legend('Input to Brockett')
title(foil_name)
```

KT_IN.m

```
%   Written by Chris Peterson

%   This file inputs data to be processed by the MATLAB version of
%   Brockett's Thesis code (Brockett.m).  This file reads input generated
%   by the Karman-Trefftz foil Conformal Transformation script
(CnfrmlTrans.m).

IPMIN    = 0;            %0:Report Data, 1: No Data
IDEN     = 1;            %0:SYMM, 1:NONSYMM
IPM      = 1;            %0:STD INPUT LOCATIONS, 1:ARBITRARY STATIONS
ICL      = 0;            %0:USE ALPHA, 1: USE INPUT CLE
if ICL == 0              %Viscous calcs require experimental data input
    AOLE     = 0;        %Experimental angle of zero lift
    ETA      = 0;        %Lift-curve slope coefficient
elseif ICL ==1
    CLE      = 0;        %Experimental lift coefficient
    AOLE     = 0;        %AOLE and ETA must be set to 0 if ICL = 1
    ETA      = 0;        %AOLE and ETA must be set to 0 if ICL = 1
end
ILK       = 0;           %0:INPUT STA X,Y  1:INPUT TAU, RHO, RHO
foil_name = 'Karman-Trefftz';

load('x_output.mat', '-mat');%Opens data generated by CnfrmlTrans.m

CC       = x_spl;        %Read in x data from K-T foil ordinates
Y        = y_spl;        %Read in y data from K-T foil ordinates
[XN, xn_ind] = min(CC); %Find nose x location and index
NX       = length(CC);   %NUMBER OF STATIONS
YN       = Y(xn_ind);    %Specifies nose y location
ALFA     = alpha_deg;    %ANGLE OF ATTACK (FOR DESIRED PRESSURE DIST)
JA       = length(ALFA); %NUMBER OF ANGLES

%Plot input
figure()
axis equal;
hold on;
plot(CC,Y, 'k.', CC,Y, 'k');
legend('Input to Brockett')
title(foil_name)

%Compare to Karman-Trefftz foil? 1-yes, 0-no
comp2kt      = 1;    %Opens previous trefftz plot and plots new data
```

Brock_IN.m

```
%   Written by Chris Peterson

%   This file inputs data to be processed by MATLAB version of
%   Brockett's Thesis program.  The data is from Brockett's published
%   minimum pressure envelopes for NACA foils (DTMB Report 1780, pg 14).

IPMIN   = 0;            %0:Report Data, 1: No Data
IDEN    = 1;            %0:SYMM, 1:NONSYMM
IPM     = 1;            %0:STD INPUT LOCATIONS, 1:ARBITRARY STATIONS
ICL     = 1;            %0:USE ALPHA, 1: USE INPUT CLE
  ETA   = 0;
  AOLE  = 0;
ILK     = 1;            %0:INPUT STA X,Y  1:INPUT TAU, RHO, RHO
ALFA    = 0;            %ANGLE OF ATTACK
JA      = length(ALFA); %NUMBER OF ANGLES

foil_name = 'NACA 66 (Mod), a=0.8';

TAO =   0.12;
F   =   0.06;
RHO =   .448;
AT_in  =   fliplr([0 0.007596 0.030154 0.066987 .116978 .178606 .25 .32899
.413176...
    .5 .586824 .671010 .75 .821394 .883022 .933013 .969846 .992404 1]);
YT_in  =   fliplr([0 .0817 .1608 .2388 .3135 .3807 .4363 .4760 .4972 .4962
.4712...
    .4247 .3612 .2872 .2108 .1402 .0830 .0462 .0333]);
YC_in  =   fliplr([0 .06006 .18381 .33684 .49874 .65407 .79051 .89831 .96994
1 ...
    .98503 .92306 .81212 .63884 .42227 .23423 .09982 .02365 0]);
YCP_in =   fliplr([7.1485 6.6001 4.7712 3.6751 2.8681 2.2096 1.6350 1.1071
.6001 ...
    .0914 -.4448 -1.0483 -1.8132 -3.1892 -3.7243 -3.7425 -3.5148 -3.2028...
    -3.0025]);

NX  = length(AT_in);
CLE = 2*pi*(1-0.83*TAO)*(deg2rad(ALFA) + 2.05*F);          %Lift Coeff

nose_rad = RHO*TAO^2;


