An Enhanced Propeller Design Program Based on

Propeller Vortex Lattice Lifting Line Theory

by

Hsin-Lung Chung

B.S. Computer Science
The Citadel (2004)

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 2007

Signature of Author _____
Department of Mechanical Engineering
May 11, 2007

Certified by _____ MAY 3 2007
Richard W. Kimball
Assistant Professor of Engineering
Maine Maritime Academy
Thesis Supervisor

Certified by _____
Patrick J. Keenan
Professor of Naval Architecture
Thesis Reader

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

An Enhanced Propeller Design Program Based on Propeller Vortex Lattice Lifting Line Theory

by

Hsin-Lung Chung

Submitted to the Department of Mechanical Engineering on May 11, 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

A suite of propeller numerical design tools was developed in MATLAB®, a high-level technical computing language. The tools were based on the FORTRAN programs developed by Professor Justin Kerwin at MIT in 2001 and include enhanced parametric design capability, Graphical User Interfaces (GUIs) and enhanced graphics capability. The MIT Propeller Vortex Lattice Lifting Line Program (PVL) is an executable file written in FORTRAN language and serves as a preliminary propeller design tool. In this thesis, PVL was rewritten in MATLAB® and presented with the GUIs. The enhancements were incorporated in a revised propeller design program designated MPVL to distinguish it from PVL. Several new features such as the user-friendly GUIs and colorful graphs were included in MPVL in addition to the full functions of PVL. Moreover, MPVL was validated by comparing its outputs with PVL and MIT Propeller Lifting Line Program (PLL) and proved to be consistent with PVL. The advantages of MPVL were revealed in this thesis and enabled MPVL to replace PVL. MPVL was developed to serve as an open source code for propeller design. It is also a base program which can be extended to perform more sophisticated propeller design applications such as ducted propellers and contra-rotating propellers. A description of the development efforts for this revised propeller design program forms the basis of this thesis.

Thesis Supervisor: Richard W. Kimball

Title: Assistant Professor of Engineering, Maine Maritime Academy

Acknowledgements

The author wishes to express his deepest gratitude to Professor Richard W. Kimball for providing the opportunity for involvement in this research as well for his guidance during its conduct.

The author would like to thank the Taiwanese Navy for giving him the opportunity to pursue higher education and for supporting him financially.

Also, the author would like to thank the faculty of the 2N program at MIT as well as its fellow students for their support. It has been a great experience working and studying with them.

Finally, the greatest credit is deserved by the author's family for their encouragement and understanding during difficult times.

Table of Contents

List of Figures

List of Tables

1 Introduction

The Propeller Vortex Lattice Lifting Line Program (PVL)[1] was rewritten in MATLAB® and presented with Graphical User Interfaces (GUIs) to create a user-friendly platform for propeller design and also to develop a base program that could be extended to perform more sophisticated propeller designs such as ducted propellers and contra-rotating propellers. MATLAB® was the prospective platform for developing the propeller design programs for the following reasons. First, MATLAB® is a high-performance language for technical computing. It integrates computation, visualization and programming in a user-friendly environment. Second, MATLAB® has a vast collection of computational algorithms and solves computing problems, especially those with matrix formulations, in a fraction of the time it would take to write a program in other languages such as C or FORTRAN. Finally, MATLAB® has evolved over years and is extensively used in both the academic environment and industry.

MATLAB® eventually stood out as the best platform for developing the propeller design programs for its capability to create and program GUIs. GUI refers to the idea of icons, buttons, etc., that are visually presented to a user as a "front-end" of a software application[2]. Nowadays computer users prefer to point their mouse pointer to a graphical representation of the application, click on it, and work with the application through interactive cues. A software application that only accepts keyboard-entered commands is now considered primitive. In addition, automation of functions is achieved, and minimal effort is required from the users by adopting GUIs in a software application. The prime motive was to present the propeller design programs with GUIs, and MATLAB® enabled this objective to be accomplished.

The MATLAB® GUI applications based upon the FORTRAN codes from the MIT Hydrofoils and Propellers course were implemented in this thesis. The Hydrofoil Vortex Lifting Line Program (VLL)[3], the Two-Dimensional Vortex Lattice Method Program (VLM)[4] and PVL are hydrofoil and propeller design tools written in FORTRAN language by Professor Justin E. Kerwin at MIT in 2001. VLL and VLM were rewritten in MATLAB® and presented with GUIs in chapter three. The purpose of implementing the two programs was to demonstrate the feasibility and simplicity of creating and programming GUIs in MATLAB®. The high-level GUI developing technique was applied to create two simple GUIs for the two-dimensional hydrofoil design applications. Figure 1.1 and Figure 1.2 show the GUIs created for VLL and VLM respectively. It was shown that engineering computations can be implemented and presented in a simple and elegant way by adopting GUIs.

[1] (Kerwin 2001, 210)
[2] (Marchand and Holland 2002, 385)
[3] (Kerwin 2001, 207)
[4] (Kerwin 2001, 199)

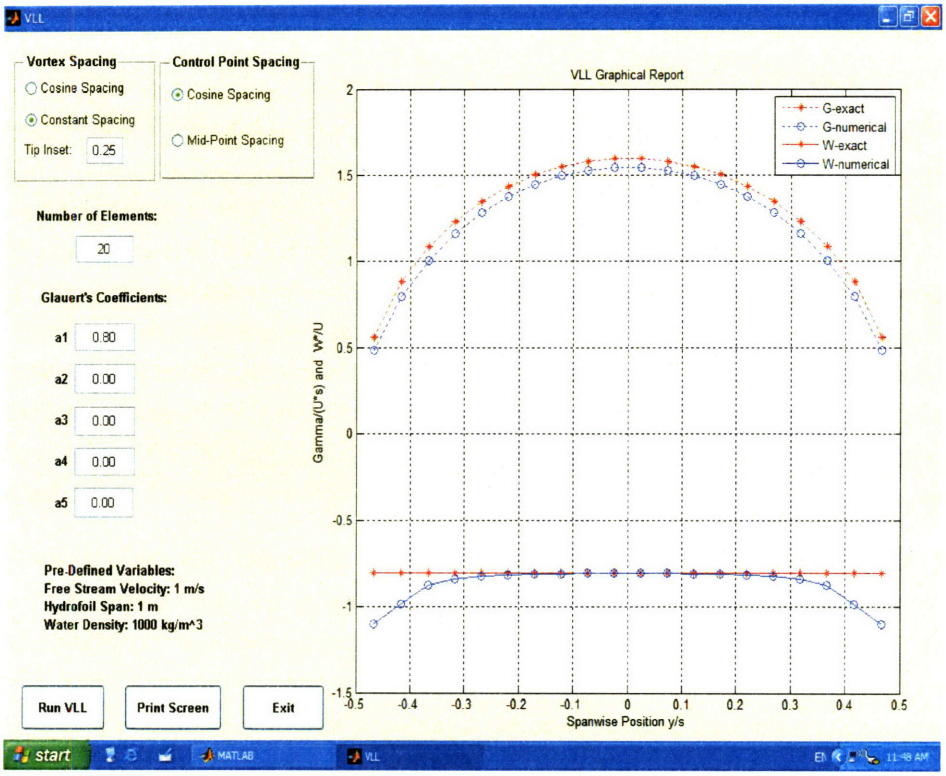


Figure 1. 1 The VLL Graphical User Interface (GUI)

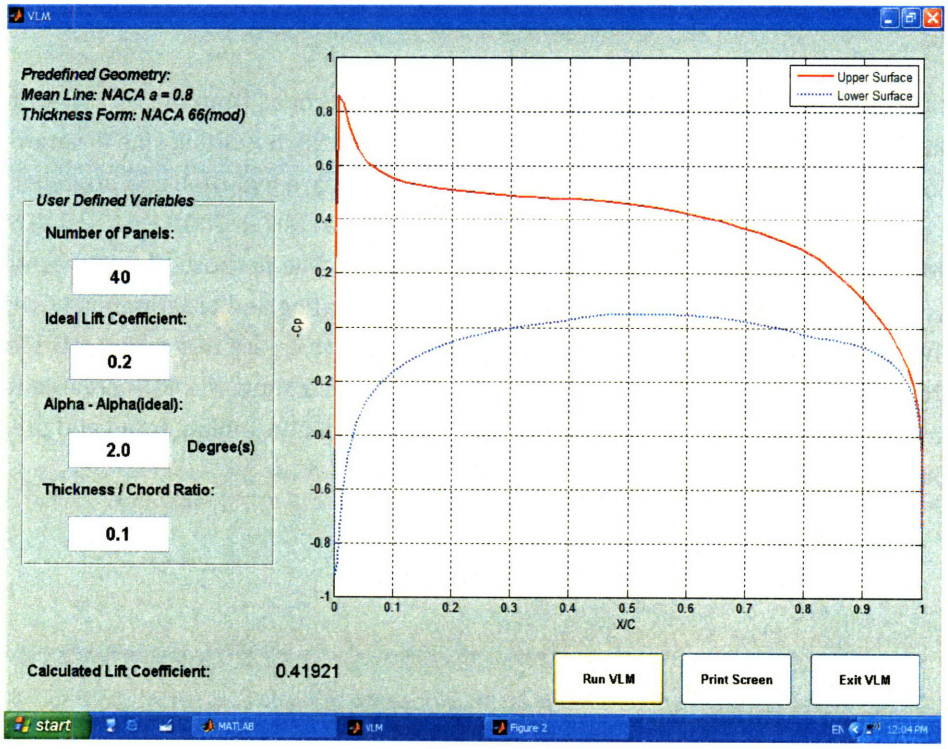


Figure 1. 2 The VLM Graphical User Interface (GUI)

PVL is a simplified version of MIT Propeller Lifting Line Program (PLL)[5], which was developed in the 1980s to handle a variety of multi component propulsors. It is capable of handling single open propeller design. PVL represents the core of this thesis and was rewritten in MATLAB® and presented with more sophisticated GUIs. The implemented GUI program was designated as MPVL, which stands for the MATLAB® version of PVL. The low-level GUI developing technique was applied to create the GUIs for MPVL. MPVL serves as a tool for the parametric analysis which helps optimize the propeller parameters and for the single propeller design. New features such as automatically generated colorful graphs and text files were included in MPVL. Figure 1.3 shows the parametric analysis GUI of MPVL, and Figure 1.4 shows the efficiency curves generated by the GUI. Figure 1.5 shows the single propeller design GUI of MPVL. Figure 1.6 shows the graphical report generated by the single propeller design GUI. Figure 1.7 shows an example of the two-dimensional blade profile. Figure 1.8 shows an example of the three-dimensional propeller image generated by the single propeller design GUI. With the enhanced features, MPVL serves as an excellent open source code for propeller design.