comp2kt = 0;
```

Appendix E: `Brockett.m` Sample Output

PROFILE CONSTANTS

X	Y	C	D	E	DY/DPHI
1.000000	0.000000	0.000000	0.000000	0.000000	0.001998
0.992404	0.000680	0.890691	0.026481	-0.069870	0.007287
0.969846	0.002700	0.933471	0.096858	-0.181740	0.015520
0.933013	0.005990	0.958680	0.186467	-0.307426	0.022185
0.883022	0.010460	0.978938	0.279256	-0.427845	0.028863
0.821394	0.015970	0.997232	0.380868	-0.545076	0.034145
0.750000	0.022360	1.016276	0.490314	-0.655257	0.038834
0.671010	0.029345	1.038498	0.615677	-0.761927	0.040635
0.586824	0.036360	1.062833	0.761407	-0.865047	0.039076
0.500000	0.042670	1.088804	0.937296	-0.967603	0.032369
0.413176	0.047445	1.114413	1.150647	-1.070719	0.021690
0.328990	0.049850	1.139587	1.422610	-1.182065	0.004568
0.250000	0.048850	1.147911	1.771430	-1.308325	-0.015548
0.178606	0.044750	1.144379	2.222439	-1.465799	-0.030556
0.116978	0.038405	1.137954	2.859395	-1.696827	-0.041660
0.066987	0.030340	1.125485	3.863976	-2.087954	-0.050476
0.030154	0.020930	1.087640	5.708723	-2.863493	-0.056743
0.007596	0.010710	0.945091	9.983659	-4.773447	-0.060158
0.000000	0.000000	0.000000	17.144979	-8.056392	-0.062063
0.007596	-0.010710	-0.945091	9.983659	-4.773447	-0.060158
0.030154	-0.020930	-1.087640	5.708723	-2.863493	-0.056743
0.066987	-0.030340	-1.125485	3.863976	-2.087954	-0.050476
0.116978	-0.038405	-1.137954	2.859395	-1.696827	-0.041660
0.178606	-0.044750	-1.144379	2.222439	-1.465799	-0.030556
0.250000	-0.048850	-1.147911	1.771430	-1.308325	-0.015548
0.328990	-0.049850	-1.139587	1.422610	-1.182065	0.004568
0.413176	-0.047445	-1.114413	1.150647	-1.070719	0.021690
0.500000	-0.042670	-1.088804	0.937296	-0.967603	0.032369
0.586824	-0.036360	-1.062833	0.761407	-0.865047	0.039076
0.671010	-0.029345	-1.038498	0.615677	-0.761927	0.040635
0.750000	-0.022360	-1.016276	0.490314	-0.655257	0.038834
0.821394	-0.015970	-0.997232	0.380868	-0.545076	0.034145
0.883022	-0.010460	-0.978938	0.279256	-0.427845	0.028863
0.933013	-0.005990	-0.958680	0.186467	-0.307426	0.022185
0.969846	-0.002700	-0.933471	0.096858	-0.181740	0.015520
0.992404	-0.000680	-0.890691	0.026481	-0.069870	0.007287
1.000000	0.000000	0.000000	0.000000	0.000000	0.001998

NON-DIMENSIONAL VELOCITY, V=(C*cos(ALFA)+D*sin(ALFA))*(1-/+DELTA*sqrt(X-X^2))+DELTA*E)
DCL/D(ALPHA)/2PI (THEORY)= 1.003996
ANGLE,CL=0 (THEORY) = 0.000000 DEG

PROFILE CONSTANTS

INTERMEDIATE VALUES

UPPER SURFACE

X	C	D	E	DY/DPHI
0.000076	0.161276	16.982900	-7.981671	-0.062038
0.000305	0.313757	16.522600	-7.769532	-0.061966
0.000685	0.450819	15.831154	-7.451069	-0.061848
0.001218	0.569054	14.992649	-7.065239	-0.061689
0.001903	0.667997	14.085384	-6.648289	-0.061492
0.002739	0.749135	13.169455	-6.228004	-0.061265
0.003727	0.814891	12.284473	-5.822656	-0.061012
0.004866	0.867893	11.452912	-5.442571	-0.060740
0.006156	0.910580	10.685001	-5.092396	-0.060456
0.011852	1.005922	8.501872	-4.103034	-0.059397
0.017037	1.043992	7.345885	-3.584794	-0.058606
0.023142	1.069452	6.436098	-3.181521	-0.057745

LOWER SURFACE

X	C	D	E	DY/DPHI
0.000076	-0.161276	16.982900	-7.981671	-0.062038
0.000305	-0.313757	16.522600	-7.769532	-0.061966
0.000685	-0.450819	15.831154	-7.451069	-0.061848
0.001218	-0.569054	14.992649	-7.065239	-0.061689
0.001903	-0.667997	14.085384	-6.648289	-0.061492
0.002739	-0.749135	13.169455	-6.228004	-0.061265
0.003727	-0.814891	12.284473	-5.822656	-0.061012
0.004866	-0.867893	11.452912	-5.442571	-0.060740
0.006156	-0.910580	10.685001	-5.092396	-0.060456
0.011852	-1.005922	8.501872	-4.103034	-0.059397
0.017037	-1.043992	7.345885	-3.584794	-0.058606
0.023142	-1.069452	6.436098	-3.181521	-0.057745

PRESSURE DISTRIBUTION
SYMMETRICAL PROFILE

ALFA	CL	DELTA	sin(ALFA)	LIFT SLOPE	ALFA,CL=0
4.090000	0.430129	0.003146	0.071323	0.959000	0.000000

X	POTNL VELOC	VISC INCRM	VISC VELOC	POTNL P/Q	VISC P/Q
1.000000	0.000000	0.000000	0.000000	1.000000	1.000000
0.992404	0.890311	-0.000269	0.890042	0.207346	0.207825
0.969846	0.938002	-0.000877	0.937124	0.120153	0.121798
0.933013	0.969537	-0.001532	0.968005	0.059997	0.062965
0.883022	0.996363	-0.002161	0.994201	0.007261	0.011564
0.821394	1.021857	-0.002763	1.019094	-0.044191	-0.038552
0.750000	1.048659	-0.003318	1.045341	-0.099685	-0.092737
0.671010	1.079765	-0.003835	1.075931	-0.165893	-0.157627
0.586824	1.114433	-0.004305	1.110128	-0.241960	-0.232384
0.500000	1.152882	-0.004731	1.148151	-0.329138	-0.318250
0.413176	1.193643	-0.005110	1.188533	-0.424784	-0.412611
0.328990	1.238150	-0.005460	1.232689	-0.533015	-0.519523
0.250000	1.271331	-0.005780	1.265552	-0.616284	-0.601622
0.178606	1.299976	-0.006129	1.293847	-0.689939	-0.674040
0.116978	1.338998	-0.006662	1.332336	-0.792914	-0.775119
0.066987	1.398210	-0.007654	1.390556	-0.954992	-0.933647
0.030154	1.492035	-0.009812	1.482224	-1.226170	-1.196987
0.007596	1.654753	-0.015487	1.639265	-1.738206	-1.687191
0.000000	1.222837	-0.025387	1.197451	-0.495331	-0.433888
0.007596	0.230616	0.015110	0.245726	0.946816	0.939619
0.030154	0.677705	0.009398	0.687103	0.540716	0.527890
0.066987	0.847027	0.007264	0.854291	0.282546	0.270187
0.116978	0.931114	0.006319	0.937434	0.133026	0.121218
0.178606	0.982953	0.005850	0.988803	0.033804	0.022270
0.250000	1.018643	0.005575	1.024218	-0.037633	-0.049022
0.328990	1.035219	0.005338	1.040558	-0.071679	-0.082760
0.413176	1.029507	0.005071	1.034578	-0.059885	-0.070351
0.500000	1.019180	0.004772	1.023952	-0.038728	-0.048479
0.586824	1.005820	0.004421	1.010241	-0.011675	-0.020588
0.671010	0.991941	0.004020	0.995961	0.016053	0.008062
0.750000	0.978717	0.003565	0.982282	0.042113	0.035122
0.821394	0.967527	0.003062	0.970590	0.063891	0.057956
0.883022	0.956528	0.002504	0.959032	0.085055	0.080258
0.933013	0.942939	0.001906	0.944844	0.110867	0.107269
0.969846	0.924185	0.001268	0.925453	0.145882	0.143537
0.992404	0.886534	0.000656	0.887190	0.214058	0.212894
1.000000	0.000000	0.000000	0.000000	1.000000	1.000000

INTEGRATED CN= 0.425593
INTEGRATED CC= -0.030299
INTEGRATED CM(X)= 0.002425, CW ABT X=0.25
INTEGRATED CM(Y)= -0.001200, CW ABT Y=0

PRESSURE DISTRIBUTION

ALFA	CL	DELTA	sin(ALFA)	LIFT SLOPE	ALFA,CL=0
4.090000	0.430129	0.003146	0.071323	0.959000	0.000000

INTERMEDIATE VALUES

UPPER SURFACE

X	POTNL VELOC	VISC INCRM	VISC VELOC	POTNL P/Q	VISC P/Q
0.000076	1.372143	-0.025188	1.346954	-0.882775	-0.814286
0.000305	1.491405	-0.024563	1.466842	-1.224289	-1.151624
0.000685	1.578802	-0.023607	1.555195	-1.492615	-1.418630
0.001218	1.636931	-0.022440	1.614491	-1.679543	-1.606580
0.001903	1.670913	-0.021175	1.649737	-1.791949	-1.721633
0.002739	1.686517	-0.019899	1.666619	-1.844340	-1.777618
0.003727	1.688985	-0.018667	1.670318	-1.852671	-1.789962
0.004866	1.682543	-0.017513	1.665029	-1.830950	-1.772323
0.006156	1.670351	-0.016452	1.653899	-1.790074	-1.735383
0.011852	1.609742	-0.013469	1.596273	-1.591269	-1.548088
0.017037	1.565266	-0.011923	1.553343	-1.450057	-1.412873
0.023142	1.525772	-0.010735	1.515037	-1.327982	-1.295338

LOWER SURFACE

X	POTNL VELOC	VISC INCRM	VISC VELOC	POTNL P/Q	VISC P/Q
0.000076	1.050412	-0.025123	1.025289	-0.103366	-0.051217
0.000305	0.865489	-0.024437	0.841053	0.250928	0.292630
0.000685	0.679460	-0.023425	0.656035	0.538334	0.569618
0.001218	0.501721	-0.022211	0.479510	0.748276	0.770070
0.001903	0.338321	-0.020906	0.317415	0.885539	0.899248
0.002739	0.192062	-0.019597	0.172465	0.963112	0.970256
0.003727	0.063355	-0.018340	0.045015	0.995986	0.997974
0.004866	0.048822	0.017165	0.065988	0.997616	0.995646
0.006156	0.146171	0.016087	0.162258	0.978634	0.973672
0.011852	0.396978	0.013070	0.410048	0.842409	0.831861
0.017037	0.517400	0.011514	0.528914	0.732298	0.720250
0.023142	0.607684	0.010321	0.618006	0.630720	0.618069

Appendix F: Modified XFOIL User Guide.

The version of XFOIL used for this project was modified with the following goals:

1) Eliminate the need for interactive user interface, such that the program execution could be automated and called by an external program.
2) Allow the data resulting from XFOIL calculations to be read and imported by an external program.
3) Remove internal plot functions such that the only output would be in the form of data files.

These goals were accomplished by the following changes:

1) Elimination of the menu prompts requiring user input that directed the execution of the original XFOIL.
2) Elimination of all plotting functions by removing them from the source code.
3) Modification of the source code to write desired output to data files.

In order to become familiar with the operation of XFOIL, the users should first download and execute the official release version of XFOIL from the website http://web.mit.edu/drela/Public/web/xfoil/. Version 6.96 was used for this project. Due to the menu interface and interactive input, it is easier to become familiar with the functionality of XFOIL using the original version. In addition, it is recommended that the user read the `sessions.txt` and `xfoil_doc.txt` to acquaint themselves with the operation and capabilities of XFOIL.

This appendix will describe how to utilize the modified version of XFOIL used in this project. For the remainder of this appendix "XFOIL" will refer to the modified version of XFOIL used for this project.

XFOIL was written and compiled to built as an executable, such that it could either be executed from the command prompt, or by a function call from an external program. For this project, MATLAB was used as the main program, instructing

XFOIL to conduct desired calculations and write desired data to files, which were then read into MATLAB variables.

The first major change to XFOIL that was implemented is that all operational instructions are required to be input as command line arguments to the function call. All desired XFOIL operations are required to be entered as command line arguments at the time of execution. Upon completion of processing command line arguments, XFOIL will terminate and will not accept additional instructions.

Commands and menus instructing XFOIL on how to process data were maintained in the same menu structure as the original XFOIL program. However, rather than being input by the user at a menu prompt within the program, they are input at the command line prior to execution. XFOIL reads all command line arguments sequentially, processing them separately until completion. The following is an example of an XFOIL execution:

```
c:\xfoil NACA 4415 OPER ALFA 5 OPER CPWR output QUIT
```

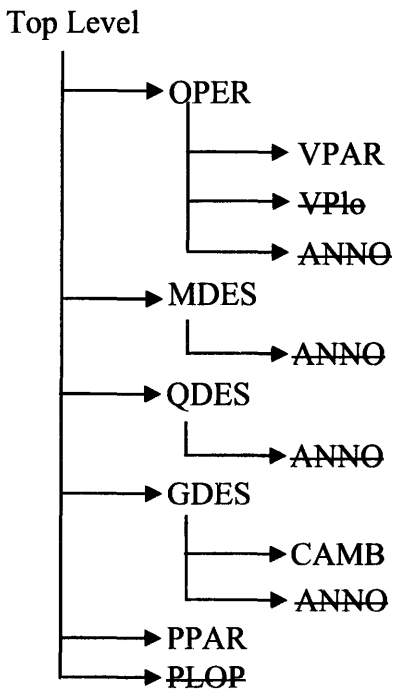
`xfoil` is the function call to execute the XFOIL program, and the rest of the items are all command line arguments read in by XFOIL. There are three types of command line arguments: Menus, commands, and command arguments. All menus and commands consist of four (or less) letters, which must be CAPITALIZED. `OPER` is an example of a menu item, instructing XFOIL that the next command line argument is located under the `OPER` menu. The argument following `OPER` must be either a command or another submenu available under the `OPER` menu.

Commands instruct XFOIL how to execute. For example, commands can specify values for items such as angle of attack or Reynolds Number, or they may instruct XFOIL to perform a calculation, such as determining the minimum pressure coefficient. The first command above is `NACA`, which instructs XFOIL that the internal definition for a NACA foil shape will be used. The next command is `ALFA`,

specifying the angle of attack. The final command is `CPWR`, instructing XFOIL to calculate the pressure coefficient distribution along the foil, and write the distribution data to a file. It should be noted that `NACA` is available under the top level menu of the original XFOIL program, and both `ALFA` and `CPWR` are available under the `OPER` menu. Upon completion of command execution, XFOIL will return to the top menu, rather than in the current menu, as the original XFOIL. For this reason, `OPER` must be entered prior to the `CPWR` command.

For some commands to execute, they require command arguments to be specified. Command arguments are different from command line arguments. Command line arguments are all of the items following `xfoil`. Command arguments are items required by XFOIL commands to be executed. In the above example, `4415` is an argument for the `NACA` command, `5` is an argument for the `ALFA` command, and `output` is an argument for the `CPWR` command. Command arguments may be integer, character string, real, or filename, as required by the command. Another difference from the original XFOIL is that when reading in command arguments, XFOIL does not recognize filename extensions. For example, `output.txt` would not be a valid command argument for the `CPWR` command. As mentioned, the menu structure and many of the command items were retained from the original XFOIL. The XFOIL 6.9 user primer provides a good description of commands, command arguments, and menus.

To the right is the XFOIL menu structure. The original XFOIL contained all menus, but the striked items were removed since they contained only plot commands. Menus are similar to directories, which contain submenu items and commands.



Although several of the original commands were retained in the XFOIL code, only certain commands used for this project were tested. Other command may work, but were not specifically tested or used in this project. In addition, some commands that were not included may only require slight modification to be used. A description of how the XFOIL code processes command line arguments will be included at the end of this appendix to aid in future implementations.

Commands employed and included in the XFOIL code will be described below. Commands will be grouped by their corresponding menu. Only commands deliberately incorporated will be discussed, although other commands may still be present in the source code. Descriptions will consist of the command, command argument designator (i-Integer, r-Real, f-Filename, s-Character String), and a brief description taken from the XFOIL 6.9 user primer. Command argument designators specify that the command requires a command argument, and describe the type of argument expected by XFOIL.

(Top Level)

`REVE`	Reverse written-airfoil node ordering
`LOAD f`	Read buffer airfoil from coordinate file
`NACA i`	Set NACA 4,5-digit airfoil and buffer airfoil
`NORM`	Buffer airfoil normalization toggle
`PANE`	Set current-airfoil panel nodes based on curvature.
`QUIT`	Exit program

OPER – Direct operating point(s)

`VISC r`	Toggle Inviscid/Viscous Mode (Reynolds number is argument)
`RE r`	Change Reynolds Number
`ITER i`	Change viscous solution iteration limit
`INIT`	Toggle BL initialization flag
`ALFL rrr`	NEW FUNCTION. Instructs XFOIL to calculate the minimum pressure coeffient over a range of angles of attack, and writes the values for minimum pressure coefficient and chordwise position for each angle of attack to a single file called `CPMINARRAY.TXT`. Both inviscid and viscous results are included, if applicable. Format is: `ALFL (Lower Angle) (Angle Increment) (Upper Angle).`
`ALFA r`	Prescribe alpha (angle of attack). Command may also be specified by `A` rather than `ALFA`

CLI	r	Prescribe inviscid lift coefficient
CL	r	Prescribe lift coefficient
DUMP	f	Output Ue, Dstar, Theta, Cf, vs s,x,y to file
CPWR	f	Output x vs Cp to file
CPMN		Output location and value of minimum surface Cp (inviscid and viscous if applicable) to file named CPMIN.TXT

OPER VPAR – Change BL parameter(s)

XTR	rr	Change trip positions Xtr/C
N	r	Change critical amplification exponent Ncrit
VACC	r	Change Newton solution acceleration parameter

MDES – Complex mapping design routine

No functions implemented

QDES – Surface speed design routine

No functions implemented

GDES – Geometry design routine

TSET	rr	Set new thickness and camber
EXEC		Set current airfoil ← buffer airfoil. (Command may also be specified by X, rather than EXEC)

GDES CAMB – Modify camber shape directly or via loading

No functions implemented

PPAR – Show/change paneling

N	i	Number of panel nodes
P	r	Panel bunching paramenter
T	r	TE/LE panel density ratio
R	r	Refined area/LE panel density ratio

There are a few simple steps required in order to use XFOIL commands. Once the modified version of XFOIL has been compiled, the executable `xfoil.exe` should be place in the desired working directory. No other files are required for execution. Start the command prompt by either typing `cmd` in the Window's Run dialog, or by selecting the Command Prompt icon, normally located under the Accessories folder of the start menu. Next, navigate to the directory where `xfoil.exe` is located, using the "cd" command. At the command prompt, run the XFOIL program.

The XFOIL program command line consists of the name of the executable, either `xfoil` or `xfoil.exe`, and the command line arguments necessary to execute the desired functions. If located in a submenu, the required menus must be included before the desired XFOIL command. If required for the command, any command arguments should follow the command. The general format is:

```
xfoil [MENU] [COMMAND] [ARGS] [MENU] [COMMAND] [ARGS]...
```

For any of the top level menu commands (`REVE, LOAD, NACA, NORM, PANE`), no menu is required. Arguments will also be omitted if not necessary for the particular command (e.g. `NORM, CPMN`). Although not required, the `QUIT` command may be placed as the last command if desired. If `QUIT` is used, screen output will be generated verifying proper termination of XFOIL. A simple test can be used to verify proper execution of XFOIL by typing "`xfoil QUIT`" at the command prompt, to which the output should be:

```
START of XFOIL
START of Menu Loop. Command is: QUIT
 QUIT. End of XFOIL
```

The typical command structure used in this project is described below. If airfoil geometry was to be input from a data file, NORM was used to ensure that the foil would be normalized upon input to XFOIL. Airfoil geometry was specified using either `NACA` or `LOAD`. If necessary, the geometry was then adjusted using `TSET` to set the desired camber and thickness (Must also include the `X` or `eXec` command to set the current airfoil from the buffer airfoil, otherwise changes will not be processed. See XFOIL 6.9 user manual for more information on buffer and current airfoils.) Finally, the minimum pressure coefficients were calculated in increments of 0.1° for angles of attack from -5° to 6°, and the results were written to `cpminarray.txt`. MATLAB used the following command to achieve the above results:

```
xfoil.exe NORM LOAD foildata PANE GDES TSET 0.06 0.04
GDES X OPER ALFL -5 0.1 6
```

The results are then read in by MATLAB from the `cpminarray.txt` file for processing.

Future modification may be made to the source code to allow for the use of additional commands, or to alter existing commands. A basic description of the code structure will aid those wishing to conduct future modifications. Upon execution, XFOIL reads up to 50 command line arguments into the `CMDARGS` variable, which is a character array. Then, within a loop structure, a single command line argument is read into the `COMAND` variable. The `CMDSTEP` variable, which is a counter, initially set to 1, is incremented each time a new command line argument is read in from `CMDARGS`. The `COMAND` variable is then process and compares `COMAND` to all of the top menu items, both menus and commands. If a menu description such as `OPER` is the first command line argument, then XFOIL will enter the `OPER` menu routine, read the next command line argument into the `COMAND` variable, and increment `CMDSTEP`. If the `COMAND` variable contains an XFOIL command, then the routine for that command will be entered. If necessary for execution, the required number of command line arguments will be read in as command arguments. Again, each time that a command line argument is read, `CMDSTEP` will increment to keep track of which command line argument is being processed.

This process will continue until all command line arguments are processed or `QUIT` is received by XFOIL. XFOIL will search through the current menu, looking for a match to the `COMAND` variable, until the end of the menu is reach, and will return to the top level menu. If no match is found, then XFOIL produces an output saying the command is not recognized. If in the top menu, and no match is found, then XFOIL will read in the next command line argument, and look for a matching command or menu. For example, if the command "`xfoil TEST QUIT`" is entered at the

command prompt (TEST is not a valid XFOIL command), the following output is produced:

```
START of XFOIL
START of Menu Loop. Command is: TEST
 TEST command not recognized.
 END OF LOOP
START of Menu Loop. Command is: QUIT
 QUIT. End of XFOIL
```

Future modifications to the code may be accomplished using the same structure as above. For commands present in the code but not implemented (those not modified to read in arguments as required), modifications to the routine such that command line arguments are read in and assigned to variables as necessary to execute the command may be made. The following is a simple example of how to perform this:

```
C==ORIGINAL CODE================================
      ELSEIF(COMAND.EQ.'NACA') THEN
       CALL NACA(IINPUT(1))
C
C================================================


C==MODIFIED CODE================================
      ELSEIF(COMAND.EQ.'NACA') THEN
       READ(CMDARGS(CMDSTEP), *) CMDNUM3
       CMDSTEP=CMDSTEP+1
      CALL NACA(CMDNUM3)
C
C================================================
```

The original code above calls the NACA function with the argument IINPUT(1), which was input by the user at the menu prompt. The modified code requires that no user input is required, and command line arguments are sequentially read from the command line. As a result, the IINPUT variable is not used in the modified code. For the modified code, after reaching the NACA command (which was the current command being processed in the COMAND variable), the next command line argument is read in from the CMDARGS variable and is assigned to CMDNUM3, which is used as the argument for the call to the NACA function. CMDSTEP is then

incremented. The next command will be read into COMAND upon restarting the next menu loop, if applicable. This structure allows the processing of arguments as necessary for the command being executed. Other existing functions may be included in XFOIL, or new functions added, as desired using a method similar to that described above.

Appendix G: Instruction for compiling modified XFOIL Code

This appendix is intended to provide basic instructions on how to obtain and compile the source code for the modified version of XFOIL used for this project. The original source code for XFOIL as released by Mark Drela can be obtained at http://web.mit.edu/drela/Public/web/xfoil/. Version 6.96 was used for this project. In order to compile the official release version of XFOIL, it is recommended to follow the instructions that are contained in the `README` that is included with the *.tar files containing the source code. Since this project was conducted on a Win32 based PC, the following recommendations are provided based on personal experience while trying to compile the source code in Windows XP

(1) Download and install Cygwin, available at http://www.cygwin.com/. Cygwin is a Linux-like environment for Windows. It consists of two parts:

- A DLL (cygwin1.dll) which acts as a Linux API emulation layer providing substantial Linux API functionality.

- A collection of tools which provide Linux look and feel.

Cygwin allows native Linux applications to be run on Windows machines, if they are rebuilt from their original source code using Cygwin. Specifically, the original source code for XFOIL made use of X11 window tools that are not normally available in Windows (Unix like plotting). The use of Cygwin was a fix to this.

Once running the Cygwin setup program, and under the "Select Packages Screen", complete the following actions prior to clicking the "Next" button:

-Under the "Devel" pull-down menu, select the "gcc-g77: Fortran Package" for installation by click the "Skip" item on the left column. This action selects the current version to include in the installation. This action will also select add-on packages required for installation.

-Under the "Devel" pull-down menu, select the "make: The GNU version of the 'make' utility" for installation by click the "Skip" item on the left column.

-Under the "X11" pull-down menu, select the "xorg-x11-base: Cygwin/X base" package for installation, and the "xorg-x11-devel: Cygwin/X headers and import libraries". Associated add-ons will also be automatically selected.

(2) A basic guide on installation these items can be found at http://www2.warwick.ac.uk/fac/sci/moac/currentstudents/peter_cock/cygwin/. Although not an official reference, the instructions and guidance provided here were found to be useful.

(3) Download and install MinGW. Instructions can be found by following the download link on the http://www.mingw.org/ web page. When running the installation program, ensure that the selection box for 'g77 compiler' is checked under "Select Components to Install:", and continue with the installation process.

(5) Add the following lines to the file "`c:\cygwin\etc\bash.bashrc`"

```
PATH=/cygdrive/c/mingw/bin:$PATH
export PATH
```

These statements place the `/Mingw/bin` ahead of `/Cygwin/bin` in the path statement for the Cygwin environment, ensuring that Mingw executables for `gcc.exe` and `g77.exe` are used, rather than the Cygwin versions. This prevents an error when XFOIL is run outside of the Cygwin environment, and eliminates the error when `cygwin1.dll` is not present.

In order to test the above step, start Cygwin and type `which gcc` at the Cygwin $ prompt. The response should be `/cygdrive/c/mingw/bin/gcc`. If the response is `/usr/bin/gcc`, then the above steps were not completed properly.

(4) Download original source code from:
`http://web.mit.edu/drela/Public/web/xfoil/xfoil6.96.zip`

(5) Unzip `xfoil6.96.zip` to a working directory (i.e. `C:\XFOIL\`).

(6) Replace the following files in the `C:\XFOIL\SRC\` directory with the files modified by the author as part of this project:

blplot.f	polplt.f	xoper.f
dplot.f	xfoil.f	xplots.f
gui.f	XFOIL.INC	xqdes.f
modify.f	xgdes.f	xtcam.f
plutil.f	xgeom.f	
pntops.f	xmdes.f	

(7) Place `Makefile` in the `C:\XFOIL\SRC\` directory. Details of this file are provided below.

Makefile

```
#***********************************************************
# Makefile for XFOIL V6.93 programs
# H.Youngren 4/24/01
# M.Drela
#***********************************************************
# Modified by Chris Peterson to generate modified version
# of XFOIL that removes menus and plots, and executes from
# command prompt.
#***********************************************************


SHELL = sh
#BINDIR = $(HOME)/bin/
BINDIR = .