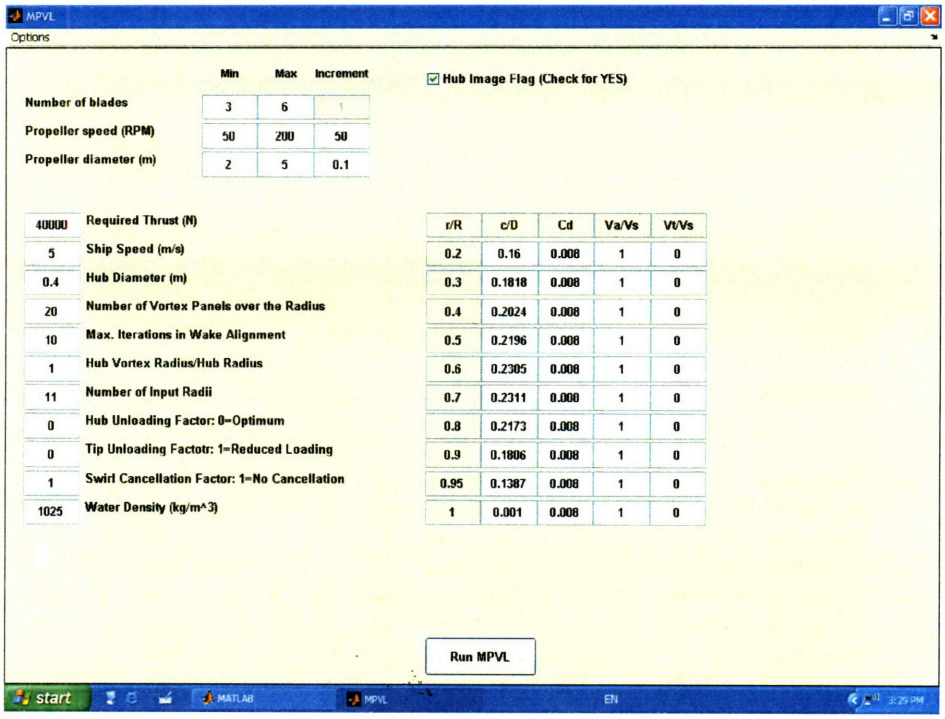


Figure 1. 3 The MPVL Parametric Analysis GUI

[5] (Coney, Hsin and Kerwin 1986)

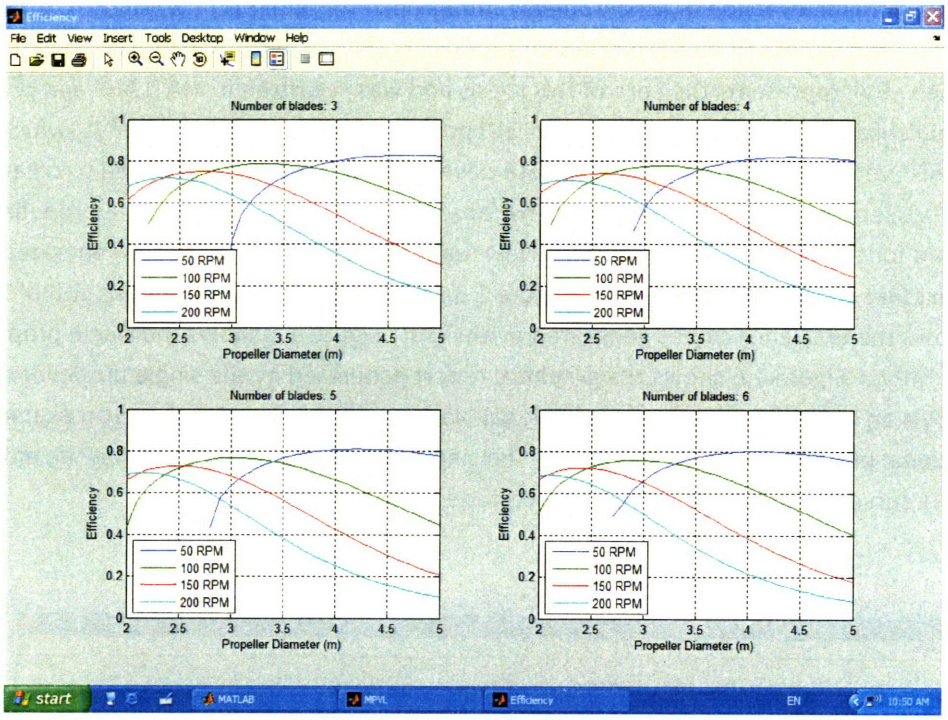


Figure 1. 4 The Efficiency Curves of the MPVL Parametric Analysis GUI

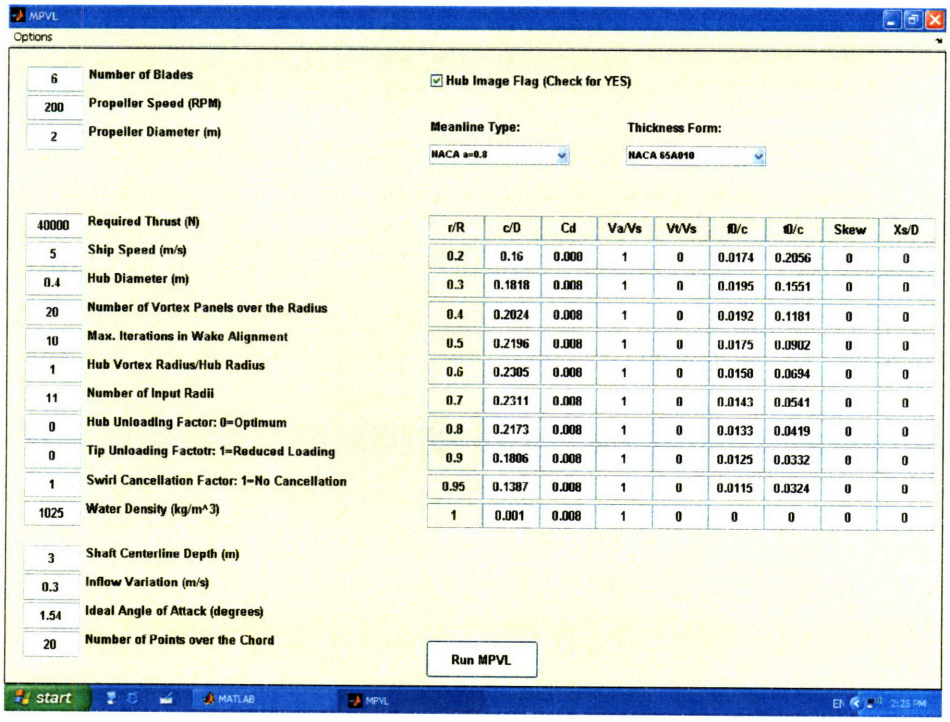


Figure 1. 5 The MPVL Single Propeller Design GUI

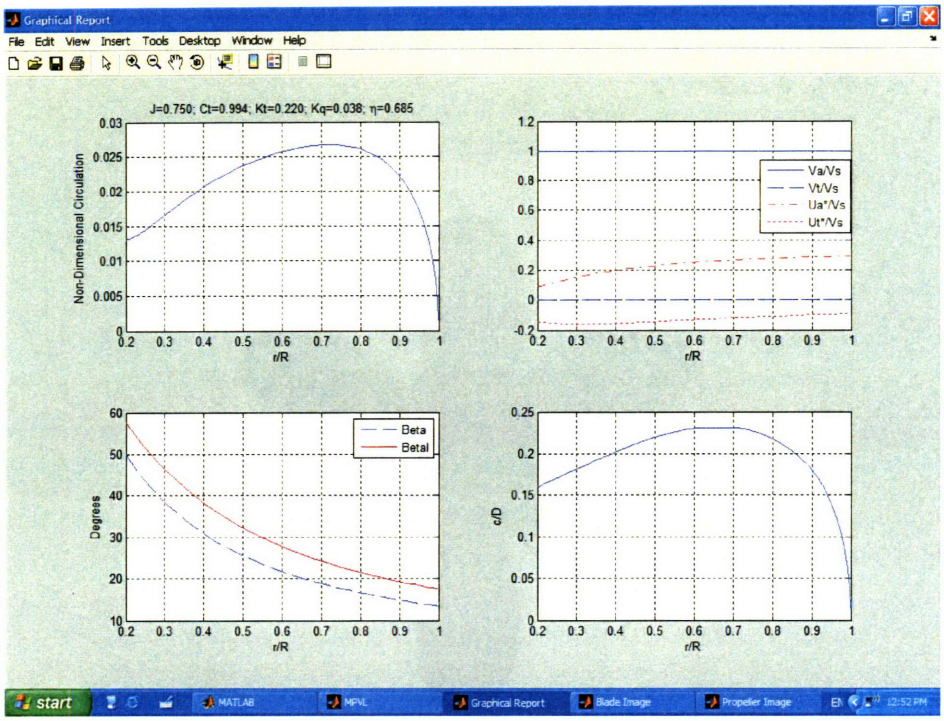


Figure 1. 6 The Graphical Report of the MPVL Single Propeller Design GUI

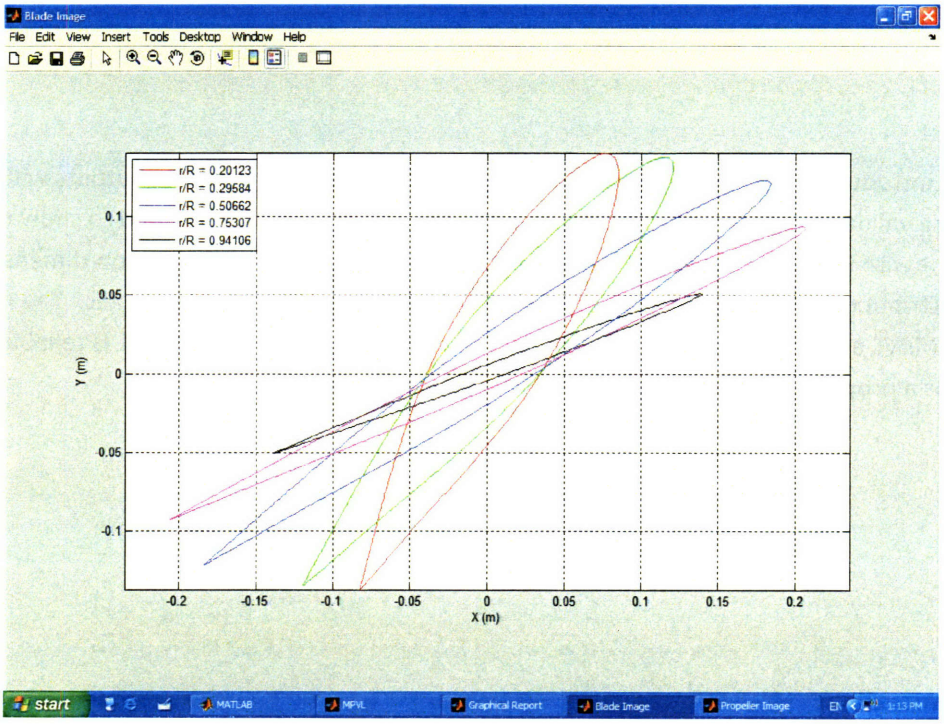


Figure 1. 7 The 2D Propeller Blade Profile of the MPVL Single Propeller Design GUI

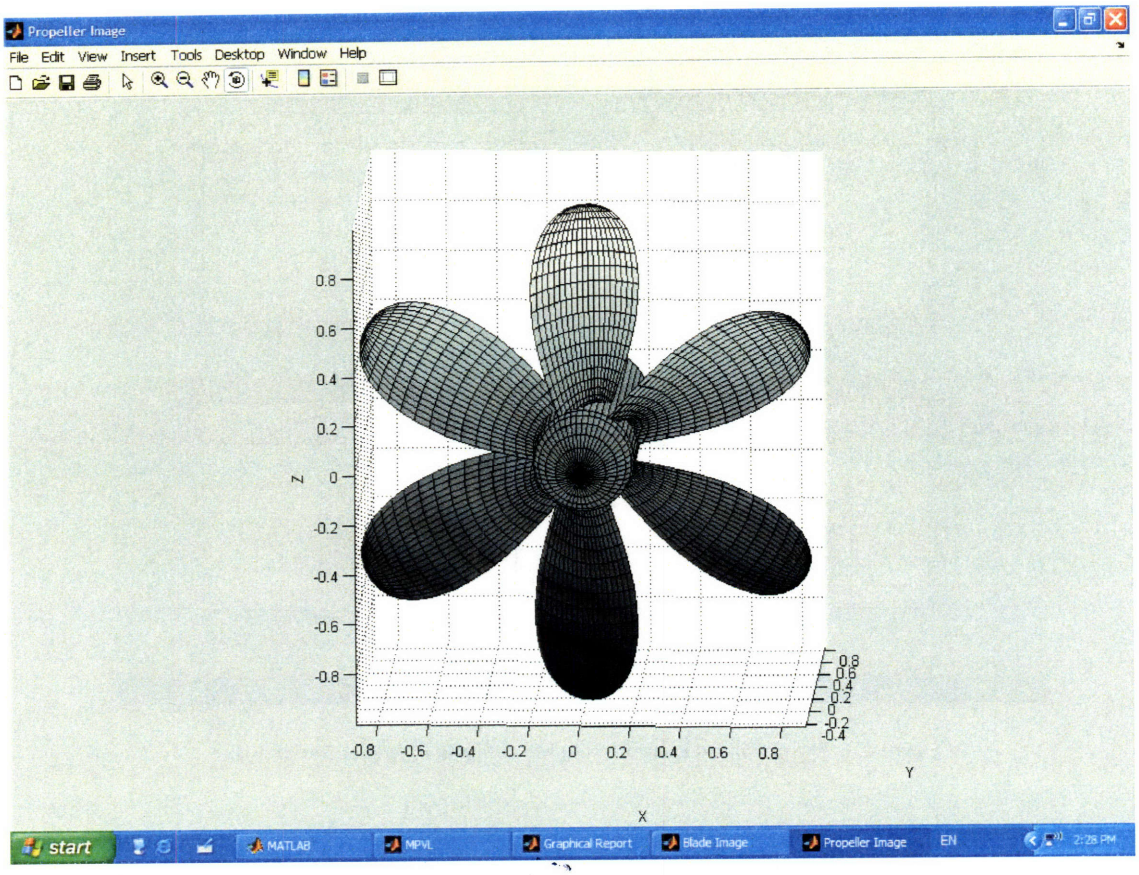


Figure 1. 8 The 3D Propeller Image of the MPVL Single Propeller Design GUI

After introducing its main features, MPVL was validated by comparing its outputs with those from PVL. Five validation cases were defined in chapter five, studied and presented to show that MPVL was consistent with PVL under different loading conditions. Figure 1.9 shows the non-dimensional circulation distribution for a moderately loaded propeller solved by MPVL, PVL and PLL. The results showed that MPVL and PVL matched perfectly. Thus it can be concluded that MPVL is reliable enough to replace PVL as a propeller design tool.