PROGS = xfoil
# pplot pxplot

SRC = ../src
OSRC = ../osrc

XFOILOBJ = xfoil.o xpanel.o xoper.o xtcam.o xgdes.o xqdes.o xmdes.o \
xsolve.o xbl.o xblsys.o xpol.o xplots.o pntops.o xgeom.o xutils.o modify.o \
blplot.o polplt.o aread.o naca.o spline.o plutil.o iopol.o gui.o sort.o \
dplot.o profil.o

#PPLOTOBJ = pplot.o polplt.o sort.o iopol.o
#PXPLOTOBJ = pxplot.o plutil.o gui.o

XUTILOBJ  = userio.o

FTNLIB =
```

```
##-----------------------------------------------------
OSOBJ = frplot0.o

# Use this for individual TS-wave frequency plotting
# OSOBJ = frplot.o ntcalc.o osmap.o getosfile.o

##-----------------------------------------------------
# PLTOBJ = ../plotlib/libPlt.a

# Use this if you have a copy of the plotlib as a system library
# PLTOBJ = -lPlt

# The extra location arg here is for Linux which places X libs in /usr/X11R6
# PLTLIB = -L/usr/X11R6/lib -lX11

###=====================================================
###  Default compilers and flags
###  FFLOPT used for xsolve.f
FC = g77
FFLAGS  = -O
FFLOPT  = -O
INSTALLCMD = install -s

CC = gcc
CFLAGS = -O -DUNDERSCORE

##-------------------------

# Uncomment flags for desired machine...

##-------------------------
### DEC Alpha with OSF and DEC f77/f90 compiler
#FC = f77
#FFLAGS = -fast -O4 -tune host
#FFLOPT = -fast -O4 -tune host
#FFLOPT = -fast -O5 -tune host -unroll 3
# Debug flags
#FFLAGS = -O0 -g
#FFLOPT = -fast -O4 -tune host
##-------------------------
### SGI setup
#FC = f77
#FFLAGS = -O2 -static
#FFLOPT = -O2 -static
##-------------------------
##  Uncomment for RS/6000
#FFLAGS = -O -qextname
#FFLOPT = -O -qextname
##-------------------------
##  Uncomment for HP-9000
#FFLAGS = -O +ppu
#FFLOPT = -O +ppu
#FTNLIB = -U77
##-------------------------
### Absoft Linux f77
#FC = f77
```

```
#FFLAGS = -O -f -s -W -B108 -N34
#FFLOPT = -O -f -s -W -B108 -N34
##-------------------------
### f2c/gcc compiler driver
#FC = fort77
#FFLAGS = -O2 -fomit-frame-pointer
#FFLOPT = -O2 -fomit-frame-pointer
##-------------------------
### GNU g77
#FC = g77
#FFLAGS = -O3 -fomit-frame-pointer
#FFLOPT = -O3 -fomit-frame-pointer
# Debug flags (symbols, array bounds)
#FC = g77
#FFLAGS = -g -O0 -C
##-------------------------
### Intel Fortran Compiler
#FC = ifort
#FFLAGS = -O
#FFLOPT = -O
#FTNLIB = -Vaxlib /usr/lib/C-ctype.o /usr/lib/C_name.o /usr/lib/ctype-info.o
#FTNLIB = -Vaxlib
#FTNLIB = -i_dynamic

##-------------------------
### Double precision option
#FFLAGS = -O -r8
#FFLOPT = -O -r8
#PLTOBJ = ../plotlib/libPltDP.a


all:   $(PROGS)

install:
      $(INSTALLCMD) $(PROGS) $(BINDIR)

clean:
      -/bin/rm $(PROGS)
      -/bin/rm $(XFOILOBJ) $(XUTILOBJ) $(OSOBJ) $(PPLOTOBJ) $(PXPLOTOBJ)
#     -/bin/rm *.o



xfoil: $(XFOILOBJ) $(XUTILOBJ) $(OSOBJ)
      $(FC) -o xfoil $(XFOILOBJ) $(XUTILOBJ) $(OSOBJ) $(PLTOBJ) $(PLTLIB)
$(FTNLIB)

#pxplot:    $(PXPLOTOBJ) $(XUTILOBJ)
#     $(FC) -o pxplot $(PXPLOTOBJ) $(XUTILOBJ) $(PLTOBJ) $(PLTLIB) $(FTNLIB)

#pplot:     $(PPLOTOBJ) $(XUTILOBJ)
#     $(FC) -o pplot $(PPLOTOBJ) $(XUTILOBJ)  $(PLTOBJ) $(PLTLIB) $(FTNLIB)


xfoil.o: $(SRC)/xfoil.f  $(SRC)/XFOIL.INC
      $(FC) -c $(FFLAGS)  $(SRC)/xfoil.f
xpanel.o: $(SRC)/xpanel.f  $(SRC)/XFOIL.INC
```

```
	$(FC) -c $(FFLOPT)  $(SRC)/xpanel.f
xoper.o: $(SRC)/xoper.f  $(SRC)/XFOIL.INC
	$(FC) -c $(FFLAGS)  $(SRC)/xoper.f
xsolve.o: $(SRC)/xsolve.f  $(SRC)/XFOIL.INC
	$(FC) -c $(FFLOPT)  $(SRC)/xsolve.f
dplot.o: $(SRC)/dplot.f  $(SRC)/XFOIL.INC
	$(FC) -c $(FFLOPT)  $(SRC)/dplot.f
xtcam.o: $(SRC)/xtcam.f  $(SRC)/XFOIL.INC $(SRC)/XDES.INC
	$(FC) -c $(FFLAGS)  $(SRC)/xtcam.f
xgdes.o: $(SRC)/xgdes.f  $(SRC)/XFOIL.INC $(SRC)/XDES.INC
	$(FC) -c $(FFLAGS)  $(SRC)/xgdes.f
xqdes.o: $(SRC)/xqdes.f  $(SRC)/XFOIL.INC $(SRC)/XDES.INC
	$(FC) -c $(FFLAGS)  $(SRC)/xqdes.f
xmdes.o: $(SRC)/xmdes.f  $(SRC)/XFOIL.INC $(SRC)/XDES.INC $(SRC)/CIRCLE.INC
	$(FC) -c $(FFLAGS)  $(SRC)/xmdes.f
xbl.o:   $(SRC)/xbl.f    $(SRC)/XFOIL.INC $(SRC)/XBL.INC
	$(FC) -c $(FFLAGS)  $(SRC)/xbl.f
xblsys.o: $(SRC)/xblsys.f                  $(SRC)/XBL.INC
	$(FC) -c $(FFLAGS)  $(SRC)/xblsys.f
xplots.o: $(SRC)/xplots.f $(SRC)/XFOIL.INC
	$(FC) -c $(FFLAGS)  $(SRC)/xplots.f
pntops.o: $(SRC)/pntops.f $(SRC)/XFOIL.INC $(SRC)/XDES.INC
	$(FC) -c $(FFLAGS)  $(SRC)/pntops.f
blplot.o: $(SRC)/blplot.f $(SRC)/XFOIL.INC
	$(FC) -c $(FFLAGS)  $(SRC)/blplot.f
xpol.o: $(SRC)/xpol.f     $(SRC)/XFOIL.INC
	$(FC) -c $(FFLAGS)  $(SRC)/xpol.f
xgeom.o: $(SRC)/xgeom.f
	$(FC) -c $(FFLAGS)  $(SRC)/xgeom.f
xutils.o: $(SRC)/xutils.f
	$(FC) -c $(FFLAGS)  $(SRC)/xutils.f
modify.o: $(SRC)/modify.f
	$(FC) -c $(FFLAGS)  $(SRC)/modify.f
aread.o: $(SRC)/aread.f
	$(FC) -c $(FFLAGS)  $(SRC)/aread.f
naca.o: $(SRC)/naca.f
	$(FC) -c $(FFLAGS)  $(SRC)/naca.f
plutil.o: $(SRC)/plutil.f
	$(FC) -c $(FFLAGS)  $(SRC)/plutil.f
userio.o: $(SRC)/userio.f
	$(FC) -c $(FFLAGS)  $(SRC)/userio.f
gui.o: $(SRC)/gui.f
	$(FC) -c $(FFLAGS)  $(SRC)/gui.f
spline.o: $(SRC)/spline.f
	$(FC) -c $(FFLAGS)  $(SRC)/spline.f
sort.o: $(SRC)/sort.f
	$(FC) -c $(FFLAGS)  $(SRC)/sort.f
profil.o: $(SRC)/profil.f
	$(FC) -c $(FFLAGS)  $(SRC)/profil.f

polplt.o: $(SRC)/polplt.f $(SRC)/PINDEX.INC
	$(FC) -c $(FFLAGS)  $(SRC)/polplt.f
iopol.o: $(SRC)/iopol.f $(SRC)/PINDEX.INC
	$(FC) -c $(FFLAGS)  $(SRC)/iopol.f

#pplot.o:  $(SRC)/pplot.f  $(SRC)/PPLOT.INC
#	$(FC) -c $(FFLAGS)  $(SRC)/pplot.f
```

```
#pxplot.o: $(SRC)/pxplot.f $(SRC)/PXPLOT.INC
#     $(FC) -c $(FFLAGS)  $(SRC)/pxplot.f


frplot0.o: $(SRC)/frplot0.f
      $(FC) -c $(FFLAGS)  $(SRC)/frplot0.f
frplot.o: $(SRC)/frplot.f
      $(FC) -c $(FFLAGS)  $(SRC)/frplot.f
#ntcalc.o: $(SRC)/ntcalc.f
#     $(FC) -c $(FFLAGS)  $(SRC)/ntcalc.f
#
#osmap.o: $(OSRC)/osmap.f
#     $(FC) -c $(FFLAGS)  $(OSRC)/osmap.f
#
#getosfile.o: $(OSRC)/getosfile.c
#     $(CC) -c $(CFLAGS)  $(OSRC)/getosfile.c
```

(8) Start Cygwin, and navigate to the location of the XFOIL source code, with appropriate files replaced by typing "`cd ../../cygdrive/c/xfoil/src`" or to the directory as appropriate.

(9) At the Cygwin prompt, type "`make`". The modified executable used for the work conducted in this thesis should compile. The `xfoil.exe` executable will be built and located in the same directory above in step (8). `xfoil.exe` may now be relocated as necessary and place in the appropriate directory for MATLAB execution.

(10) Future modifications may be made to the source code files (*.f) in order to alter the program as further desired. If the source code files are altered, repeat steps (8) and (9) to generate a new executable file.

Appendix H: MATLAB Files for Calculation of Minimum Pressure Envelopes

XBucket.m

```
%Code by Chris Peterson.
%Code intended to produce minimum pressure envelopes using XFOIL to
%calculate minimum pressure for foil geometry.  Code can either use NACA 4-
%or 5-digit airfoils built into XFOIL, or may read in properly formatted
%thickness and camber distributions from text files.

clc; clear all; close all;

foil_type = 'LOAD';          %Either 'LOAD', or 'NACA' for 4 or 5 digit

if foil_type == 'LOAD'       %Filenames thickness & camber, and data file
    foil_name   = 'foildata';
    load_mean   = 'Brock08act.txt';
    load_thck   = 'Brock66act.txt';
elseif foil_type == 'NACA'
    foil_name = 'FOUR';      %Or 'FIVE'
    fo_loc = 0.4;            %0.X for 4-digt, or 0.05*k for 5-digit (k=1-5)
end

xdir        = '.\xfoil\';    %Specify XFOIL.EXE executable location
panels      = 175;           %Sets number of panels if resetting in XFOIL
Alpha_lim   = [-5 8];        %Angle of attack range
Alpha_delta = 0.1;           %Angle of attack increment

foc_rng     = [0.00 0.06];   %Camber ratio range (Must be <0.1 for 4-digit
NACA)
foc_step    = 0.01;          %Camber ratio increment

toc_rng     = [0.02 0.2];     %Thicness ratio range
toc_step    = 0.02;          %Thickness ratio increment

visc_tog    = 0;             %1-yes(viscous), 0-no(Inviscid)
iter_lim    = 500;           %XFOIL viscous calc iteration limit
Re_no       = 1e7;           %Reynolds number for visc calcs

if visc_tog == 1             %Add visc functionality to XFOIL command line
    visc_cmd = ['OPER ITER ', num2str(iter_lim),' ',...
        'OPER VISC ', num2str(Re_no), ' '];
elseif visc_tog == 0
    visc_cmd = '';
end

%Determine alphas to calculate
A_rng       = Alpha_lim(1):Alpha_delta:Alpha_lim(2);

%Start of main calculation loops
for fo_c = foc_rng(1):foc_step:foc_rng(2)        %Calculate over range of f/c
    k = int8((fo_c+foc_step)/foc_step);      %k is index for data array below
```

```
    for to_c = toc_rng(1):toc_step:toc_rng(2)    %Calculate over range of t/c
        j = int8((to_c-toc_rng(1)+toc_step)/toc_step);

        %IF below determines if XFOIL database will be used, and creates
        %foil if tabulated data is to be read in
        if strcmp(foil_type, 'NACA')
            if strcmp(foil_name, 'FOUR')
                name = get4_nm(fo_loc, fo_c, to_c);
            elseif strcmp(foil_name, 'FIVE')
                name = get5_nm(fo_loc, to_c);
            end
        elseif strcmp(foil_type, 'LOAD')    %Makes foil if req'd
            makefoil(to_c, fo_c, load_mean, load_thck, foil_name);
            name = foil_name;
        end

        %CMD generates call to run the XFOIL executable to calc CPmin.
        cmd = [xdir, 'xfoil.exe ',...
            'NORM ',...
            foil_type, ' ', name, ' ',...
            'GDES TSET ', num2str(to_c), ' ',num2str(fo_c), ' ',...
            'GDES X ',...
             visc_cmd, ...
            'OPER ALFL ', num2str(Alpha_lim(1)),' ',...
            num2str(Alpha_delta),' ', num2str(Alpha_lim(2)), ' '];

        system(cmd);                          %Calls XFOIL
         %Reads in -Cpmin, and x-location of -Cpmin
        fid = fopen('CPMINARRAY.txt');
        clear datain;
        if visc_tog == 0;
            datain = textscan(fid, '%f64  %f64', 'headerlines', 1);
        elseif visc_tog == 1;
            datain = textscan(fid, '%f64  %f64  %f64  %f64', ...
                'headerlines', 1);
        end
        fclose(fid);
        if visc_tog == 1;
            cpmni(j,:,k)    = datain{1,1};
            xcpmni(j,:,k)   = datain{1,2};
            cpmnv(j,:,k)    = datain{1,3};
            xcpmnv(j,:,k)   = datain{1,4};
        elseif visc_tog == 0
            cpmni(j,:,k)    = datain{1,1};  %Data array for minimum Cp
            xcpmini(j,:,k)  = datain{1,2};  %Data array for location of CPmin
        end
    end

%Generates Bucket diagrams, new plot for each Fo/C
    figure();
    hold on; grid on;
    cmap = colormap(hsv(toc_rng(2)/toc_step+1)); %Generates color distibution
    set(gca,'ColorOrder',cmap);
    plot(-cpmni(:,:,k), A_rng(1:length(cpmni))); %Plots Alpha vs. -Cpmin
    xlim([0 3]);
    if fo_c > 0
```

```
        ylim(Alpha_lim);                          %Set plot X/Y limits
    else
        ylim([0 8]);
    end
    xlabel('-CP_m_i_n'); ylabel('Angle of Attack (\alpha)');
    if foil_type == 'NACA'
        title_name = [foil_type, ' ', foil_name];
    elseif foil_type == 'LOAD'
        title_name = ['Meanline: ', load_mean, '.  Thickness: ', load_thck];
    else
        title_name = 'UNKNOWN TYPE';
    end
    title({['INVISCID Brockett Diagram',10, title_name, 10,...
        ' Fo/c = ', num2str(fo_c), '    ', '\Delta\alpha = ',
num2str(Alpha_delta)]});
    tau = toc_rng(1):toc_step:toc_rng(2);          %Used for legend
    leg_st = cell(1,length(tau));                  %Initializes cells
    for i = 1:length(tau);                         %Set vales to cells
        leg_st(i) = {num2str(tau(i))};
    end
    legend(leg_st, 'Location', 'SouthEast')

    if visc_tog == 1
        figure();
        hold on; grid on;
        cmap = colormap(hsv(toc_rng(2)/toc_step+1));
                    %Generates color distibution
        set(gca,'ColorOrder',cmap);
        plot(-cpmnv(:,:,k), A_rng(1:length(cpmnv)));
                    %Plots Alpha vs. -Cpmin
        xlim([0 3]);
        ylim(Alpha_lim);                %Set plot X/Y limits
        xlabel('-CP_m_i_n'); ylabel('Angle of Attack (\alpha)');
        if foil_type == 'NACA'
            title_name = [foil_type, ' ', foil_name];
        elseif foil_type == 'LOAD'
            title_name = ['Meanline: ', load_mean, '.  Thickness: ',
load_thck];
        else
            title_name = 'UNKNOWN TYPE';
        end
        title({['VISCOUS Brockett Diagram',10, title_name,10,...
            ' Fo/c = ', num2str(fo_c), '    \Delta\alpha = ',
num2str(Alpha_delta)]});
        tau = toc_rng(1):toc_step:toc_rng(2);          %Used for legend
        leg_st = cell(1,length(tau));                  %Initializes cells
        for i = 1:length(tau);                         %Set vales to cells
            leg_st(i) = {num2str(tau(i))};
        end
        legend(leg_st, 'Location', 'SouthEast')
    end
end
```

makefoil.m