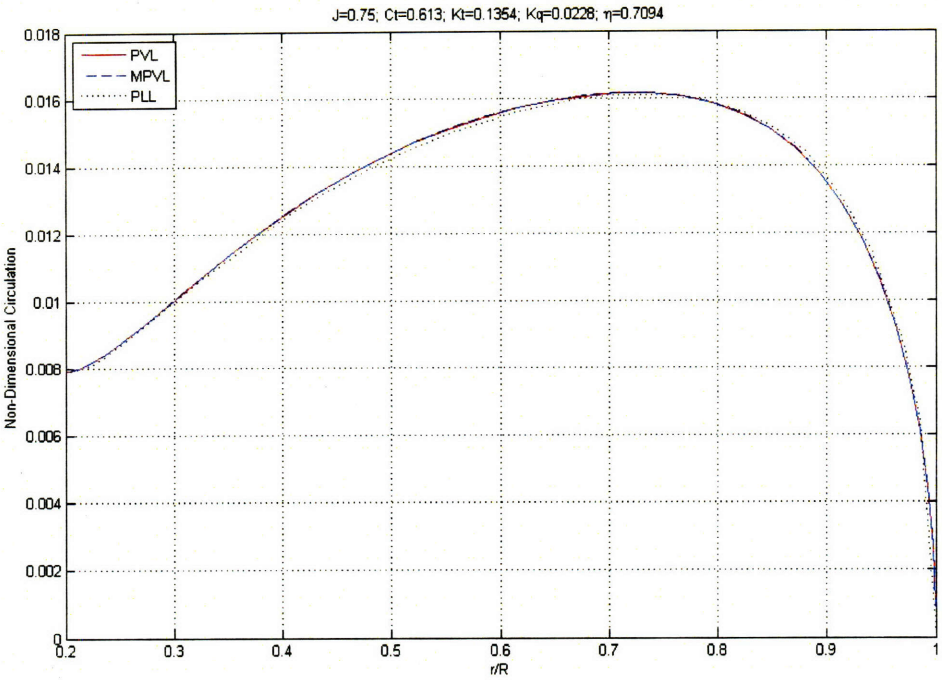


Figure 1. 9 Circulation Distribution of a Moderately Loaded Propeller

A design example and the applications of MPVL were demonstrated in chapter six. Chapter six also serves as a compact user's manual and illustrates the features of MPVL. The input fields in MVPL GUIs were introduced in detail. The parametric analysis GUI (See Figure 1.3.) was first used to determine the optimum propeller parameters for the design example. The efficiency curves (See Figure 1.4.) were generated in only one to several minutes depending on the range and increment of the parameters. The users can then determine the optimum propeller speed, propeller diameter and number of blades based on the efficiency curves. More input fields such as the blade geometry data were then required in the single propeller design GUI (See Figure 1.5) to produce a complete propeller design. It only took less than ten seconds for the single propeller design GUI to finish the calculation process and to produce the graphs and text files. Through this design example, it was shown that laborious computations can be done in fairly short time with MPVL, and automation was achieved by adopting GUIs to demand the minimal efforts from the users.

2 Code Methodology

2.1 Hydrofoil and Propeller Design Applications in FORTRAN

The Hydrofoil Vortex Lifting Line Program (VLL), the Two-Dimensional Vortex Lattice Method Program (VLM) and the Propeller Vortex Lattice Lifting Line Program (PVL) are open source codes written by Professor Justin Kerwin at MIT for the Hydrofoils and Propellers course. The codes are listed in the published course notes[6]. The three programs are written in FORTRAN language in the format of executable files which run in MS-DOS environment. Upon execution of the programs, an MS-DOS window pops on the screen as the operation platform. Users then type certain commands or input fields corresponding to the instructions on the screen. The greatest advantage of this platform is that the executable files are usually compact and can be executed on most personal computers without installing any special software or compiler. However, the first drawback is that entering commands or input fields sometimes can be very tricky and not intuitive. Inexperienced users may be easily frustrated without any instruction or assistance. The second drawback is that the MS-DOS platform has limited capability in data visualization. Graphical representation of data is desired because it is more intuitive and more effective than texts. The third drawback is that it requires an experienced programmer and a special compiler to modify the codes. Thus the programs' potential in extended applications is compromised. Those drawbacks were the prime motives for implementing PVL in MATLAB® and presenting it with GUIs. Figure 2.1 shows an example of the output of MIT Propeller Lifting Line Program (PLL), which is also a FORTRAN executable program for propeller design.

[6] (Kerwin 2001)

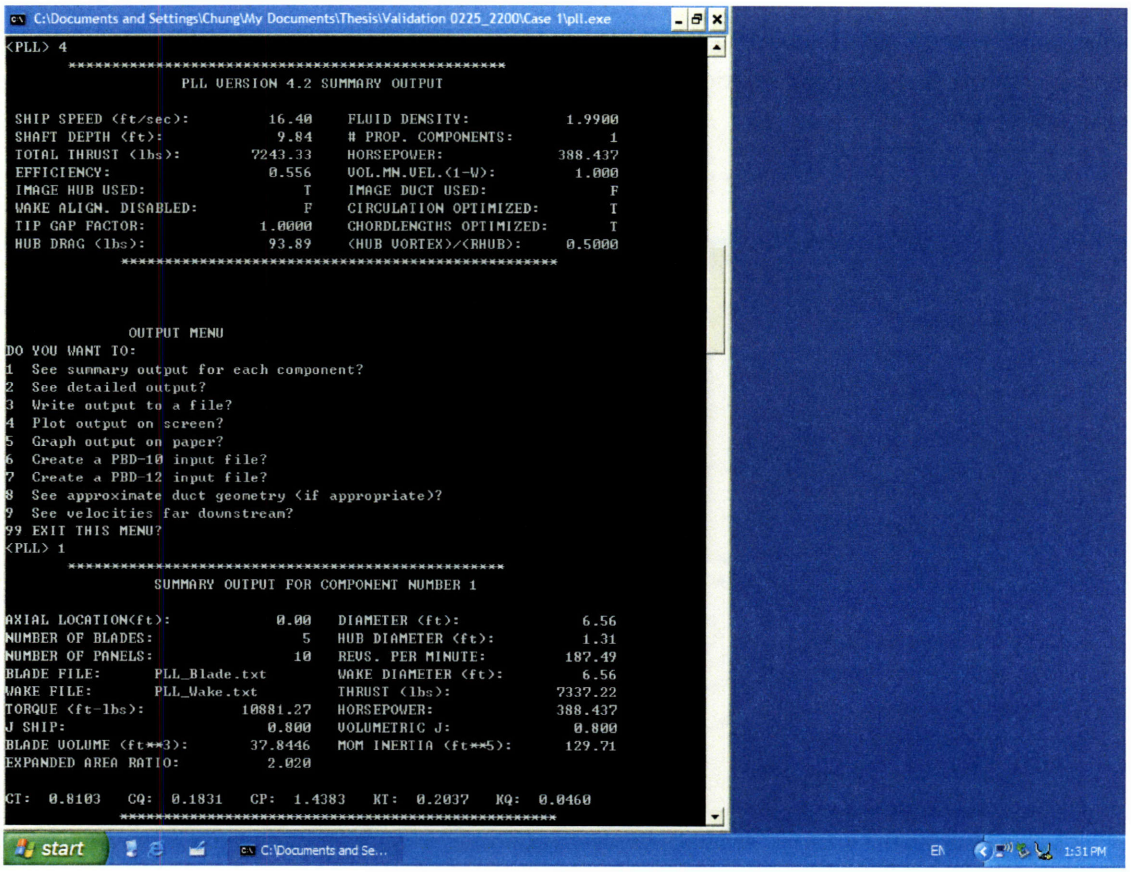


Figure 2. 1 Example of Output of Propeller Lifting Line Program (PLL)

2.2 Graphical User Interface (GUI) in MATLAB®

Graphical User Interfaces (GUIs) have been adopted by most mainstream computer software such as Microsoft Windows® operating system. GUIs provide an interactive, intuitive and user-friendly interface and have become a universal standard for computer interfaces. The greatest advantage of GUIs is that GUIs provide a simple graphical representation of the software application and relieve the users from typing strict programming commands. Events are automatically triggered by clicking on the buttons, menus, or other graphical objects in the GUIs. In short, GUIs provide an environment which demands the minimal knowledge and efforts from the users.

MATLAB® is a high-level technical computing language and interactive environment created by MathWorks. It has been extensively used in a wide range of engineering applications. MATLAB® is renowned for its powerful matrix operations, built-in functions and data visualization. MathWorks has provided MATLAB® programmers with a set of user interface control objects (uicontrol objects)[7] that can be used to create GUIs. With GUIs, sophisticated programs can be presented by minimal, simple and

[7] (Marchand and Holland 2002, 391)

intuitive graphical components similar to those seen in MS Windows®. Events are triggered by the users' interactions with the GUIs, and automation is thus achieved. Figure 2.2 shows an example of a GUI created in MATLAB®.

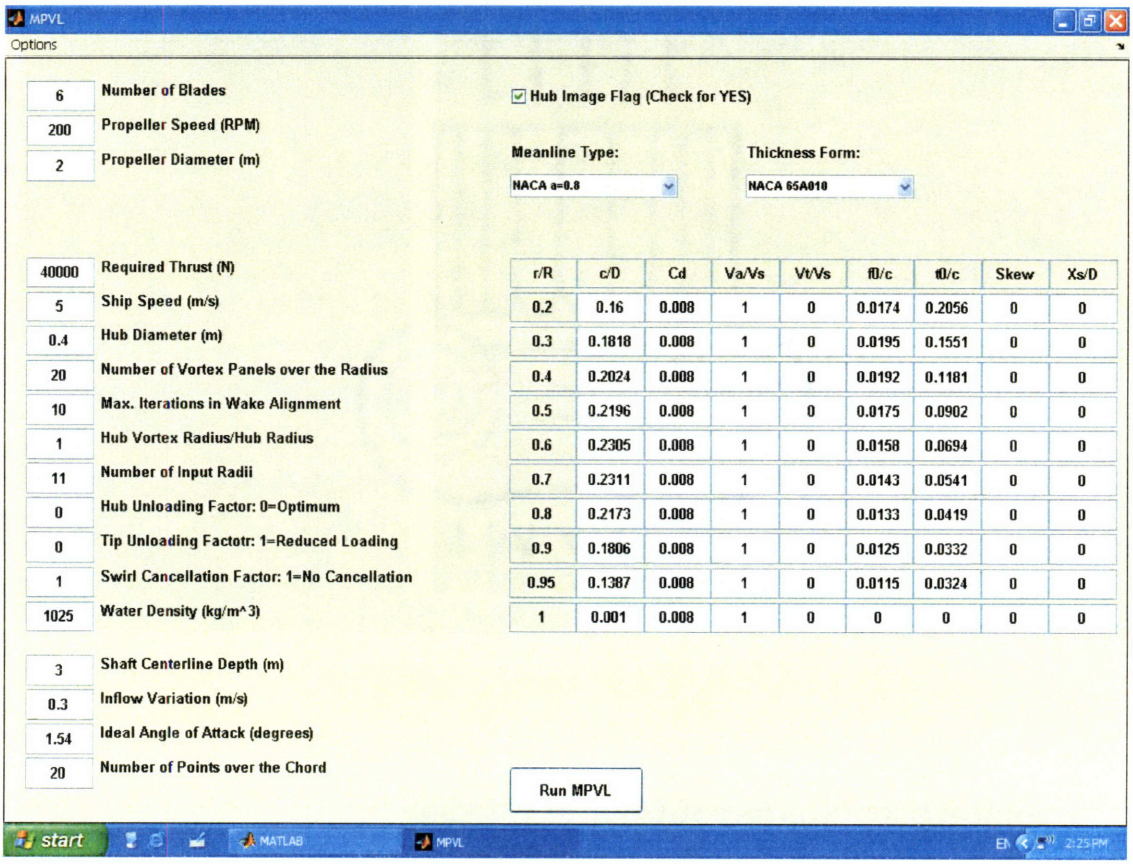


Figure 2. 2 Example of a Graphical User Interface in MATLAB®

There are two approaches to create GUIs in MATLAB®. One is the high-level developing technique which utilizes the Graphical User Interface Development Environment (GUIDE)[8]. The other one is to program directly in M-files which is a low-level technique. Both approaches were introduced in the subsequent subsections. The high-level approach which uses GUIDE is more point-and-click. It is suitable for creating simple GUIs which have a small number of graphical objects. On the other hand, the low-level approach consists of pure hand-programming in M-files to create GUIs. All graphical objects are created in the M-files, and their properties are defined by lines of programming commands. Both approaches have their own advantages and can be adopted based on the requirements or the preferences of the programmers.

[8] (Marchand and Holland 2002, 450)

2.2.1 High-Level GUI Development

The high-level GUI development in MATLAB® utilizes the Graphical User Interface Development Environment (GUIDE). GUIDE is a set of tools that allows MATLAB® users to lay out GUIs by simply clicking and dropping objects, then dragging and resizing them in the manner in which the users want them to appear. It also includes a Property Inspector[9] that allows the users to edit the properties of any graphical object. The properties of a graphical object include the size, position, font size, visibility, etc. depending on what type of graphical object it is (e.g., a pushbutton, a pop-up menu or a radio button). Figure 2.3 shows the GUIDE platform and the Property Inspector window in MATLAB®. Editing properties of the objects is easy and intuitive in GUIDE. As the users save the file, GUIDE creates a FIG-file and an M-file of the same name. The FIG-file contains the physical layout of the GUI while the M-file contains the callback function prototypes which are the heart of the GUI. Users then edit the callback function prototypes in the M-file to decide what event(s) to be triggered when a graphical object is called.

GUIDE is a highly efficient tool for creating GUIs which have a small number of graphical objects. Because it is both simple and intuitive, it is recommended for beginners. In chapter three, VLL and VLM were rewritten in MATLAB® and presented with GUIs created in GUIDE to demonstrate the feasibility and simplicity of implementing the hydrofoil design applications with GUIs.

[9] (Marchand and Holland 2002, 452)

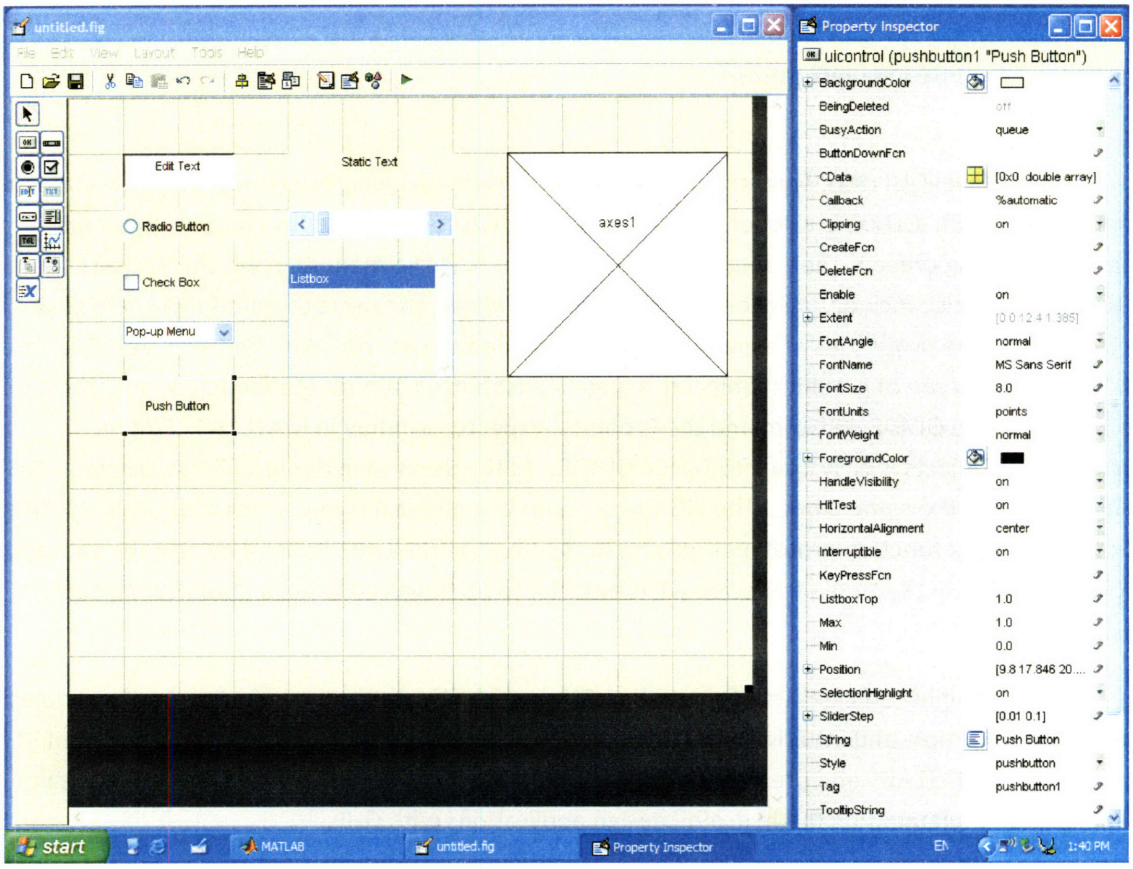


Figure 2. 3 Graphical User Interface Development Environment (GUIDE) in MATLAB

2.2.2 Low-Level GUI Development

The low-level GUI development involves pure hand-programming in M-files. Lines of programming commands define the properties of the graphical objects and the functions of the GUIs. Unlike creating GUIs in GUIDE, low-level GUI programming is less intuitive and requires more programming knowledge. However, it is very efficient when the objective GUIs contain a large number of graphical objects. With a large number of graphical objects, clicking and dropping objects, then dragging, resizing and aligning them becomes a tedious and time consuming task. Thus creating GUIs with M-files is more efficient when the graphical objects are similar, and thus the **for** loops can be utilized to duplicate identical objects. The low-level GUI developing approach was adopted to create the GUIs for MPVL because MPVL GUIs contained a large number of input fields. Figure 2.4 shows part of the code that creates the MPVL GUIs.

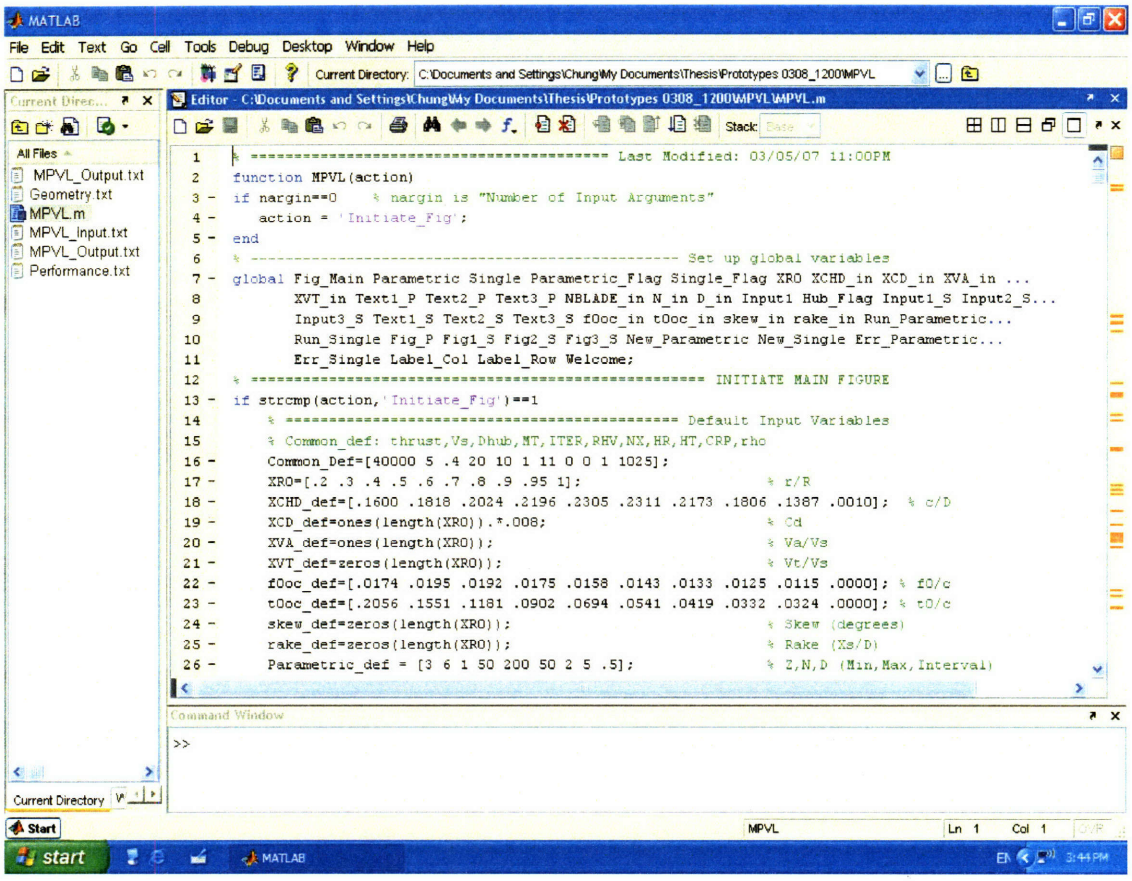


Figure 2. 4 Low-level GUI Development in MATLAB

In MPVL, input fields were stored in matrices (See Figure 2.5.) rather than as standalone variables when creating the uicontrol objects (Graphical objects are referred as uicontrol objects in MATLAB®.). Though later on in the calculation process, the matrices were broken into standalone variables. By referring to the matrix index, the **for** loops can be easily applied to create multiple uicontrol objects (See Figure 2.6.) and to group the objects in a block (See Figure 2.2.). Since manipulating a block of objects is more efficient than manipulating one object at a time, this programming strategy saved the programmers a great deal of work. Moreover, extra input fields can be added to the GUIs by simply adding them to the matrix of input fields. New uicontrol objects can be created either by copying the code of the existing uicontrol objects, pasting it and modifying it, or by adding new code.

For example, the atmosphere pressure was added as an input field in the single propeller design GUI. First, the **MPVL.m** file was opened in the Editor window in MATLAB®, and the default value (101 in kPa) was added to the matrix **Single_def2** as shown in Figure 2.5. A text label was required for describing the input field and was added to the corresponding string **Label2_S** also seen in Figure 2.5. **Single2_S** and **Label2_S** were names given by the programmer to distinguish each matrix and to pass their contents in the program.