```
%Code by Chris Peterson.  Code will read in specified camber and thickness
%   distributions and generate foil geometry file for XFOIL.  Thickness and
%   camber are scaled to t_set and f_set.
%   Coordinates start at TE, go forward CCW along upper surfact to LE,
%   and back to TE along lower surface.

function [] = makefoil(t_set, f_set, mean_type, thick_type, save_as)

%     clc; clear all; close all;
%     t_set      = 0.1;
%     f_set          = 0.08;
%     mean_type    = 'NACAa=08(Brockett).txt';
%     thick_type   = 'NACA66(Brockett).txt';
%       save_as      = 'brockett';

make_plot       = 'no'; %Generate plot toggle ('yes' or 'no')
N_parab_def     = 35;   %Number of points to make nose parabola. Fails at
numbers < ~20
N_parab_eval    = 11;   %Number of points to include at the nose in data
export;
N_surf_pts      = 80;   %Number of points along body to TE (not including LE)
                        %N_parab_pts + N_surf_pts must be < 150
fract           = 1-2/N_parab_eval;  %Fraction of parabola to use from LE to
0.005.
                        %Max parabola point must be less than 0.005
                        %to prevent sharp cornder at 0.005.
conc_fact       = 2;    %Power for exponential disribution at LE.  This
                        %concentrates point near tip.

%Get meanline and dy/dx distributions from mean line data base
[x_f fc_o dydx_o]   = getmeanline(mean_type);
[x_t tc_o RLE_o]    = getthickdist(thick_type);

%Scale appropriately
t_set   = t_set/2;              %uses 1/2 thickness
if max(fc_o) ~= 0
    f_scale = f_set/max(fc_o);
elseif max(fc_o) == 0
    f_scale = 0;
end
f_c     = fc_o * f_scale;
dydx    = dydx_o * f_scale;
t_scale = t_set/max(tc_o);
t_c     = tc_o * t_scale;
RLE     = RLE_o * (t_scale)^2;

%Find points along RLE nose parabola
x_RLE = fract*0.005*(0:1/(N_parab_def-1):1).^conc_fact;
t_RLE = sqrt(2*RLE*(x_RLE));

%Spline parabola and tabulated data for thickness function
x_locs  = [x_RLE x_t(2:end)];                   %New combined x/c values
```

```
t_fnct  = csape(x_locs, [1e10 t_RLE t_c(2:end) 1],[1 0]); %1e8 sets init
slope = -inf
%Make x locations for generating data file
    %Cosine spacing from 0.005 to TE
x_cos_sp= 0.005 + 0.5*0.995*(1-cos(0:pi/(N_surf_pts-1):pi));
    %Exponential spacing for nose
x_eval_LE = fract*0.005*(0:1/(N_parab_eval-1):1).^conc_fact;
t_eval_LE = sqrt(2*RLE*(x_eval_LE));
x_eval_mb = [x_cos_sp];  %Establishes eval points
t_eval_mb = fnval(t_fnct, x_eval_mb); %Evaluates spline at eval points
x_eval = [x_eval_LE x_eval_mb];
t_eval = [t_eval_LE t_eval_mb];

%Spline tabulated data for camber at same x/c locations as thickness
f_fnct  = csape(x_f, f_c);
f_eval  = fnval(f_fnct, x_eval);
dydx_eval = fnval(fnder(f_fnct), x_eval);

%Plotting for unrotated parameters
if strcmp(make_plot,'yes')
    figure();
    hold on;
    axis equal;             %Set X:Y to unity
    title('Camber, Thickness, and LE Graphical Display')
    xlabel('X/C');
    xlim([-0.01 0.25]);     %Set Initial Zoom
        %Plot thickness
        fnplt(t_fnct, 'y'); fnplt(f_fnct, 'g')
        plot(x_t, t_c, 'co'); plot(x_f, f_c, 'ro')
        plot(x_RLE, t_RLE, 'k.');
        %Plot RLE Circle and parabola for viewing on plot
        plot(RLE - RLE*cos(0:pi/100:pi), RLE*(sin(0:pi/100:pi)), 'b:');
        plot((0:1/10000:0.2), sqrt(2*RLE*(0:1/10000:0.2)), 'r:');
        %Plot camber


    legend('Splined Thickness', 'Splined Camber',...
        'Tabulated Thickness (Scaled)', 'Tabulated Camber (Scaled)',...
        'Calcuated Parabola', 'Leading Edge Radius', 'LE Parabola',...
        'Location', 'southeast')
end

%Calculate upper and lower surface ordinates
x_u = x_eval - t_eval.*sin(atan(dydx_eval));
y_u = f_eval + t_eval.*cos(atan(dydx_eval));
x_l = x_eval + t_eval.*sin(atan(dydx_eval));
y_l = f_eval - t_eval.*cos(atan(dydx_eval));

%Solve for most forward point on foil
[x_fwd, min_i] = min(x_u);
y_fwd = y_u(min_i);

%New plot for actual upper and lower surfaces
if strcmp(make_plot,'yes')
    figure();
```

```
    hold on;
    axis equal;              %Set X:Y to unity
    xlim([0 1]);     %Set Initial Zoom
    plot(x_u, y_u, 'b-', x_u, y_u, 'r.')
    plot(x_l, y_l, 'b-', x_l, y_l, 'r.');
    plot(x_eval, f_eval, 'g-', x_eval, f_eval, 'r.')
    plot(x_fwd,y_fwd, 'kp')
end

%Combine coordinates into a single array of points from TE along upper
%surface around LE back to TE along lower surface
x_comb = [fliplr(x_u) x_l];
y_comb = [fliplr(y_u) y_l];

%Rotate and scale such that max forward point is at 0,0, and TE is at 0,1.
%Assumes TE is already at 0,0 (Uses method in Brockett Report)
shift_ang = atan(y_fwd/(1-x_fwd));
%Scaled chord length back to 1 (accounts for portion forward of 0)
x_scaled = (x_comb-x_fwd)./(1-x_fwd);
y_scaled = (y_comb-y_fwd)./(1-x_fwd);
%Rotate so that most forward point is at 0,0
x_rot = (x_scaled.*cos(shift_ang) - y_scaled.*sin(shift_ang))/...
          sqrt(1+(y_fwd/(1-x_fwd))^2);
y_rot = (y_scaled.*cos(shift_ang) + x_scaled.*sin(shift_ang))/...
          sqrt(1+(y_fwd/(1-x_fwd))^2);

%New plot for final upper and lower surfaces
if strcmp(make_plot,'yes')
    figure();
    hold on;
    title('Final Points exported to Data File.');
    axis equal;              %Set X:Y to unity
    xlim([0 1]);     %Set Initial Zoom
    plot(x_rot, y_rot, x_rot, y_rot, 'r.');
    legend('Connect the dots', 'Actual data points');
end

%Write to text file for use in XFOIL.
cmd = ['del ', save_as];        %save_as is file name to be written to
system(cmd);                    %Delets previous file
fid = fopen(save_as, 'w');
for i = 1:length(x_rot)
    fprintf(fid, '%10.8f %10.8f\n', x_rot(i), y_rot(i));
end
fclose(fid);
```

getmeanline.m