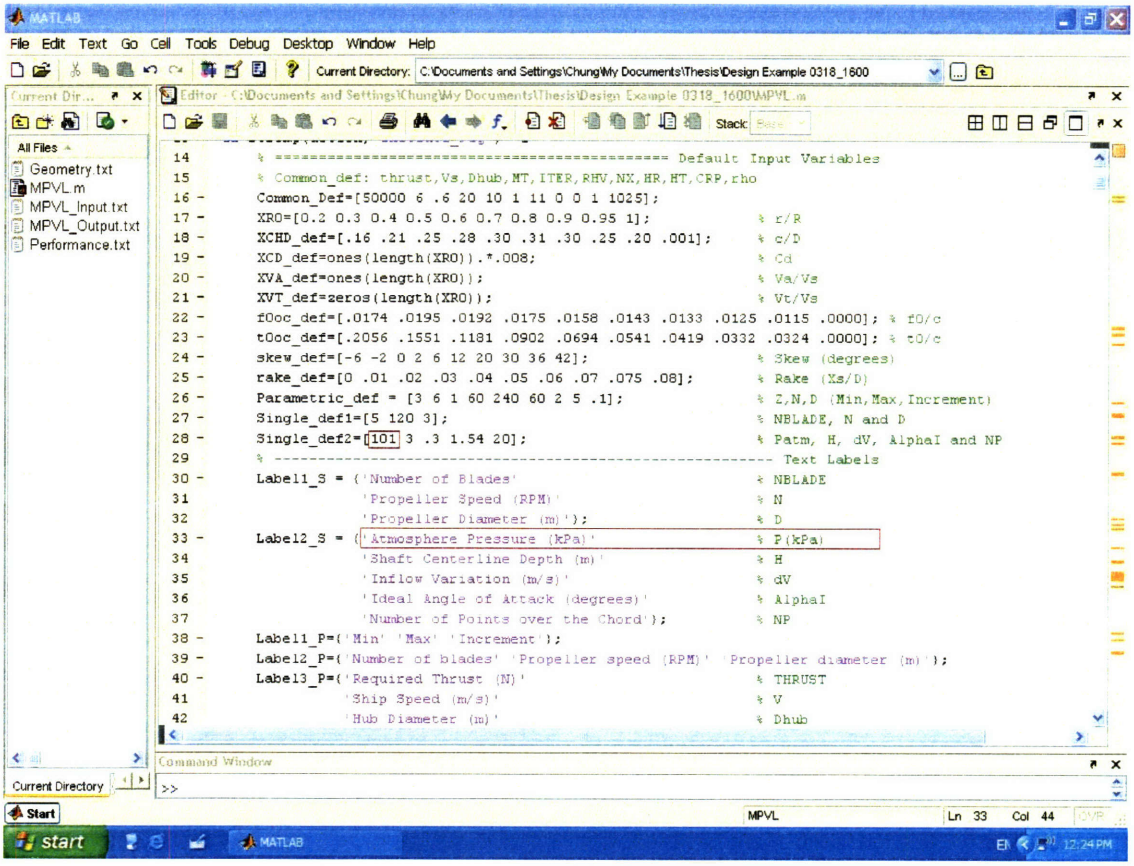


Figure 2. 5 Example of Modification of the MPVL GUI

In Figure 2.6, **Input2_S** and **Text2_S** were the handles to the graphical objects in the GUI. The graphical objects were created by the built-in function "**uicontrol**", and their properties were defined by the settings inside the parentheses. Property names and their values were listed inside the parentheses and separated by the comma. For example, for **Input2_S** the property "**style**" was followed by the value "**edit**", which meant this uicontrol object was an editable text box. The property "**units**" was followed by the value "**normalized**", which meant the units of the extent and position properties of the object were measured from the lower-left corner of the parent object (0,0) to the upper-right corner (1,1). This way the uicontrol objects can be scaled proportionally. The property "**FontSize**" was followed by an integer ten. The property "**FontWeight**" was followed by the value "**bold**." The property "**BackgroundColor**" was followed by "**w**", which meant white. The property "**callback**" was followed by "**MPVL("Update")**", which meant that when this object was activated (The text was edited in the case of an editable text box.), the function **MPVL("Update")** was then called to update the matrix to the current contents of the editable text boxes. The property "**position**" was followed by a set of numbers in the brackets. The set of numbers defined the location and size of the object and was specified as **[left bottom width height]**. **left** and **bottom** are the distance from the lower-left corner of the parent object to the lower-left corner of the uicontrol object. **width** and **height** are the dimensions of the uicontrol object. All measurements were in the units specified by the property "**units**" mentioned above. In this

case, **X1**, **Y3**, **h1**, **ew** and **eh** contained deliberate numbers so that the same values could be called repeatedly and could be modified easily. See Figure 2.7 for illustration. The property "**string**" was the text displayed on objects such as editable text, static text, push buttons, check boxes, radio buttons and toggle buttons. It was followed by the matrix **Single_def2** in this case. This way the contents of **Single_def2** were then displayed on the editable text boxes. The property "**visible**" was followed by the value "**off**." That meant the created object was not visible at this time. The value can be changed to "**on**" to make the object visible when needed. **Text2_S** was the text labels coupled with the editable text boxes **Input2_S** (See Figure 2.7.), and its properties were defined in a manner similar to **Input2_S**. Notice that different uicontrol objects have different properties. The objects and their properties can be looked up in the MATLAB® help menu.

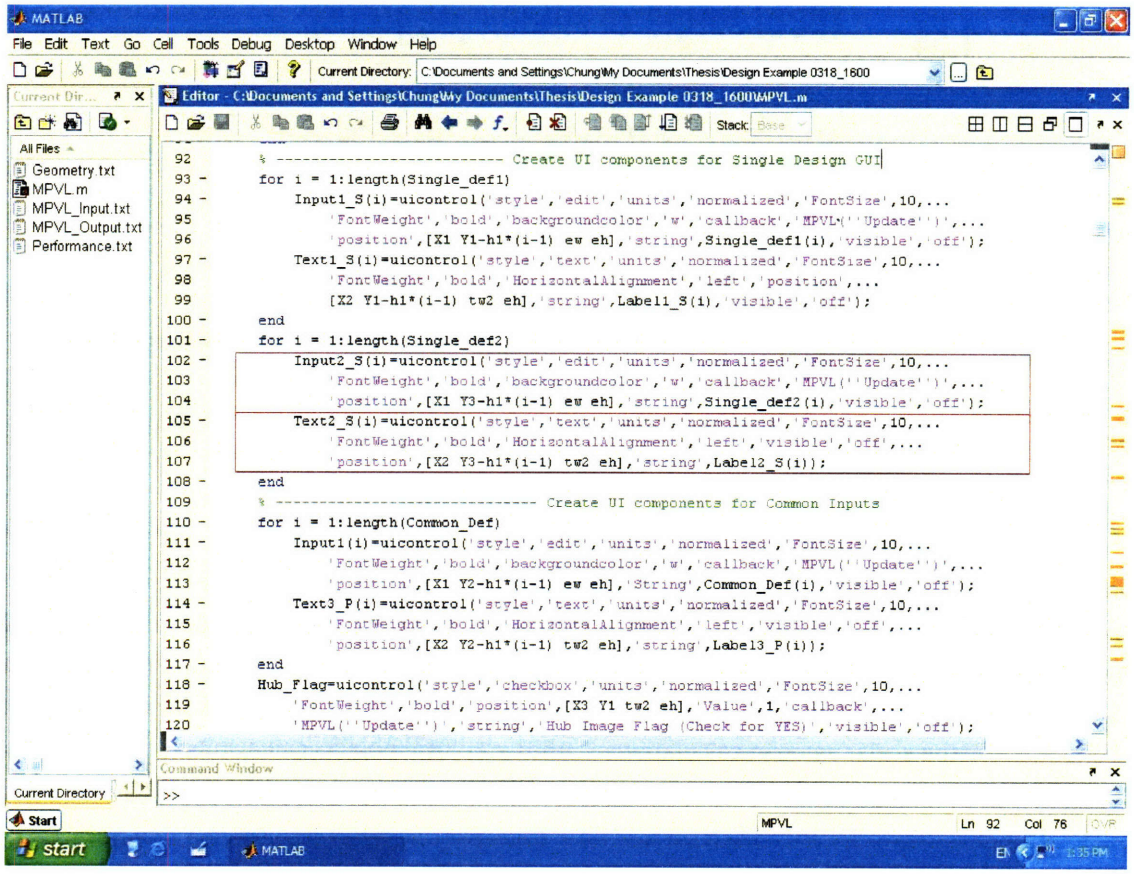


Figure 2. 6 Example of Creating Uicontrol Objects

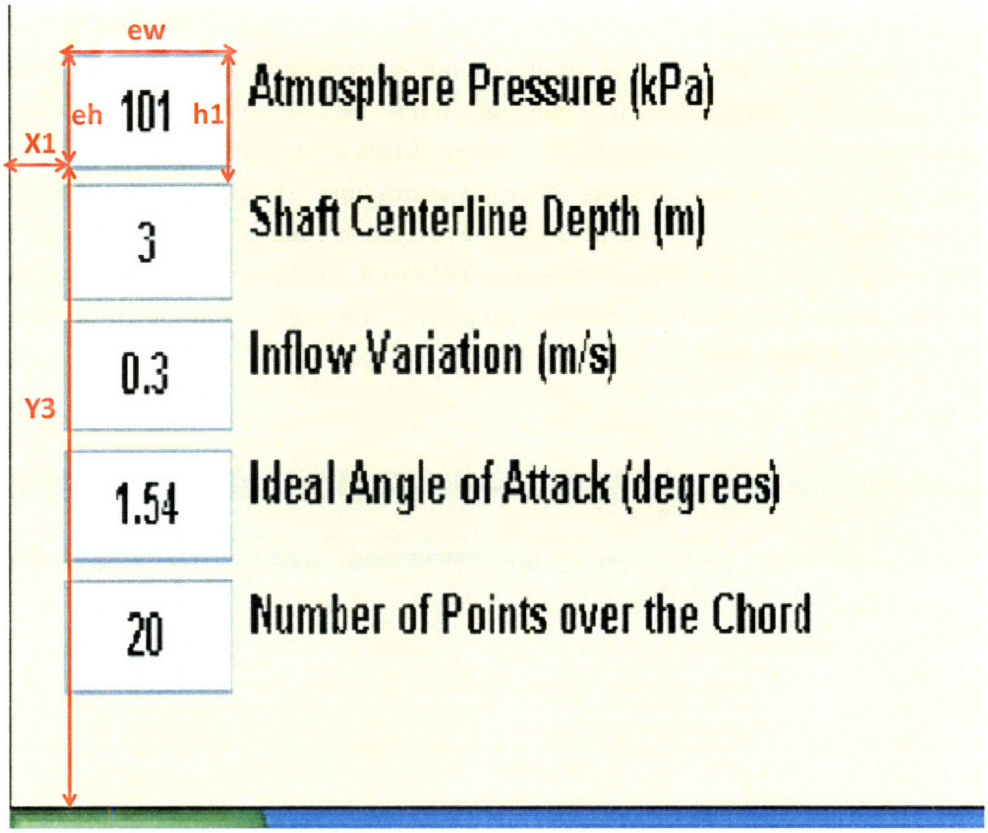


Figure 2. 7 Notation of Position Property

Once the code editing process was finished, the **MPVL.m** file was then saved and executed in MATLAB®. The newly added input field (atmosphere pressure) and its text label were shown in the single propeller design GUI as seen in Figure 2.8. When the "Run MPVL" button was clicked, **Input2_S(1)** contained the value of the atmosphere pressure shown on the screen. **Input2_S(2)** contained the value of the shaft centerline depth and so on. A name can then be assigned to **Input2_S(1)** in the **MPVL.m** file to pass and identify the scalar value in calculations. Other graphical objects can also be created in the similar manner. Thus MPVL can be easily modified according to the users' specific interests.

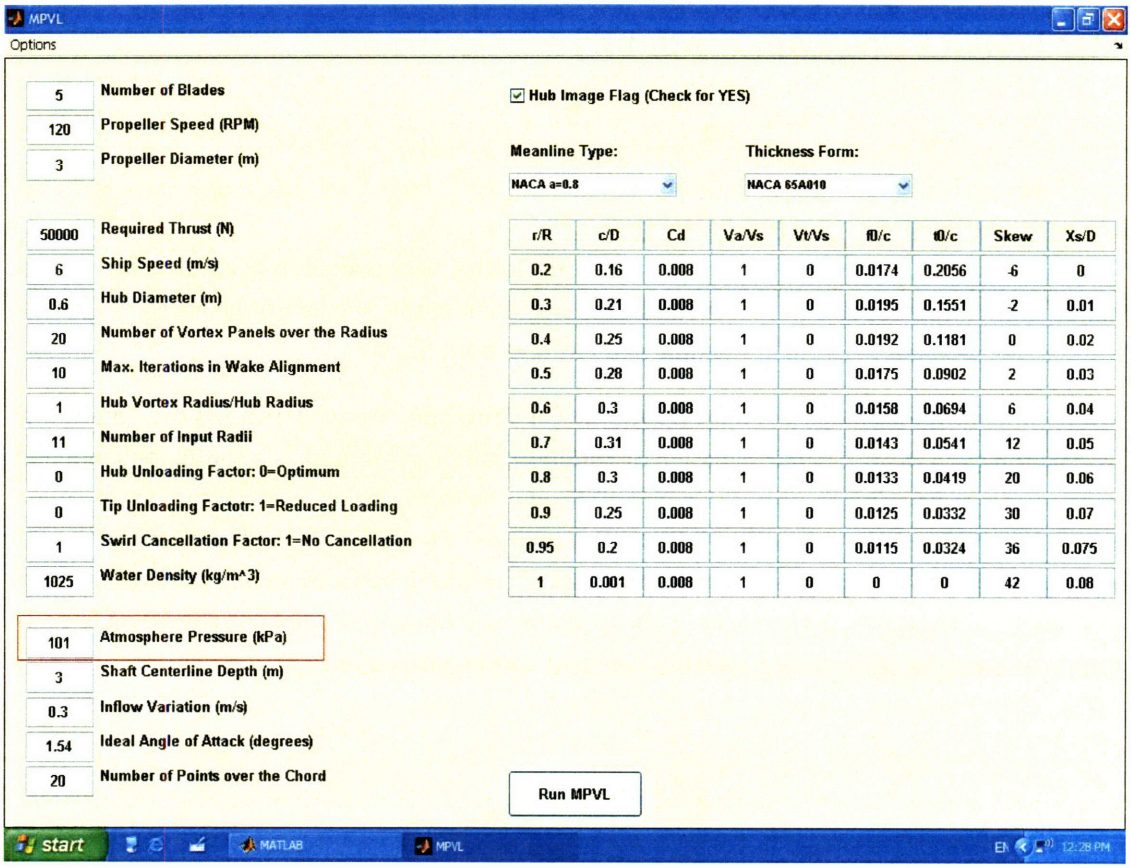


Figure 2. 8 Result of the Modification

3 Hydrofoil Design Applications

3.1 Hydrofoil Vortex Lattice Lifting Line Program (VLL)

VLL is a FORTRAN code that demonstrates vortex lattice approximation of a lifting line for two-dimensional foils[10]. VLL was selected as the first step in implementing the hydrofoil design programs with GUIs. It was selected because it is simple and has few input fields.

A two-dimensional wing can be represented by a lifting line. Figure 3.1 shows the notation of the classical lifting line theory. The circulation distribution can be presented as a continuous function with the Glauert's coefficients[11]. This is the analytical solution to the lifting line theory. On the other hand, the vortex lattice method provides a numerical solution. Figure 3.2 shows the vortex lattice lifting line model. The vortex lattice method divides the span of the lifting line into a finite number of vortex panels. If there are M panels, there are M+1 vortex points and M control points on the lifting line. The circulation is in a stepped distribution which is constant within each panel.

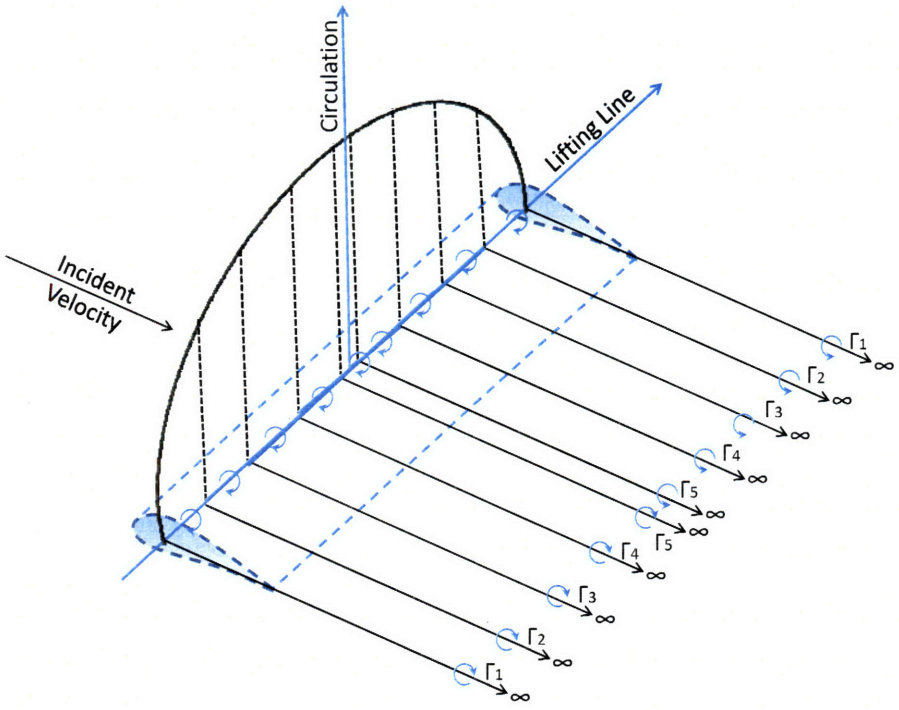


Figure 3. 1 Notation of Classical Lifting Line Theory

[10] (Kerwin 2001, 99)
[11] (Glauert 1926)

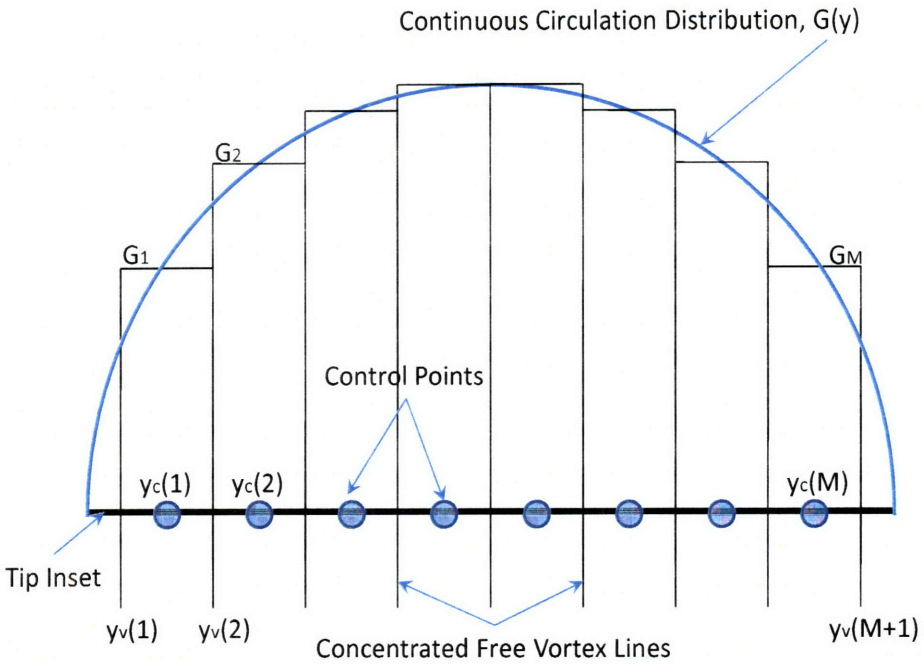


Figure 3. 2 Vortex Lattice Lifting Line Model

VLL requires the Glauert's coefficients from the users to calculate the analytical solution for the circulation distribution and the downwash velocity over the span of the lifting line. With the analytical solution known, VLL then sets up the vortex lattice lifting line model and calculates the numerical solution for the circulation distribution and the downwash velocity. Therefore, numerical error can be expected and is calculated by VLL.

In order to determine the circulation distribution and the downwash velocity over the span of the lifting line, the vortex points and control points must be defined first. Two spacing methods are provided for placing the vortex points: the cosine spacing[12] and the constant spacing. For the cosine spacing, a spanwise variable $\tilde{y}$ is related to the physical coordinate y as shown in Equation 3.1.

$$y = -\frac{s}{2}\cos(\tilde{y}) \qquad (3.1)$$

Notice that y = -s/2 when $\tilde{y} = 0$, and y = s/2 when $\tilde{y} = \pi$. Thus the vortex points can be angle-wise equally spaced. On the other hand, the constant spacing divides the span into length-wise equally spaced panels with an inset at each tip (See Figure 3.2.). The control points can be placed by two spacing methods: the cosine spacing and the mid-point spacing. The cosine spacing places a control

[12] (Lan September 1974)

point in the mid-angle point between two vortex points while the mid-point spacing places a control point at the middle of the vortex panel.

With the Glauert's coefficients, the spanwise circulation distribution can then be presented by the series shown in Equation 3.2.

$$\Gamma(y) = 2Us \sum_{n=1}^{\infty} a_n \sin(n\tilde{y}) \tag{3.2}$$

The downwash velocity on the lifting line can be computed by Equation 3.3.

$$\omega^*(y) = -U \sum_{n=1}^{\infty} \frac{n a_n \sin(n\tilde{y})}{sin\tilde{y}} \tag{3.3}$$

The incident velocity, *U*, and the foil span, *s*, were set to one in VLL for non-dimensional calculation. With the analytical solution known, the downwash velocity from the vortex lattice method can be calculated by Equation 3.4.

$$\omega^*(y_c(n)) = \sum_{m=1}^{M} \Gamma_m \omega_{n,m} \tag{3.4}$$

The influence function, $\omega_{n,m}$, is defined in Equation 3.5.

$$\omega_{n,m} = \frac{1}{4\pi(y_v(m) - y_c(n))} - \frac{1}{4\pi(y_v(m+1) - y_c(n))} \tag{3.5}$$

Figure 3.3 shows an example of the VLL GUI. The VLL GUI is launched on the screen after the **VLL.m** file is executed in MATLAB®. Default values of the input fields are defined in GUIDE and present when the GUI is launched. The users can use or modify the default input values. When the "Run VLL" button in the GUI is clicked, the input fields are read and fed to the algorithm in the **Main.m** file. When the calculation process is complete, the analytical and numerical solutions for the circulation distribution and downwash velocity are plotted vs. the non-dimensional spanwise position as shown on the right of the GUI (See Figure 3.3.). An output text file is automatically created to store the detailed information of the run. Table 3.1 shows an example of the VLL output text file. The text file can be saved or printed as desired. The users can then either modify the input fields in the GUI and run the program again, or click the "Print Screen" button in the GUI to print the current GUI, or click the "Exit" button to terminate the GUI. The MATLAB® code of VLL is listed in Appendix A.