```
%   Code by Chris Peterson
%   Code developed to read meanline information from data file 'filename'.
%   Data will be read in from file, and returned to function call.  Data
%   return is vectors containing x-locations, camber distribution, and
%   camber line slope values.  Function checks for 999 value specifying
%   less data points than standard input format.

function [x_loc f_c dy_dx] = getmeanline(filename)

cd('./Meanline');

input   = dlmread(filename, '\t', 4, 0);
M       = input';
x_loc_in= M(1,:)/100;
f_c_in  = M(2,:)/100;
dy_dx_in= M(3,:);

for i=1:length(x_loc_in)
    if x_loc_in(i) == 9.99 %Checks to see if formatted with less points
        x_loc   = x_loc_in(1:i-1);
        f_c     = f_c_in(1:i-1);
        dy_dx   = dy_dx_in(1:i-1);
        cd ..;
        return
    else
        x_loc   = x_loc_in;
        f_c     = f_c_in;
        dy_dx   = dy_dx_in;
    end
end

cd ..
```

getthickdist.m

```
%   Code by Chris Peterson
%   Code developed to read thickness information from data file 'filename'.
%   Data will be read in from file, and returned to function call.  Data
%   return is vectors containing x-locations, thickness distribution, and
%   value of leading edge radius.  Function checks for 999 value specifying
%   less data points than standard input format.

function [x_loc t_c RLE] = getthickdist(filename)

cd('./Thickness');

input   = dlmread(filename, '\t', [4 0 29 2]);
M       = input';
x_loc_in= M(1,:)/100;
t_c_in  = M(2,:)/100;
fid     = fopen(filename);
RLE     = textscan(fid, '%s', 'headerlines', 29);
fclose all;
RLE     = str2num(RLE{1}{7})/100;


for i=1:length(x_loc_in)
    if x_loc_in(i) == 9.99 %Checks to see if formatted with less points
        x_loc   = x_loc_in(1:i-1);
        t_c     = t_c_in(1:i-1);
        cd ..;
        return
    else
        x_loc   = x_loc_in;
        t_c     = t_c_in;
    end
end

cd ..;
```

get4_nm

```
%Code by Chris Peterson.  Code generates a 4 digit string based on location
%of max camber, camber and thickness to generate NACA 4-digit designation.

function [name] = get4_nm(loc, fo_c, to_c)

if fo_c == 0
    no1 = '0';
    no2 = '0';
else
    no1     = num2str(int8(100*fo_c));
    no2     = num2str(int8(10*loc));
end

if to_c < 0.1
    no34    = strcat('0', num2str(int8(100*to_c)));
else
    no34 = num2str(int8(100*to_c));
end

name = strcat(no1, no2, no34);
```

get5_nm.m

```
%Code by Chris Peterson.  Code generates a 5 digit string based on location
%of max camber, camber and thickness to generate NACA 4-digit designation.

function [name] = get5_nm(loc, to_c)

no1 = '2';      %Only designs implemented in XFOIL are 210, 220,..., 250

if loc > 0.25 | loc < 0.025        %Will round to nearest 10%
    error('Improper location for Max Camber.')
else
    no23 = num2str(10*int8(2*10*loc));
end

if to_c < 0.1
    no45    = strcat('0', num2str(int8(100*to_c)));
else
    no45 = num2str(int8(100*to_c));
end

name = strcat(no1, no23, no45);
```

Appendix I: Meanline and Camber Data File Format

Meanline and thickness distributions for various NACA foils are available at the Public Domain Aeronautical Software website (http://www.pdas.com/avd.htm). The formats of these files were used as input for the meanline and thickness distributions, and the MATLAB code assumes similar formatting for other tabulated offsets. Examples of meanline and thickness data files are shown below for the NACA a=0.3 Meanline, and the NACA 66-008 thickness distributions:

NACAa=0.8.txt

NACA Mean Line a=0.3
(Stations and ordinates given
in per cent of airfoil chord)

x	y	dy/dx
0	0	0
0.5	0.3892	0.6554
0.75	0.5463	0.6052
1.25	0.8317	0.5416
2.5	1.4478	0.454
5	2.4575	0.3634
7.5	3.2925	0.3078
10	4.008	0.2662
15	5.1721	0.2025
20	6.052	0.1507
25	6.6853	0.1028
30	7.0721	0.0483
35	7.1754	-0.002
40	7.0738	-0.0371
45	6.8162	-0.0649
50	6.4333	-0.0875
55	5.9488	-0.1057
60	5.3828	-0.1201
65	4.7531	-0.1312
70	4.0763	-0.139
75	3.3683	-0.1436
80	2.6453	-0.145
85	1.9243	-0.1428
90	1.2244	-0.1364
95	0.5698	-0.1243
100	0	0

NACA66-008.txt

NACA 66-008
(Stations and ordinates given
in per cent of airfoil chord)

x	y	dy/dx
0	0	0
0.5	0.6111	0.5674
0.75	0.7341	0.4353
1.25	0.9151	0.306
2.5	1.2183	0.2079
5	1.6716	0.1605
7.5	2.0321	0.1303
10	2.3336	0.1123
15	2.8245	0.0857
20	3.2003	0.0658
25	3.4904	0.0505
30	3.7091	0.0372
35	3.8642	0.0253
40	3.9603	0.0131
45	3.9984	0.0016
50	3.9777	-0.0102
55	3.8945	-0.0236
60	3.7378	-0.0408
65	3.4659	-0.0693
70	3.0593	-0.0915
75	2.5713	-0.1039
80	2.0256	-0.1137
85	1.445	-0.1169
90	0.8674	-0.1131
95	0.3378	-0.0952
100	0	-0.0038

L.E. radius = 0.389 percent chord

All values are specified as a percentage of chord length. Note that for meanline data, 'y' values represent camber offsets, and dy/dx is camberline slope. For thickness distribution, 'y' represents thickness values perpendicular to the meanline, and dy/dx is thickness slope. In addition, leading edge radius must be specified.

If desired, meanline and thickness for arbitrary foil shapes may be specified using the above format, or alternatively, if offsets are available, but not at the locations specified above, the following formats may also be used. These meanline and thickness offsets were taken from reference [4]

Brock08act.txt

NACA Mean Line a=0.8(modified)
(Stations and ordinates given
in per cent of airfoil chord)

x	y	dy/dx
0	0	0.71485
0.7596	0.6006	0.66001
3.0154	1.8381	0.47712
6.6987	3.3684	0.36751
11.6978	4.9874	0.28681
17.8606	6.5407	0.22096
25	7.9051	0.1635
32.899	8.9831	0.11071
41.3176	9.6994	0.06001
50	10	0.00914
58.6824	9.8503	-0.04448
67.101	9.2306	-0.10483
75	8.1212	-0.18132
82.1394	6.3884	-0.31892
88.3022	4.2227	-0.37243
93.3013	2.3423	-0.37425
96.9846	0.9982	-0.35148
99.2404	0.2365	-0.32028
100	0	-0.30025
999	0	0
0	0	0
0	0	0
0	0	0
0	0	0
0	0	0
0	0	0

Brock66act.txt

NACA 66 (Mod)-From Brockett
(Stations and ordinates given
in per cent of airfoil chord)

x	y	dydx
0	0	0
0.7596	0.817	0
3.0154	1.608	0
6.6987	2.388	0
11.6978	3.135	0
17.8606	3.807	0
25	4.363	0
32.899	4.76	0
41.3176	4.972	0
50	4.962	0
58.6824	4.712	0
67.101	4.247	0
75	3.612	0
82.1394	2.872	0
88.3022	2.108	0
93.3013	1.402	0
96.9846	0.83	0
99.2404	0.462	0
100	0.333	0
999	0	0
0	0	0
0	0	0
0	0	0
0	0	0
0	0	0
0	0	0

L.E. radius = .448 percent chord

The file formats for Brock08act.txt and Brock66act.txt utilize the same number of rows and columns as the previous formats (NACAa=0.8.txt and NACA66-008.txt). The only difference is that when there are less than 26 offset locations, the number 999 must be put after the last data point. This instructs the code to stop reading in data points. All other values after the last offset location must be filled in with zeros to maintain proper file format and size. Also, dy/dx values for thickness distributions are not required, and may be filled in with zeros if unknown.