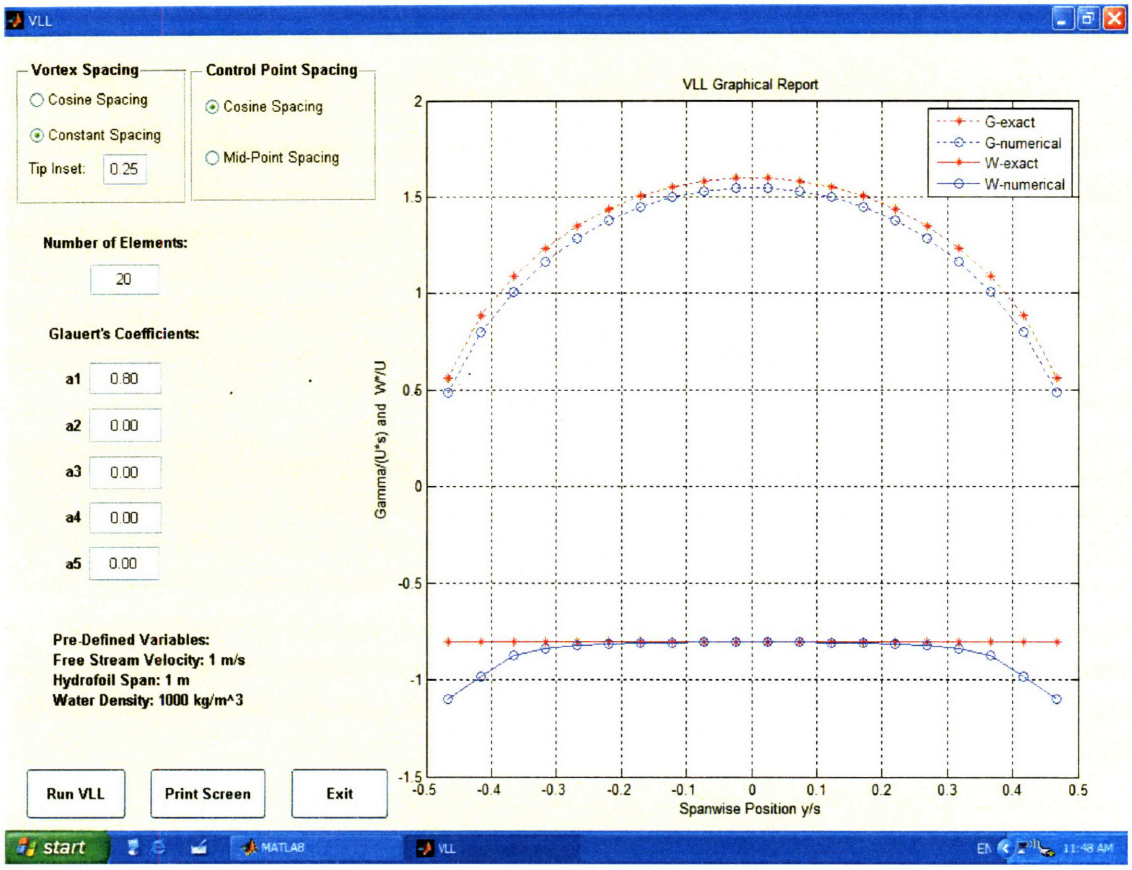


Figure 3. 3 VLL Graphical User Interface

```
VORTEX LIFTING LINE SOLUTION WITH 10 ELEMENTS

CIRCULATION COEFFICIENTS
 1.000    0.000    0.000    0.000    0.000
Cosine spaced vortices and control points
N        YV          YC         GAMMA       W(EXACT)     W(NUM)       G(NUM)
1     -0.5000     -0.4938     +0.3129     -1.0000     -0.9959     +0.3142
2     -0.4755     -0.4455     +0.9080     -1.0000     -0.9959     +0.9117
3     -0.4045     -0.3536     +1.4142     -1.0000     -0.9959     +1.4200
4     -0.2939     -0.2270     +1.7820     -1.0000     -0.9959     +1.7894
5     -0.1545     -0.0782     +1.9754     -1.0000     -0.9959     +1.9835
6     -0.0000     +0.0782     +1.9754     -1.0000     -0.9959     +1.9835
7     +0.1545     +0.2270     +1.7820     -1.0000     -0.9959     +1.7894
8     +0.2939     +0.3536     +1.4142     -1.0000     -0.9959     +1.4200
9     +0.4045     +0.4455     +0.9080     -1.0000     -0.9959     +0.9117
10    +0.4755     +0.4938     +0.3129     -1.0000     -0.9959     +0.3142
      +0.5000
PERCENT ERROR IN NUMERICAL SOLUTION:
COMPUTING DOWNWASH FOR GIVEN CIRCULATION
LIFT: -0.41   DRAG: -0.82   DRAG/LIFT^2:   0.00

COMPUTING DOWNWASH FOR GIVEN DOWNWASH
LIFT: -0.00   DRAG: -0.00   DRAG/LIFT^2:   0.00
```

Table 3. 1 Example of VLL Output Text File

3.2 Two-Dimensional Vortex Lattice Method Program (VLM)

VLM was the second hydrofoil design application to be implemented in MATLAB®. The algorithm of VLM is slightly more sophisticated than that of VLL. VLM presents a vortex lattice solution to the two-dimensional foil problem. Figure 3.4 shows the notation of a vortex lattice model for a two-dimensional foil. The vortex lattice method was first developed by Faulkner[13], who divided the chord of the foil into equally spaced panels, placed the vortex on each panel and computed the velocity induced by the vortex at the control points. Actually, the vortex lattice method can be considered as a subclass of the lifting surface models[14].

For an arbitrary two-dimensional foil, it is known that the velocity on the surface is dominated by the angle of attack, the camber distribution and the thickness distribution. Therefore, those three parameters are the sole input fields of VLM. Once the velocities on the surface of the foil are solved, VLM then calculates the pressure coefficient (–Cp) on the upper and lower surfaces of the foil. The meanline type and the thickness form of the foil are respectively defined in the **Meanline.m** and **Thickness.m** files and can be easily modified. So far, the default meanline type is the NACA a=0.8[15], and the default thickness form is the NACA 66 (mod)[16]. More meanline types and thickness forms can be added to the program to provide more options in the future.

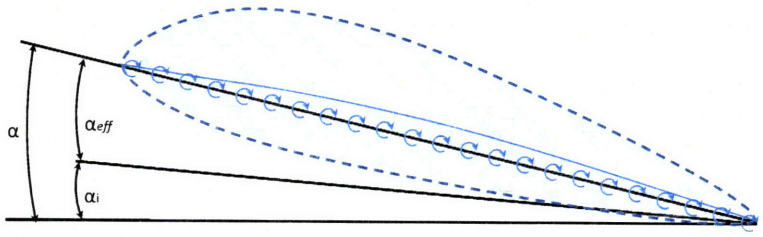


Figure 3. 4 Two-Dimensional Vortex Lattice Model

[13] (Faulkner 1947)
[14] (Carlton 1994, 158)
[15] (Abbott and Von Doenhoff 1959, 403)
[16] (Carlton 1994, 41)

In the vortex lattice method, the chord is divided into a finite number of vortex panels. Then the vortices and the control points are placed on the panels by the cosine spacing method shown in Equation 3.6 and 3.7.

$$x_v(n) = \frac{c}{2}\left[1 - cos\left(\frac{(n-\frac{1}{2})\pi}{N}\right)\right] \tag{3.6}$$

$$x_c(n) = \frac{c}{2}\left[1 - cos\left(\frac{n\pi}{N}\right)\right] \qquad n = 1,\ldots, N \tag{3.7}$$

Notice that c is the chord length, and N is the number of panels on the chord. The circulation can then be solved by Equation 3.8.

$$-\frac{1}{2\pi}\sum_{m=1}^{N}\frac{\Gamma_m}{x_v(m)-x_c(n)} = U\left[\frac{df}{dx_c(n)} - (\alpha - \alpha_i)\right] \tag{3.8}$$

U is the incident velocity, df/dx is the slope of the camber, α is the angle of attack, and α_i is the ideal angle of attack. The incident velocity, U, and the chord length, c, were set to one in VLM for non-dimensional calculation. With the circulation distribution solved, the vortex sheet strengths can be presented as Equation 3.9.

$$\gamma\left(x_v(n)\right) = \frac{N\Gamma_n}{\pi\sqrt{x_v(n)(c-x_v(n))}} \tag{3.9}$$

Furthermore, the thickness induced velocity on the surface of the foil can be solved by Equation 3.10.

$$\frac{u_t(n)}{U} = \frac{1}{2\pi}\sum_{m=1}^{N}\frac{y_c(m)-y_c(m-1)}{x_c(n)-x_v(m)} \tag{3.10}$$

The non-dimensional velocity on the foil surface can be calculated by Equation 3.11.

$$\frac{q(x)}{U} = \left[1 + \frac{u_t(x)}{U} + \frac{u_c(x)}{U}\right]\sqrt{\frac{x}{x+r_L/2}} + (\alpha - \alpha_{ideal})\sqrt{\frac{c-x}{x+r_L/2}} \tag{3.11}$$

Notice that r_L is the radius of the leading edge. With the Lighthill correction[17], the non-dimensional velocity at the leading edge has the finite value as shown in Equation 3.12.

$$\frac{q(0)}{U} = (\alpha - \alpha_{ideal})\sqrt{\frac{2}{r_L/c}} \tag{3.12}$$

Finally, the pressure coefficient on the surface can be presented as Equation 3.13.

$$C_P(x) = 1 - \left(\frac{q(x)}{U}\right)^2 \tag{3.13}$$

Figure 3.5 shows an example of the VLM GUI. When the **VLM.m** file is executed in MATLAB®, the VLM GUI is launched and the default input values are present. VLM acquires only four input fields

[17] (Kerwin 2001, 50)

from users: the number of panels on the chord, the ideal lift coefficient, the angle of attack (α- α_i) and the maximum thickness ratio. The ideal lift coefficient is used to scale the maximum camber ratio. By clicking the "Run VLM" pushbutton in the GUI, the input fields are read and fed to the algorithm in the **Main.m** file. When the calculation process is complete, the pressure coefficient (–Cp) on the upper and lower surfaces of the foil is plotted on the right of the GUI (see Figure 3.5.). The circulation distribution, the thickness induced velocity, the camber distribution and the thickness distribution are plotted in another figure (See Figure 3.6.). An output text file (See Table 3.2.) is automatically created to store the detailed information of the run. This text file can be saved and printed as desired. The MATLAB® code of VLM is listed in Appendix B.

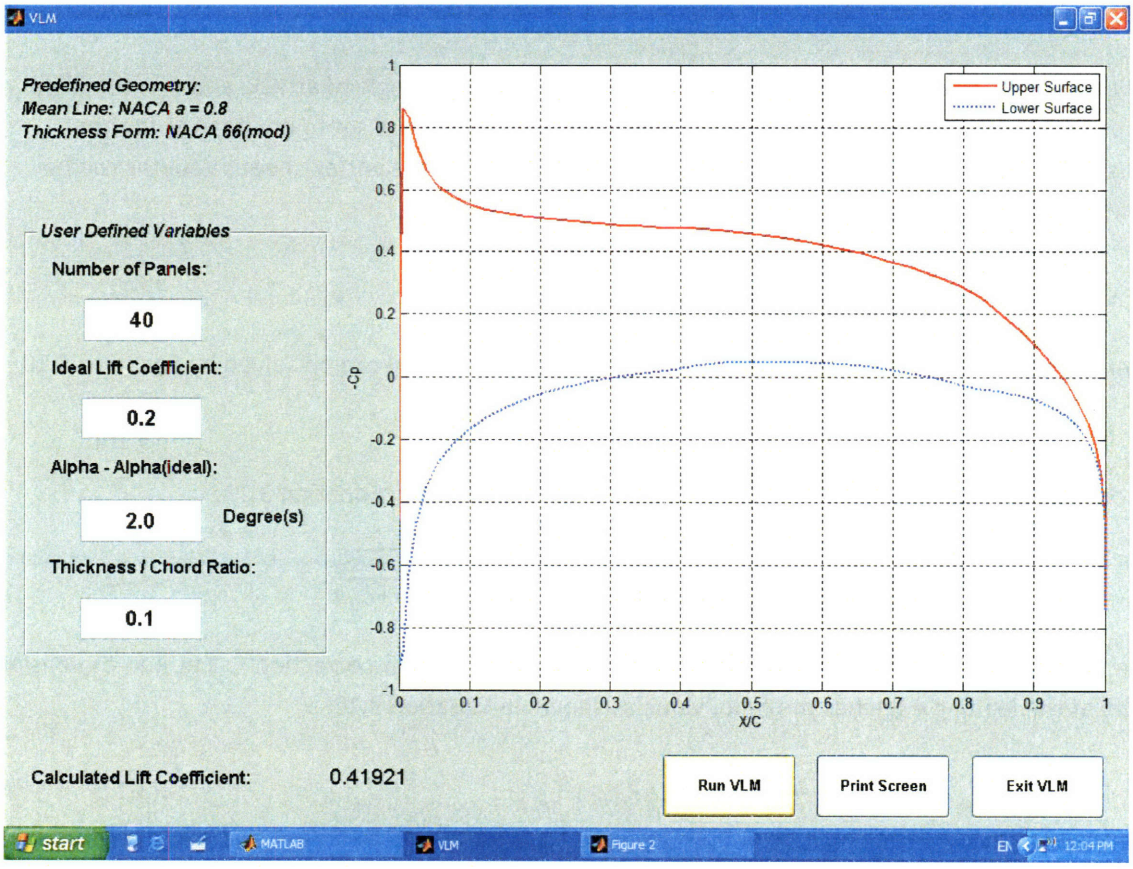


Figure 3. 5 VLM Graphical User Interface

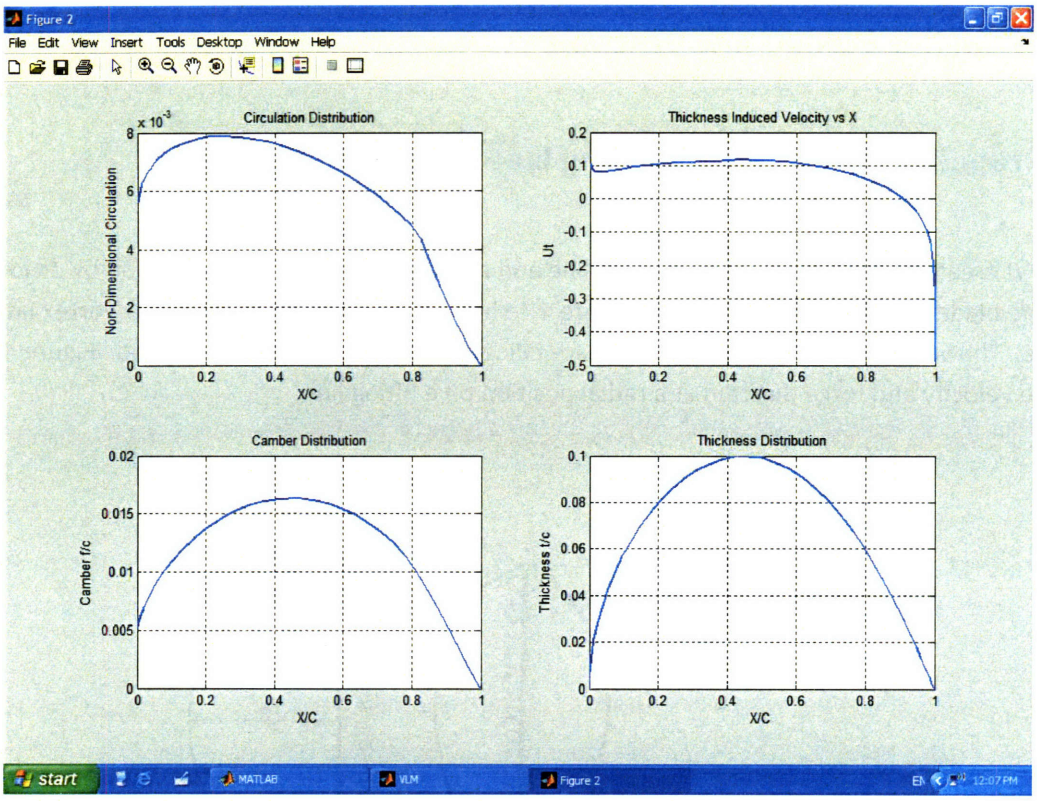


Figure 3. 6 VLM Graphical Report

```
2D Vortex Lattice with Lighthill Correction Program (VLM)

Number of Panels: 20
Ideal Lift Coefficient:  0.10
Angle of Attack (Alpha-AlphaIdeal):   1.0 degree
Maximum Thickness/Chord Ratio (T/C):  0.10
Radius of Leading Edge (RLE/C): 0.00448
Calculated Lift Coefficient (CLNum): 0.20945
```

x/c	f/c	t/c	Gamma	G	UT	UTVP	CPU	CPL
0.0015	0.0028	0.0000	+0.0058	+0.9395	+0.0897	+0.0941	-0.8640	-0.8640
0.0138	0.0033	0.0147	+0.0064	+0.3499	+0.0813	+0.0847	-0.0007	-0.8421
0.0381	0.0041	0.0290	+0.0069	+0.2308	+0.0848	+0.0815	+0.3703	-0.2868
0.0737	0.0049	0.0431	+0.0074	+0.1791	+0.0915	+0.0883	+0.3651	-0.1094
0.1198	0.0058	0.0568	+0.0077	+0.1500	+0.0995	+0.0953	+0.3569	-0.0235
0.1753	0.0066	0.0697	+0.0078	+0.1312	+0.1058	+0.1030	+0.3577	+0.0330
0.2388	0.0073	0.0810	+0.0079	+0.1178	+0.1098	+0.1081	+0.3600	+0.0725
0.3087	0.0078	0.0900	+0.0078	+0.1077	+0.1128	+0.1112	+0.3598	+0.0995
0.3833	0.0081	0.0963	+0.0076	+0.0997	+0.1165	+0.1148	+0.3584	+0.1192
0.4608	0.0082	0.0997	+0.0073	+0.0932	+0.1161	+0.1170	+0.3599	+0.1372
0.5392	0.0080	0.0992	+0.0069	+0.0877	+0.1101	+0.1137	+0.3519	+0.1438
0.6167	0.0076	0.0946	+0.0063	+0.0829	+0.0996	+0.1054	+0.3316	+0.1369
0.6913	0.0069	0.0862	+0.0057	+0.0787	+0.0840	+0.0926	+0.3020	+0.1196
0.7612	0.0060	0.0746	+0.0050	+0.0748	+0.0631	+0.0738	+0.2619	+0.0912
0.8247	0.0047	0.0609	+0.0039	+0.0654	+0.0367	+0.0521	+0.2112	+0.0521
0.8802	0.0033	0.0462	+0.0024	+0.0463	+0.0040	+0.0169	+0.1438	+0.0081
0.9263	0.0020	0.0317	+0.0012	+0.0304	-0.0370	-0.0032	+0.0551	-0.0379
0.9619	0.0010	0.0188	+0.0005	+0.0176	-0.0917	-0.0912	-0.0432	-0.1017
0.9862	0.0003	0.0087	+0.0001	+0.0080	-0.1754	-0.0725	-0.1589	-0.1908
0.9985	0.0000	0.0022	+0.0000	+0.0018	-0.4093	-0.3347	-0.3135	-0.3267

Table 3. 2 Example of VLM Output Text File

4 Propeller Design Applications

4.1 Propeller Vortex Lattice Lifting Line Program (PVL)

PVL treats a propeller blade as a lifting line and applies the vortex lattice method by dividing the span of the blade into finite vortex panels. Figure 4.1 shows the notation of a propeller vortex lattice lifting line. The vortices and the control points are placed by the cosine spacing method. Figure 4.2 shows the velocity and force diagram at a radial position on a lifting line.

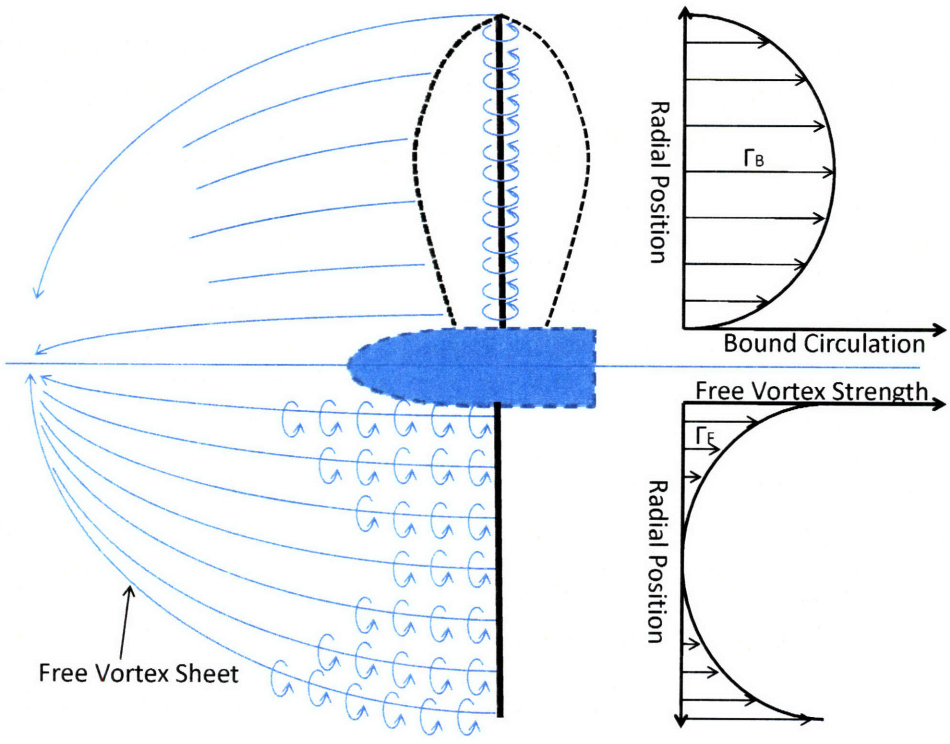


Figure 4. 1 Propeller Vortex Lattice Lifting Line Model

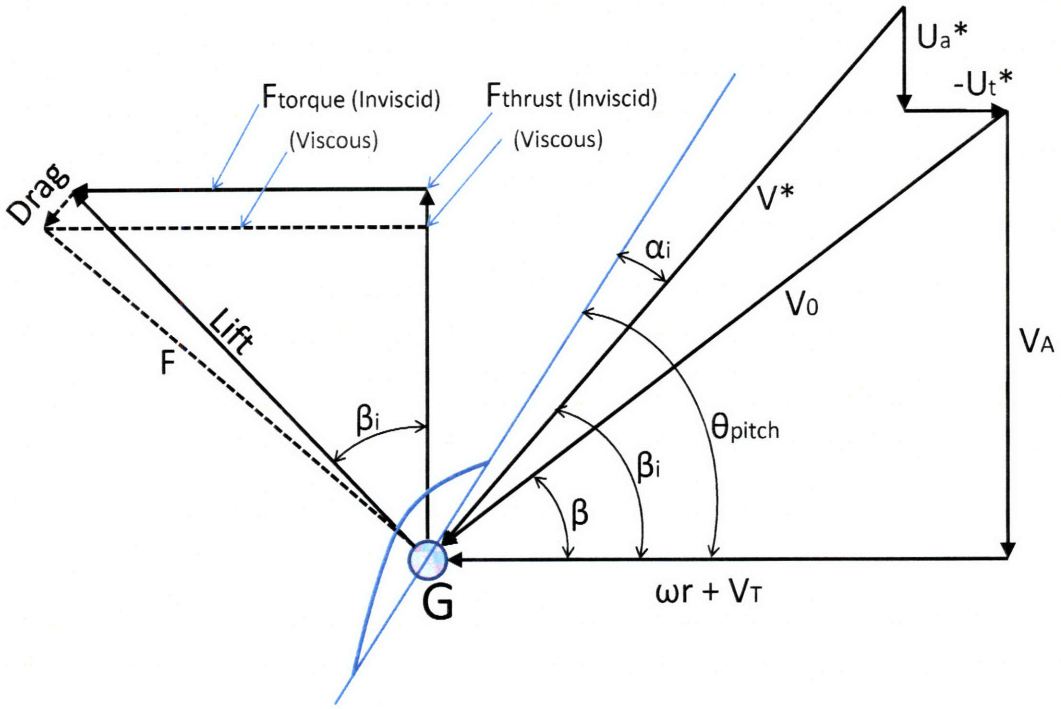


Figure 4. 2 Velocity and Force Diagram at a Radial Position on a Lifting Line

PVL first estimates the hydrodynamic pitch angle (β_i) based on the undisturbed flow angle (β) and the efficiency of the actuator disk. The circulation distribution can then be solved. The vortex induced velocities on each panel can be solved based on the circulation distribution. With the induced velocities known, the force calculation can be performed. Finally, the calculated thrust is compared with the desired thrust. The hydrodynamic pitch angle is then iteratively adjusted, and the process is repeated until the desired thrust is achieved.

Moreover, PVL is a wake-adapted[18] method which calculates the circumferential volumetric mean inflow which can vary with radius. PVL then optimizes the circulation and pitch distribution for this inflow. The users can choose to unload the tip or hub to reduce the susceptibility to cavitation, blade stress and blade-frequency pressure on the hull[19].

Figure 4.2 shows the graphical presentation of the algorithm. First, the undisturbed flow angle β at any radial position on the lifting line can be computed by Equation 4.1.

[18] (Edward 1988, 208)
[19] (Edward 1988, 184)

$$\tan\beta(r) = \frac{V_A(r)}{\frac{\pi r}{J} + V_T(r)} \quad (4.1)$$

J is the advance coefficient, V_A is the axial inflow velocity, and V_T is the tangential inflow velocity. The next step is to specify a trial value of the hydrodynamic pitch angle $\beta_i(r)$. $\beta_i(r)$ can either be the optimum distribution which satisfies Lerbs Criterion[20] as shown in Equation 4.2, or may be modified to unload the hub or the tip.

$$\frac{tan\beta(r)}{tan\beta_i(r)} = \Upsilon\sqrt{1-\omega_x(r)} \quad (4.2)$$

In either case, $tan\ \beta_i$ is derived from $tan\ \beta$ in terms of an unknown multiplier. The hub and tip unloading were discussed in chapter six. Since the desired thrust loading coefficient is specified, the efficiency of the actuator disk can be calculated. PVL then uses 90% of the efficiency of the actuator disk to determine $\beta_i(r)$. With $\beta_i(r)$ known, the circulation Γ_m can be solved by Equation 4.3.

$$\sum_{m=1}^{M}[\bar{u}_a(n,m) - \bar{u}_t(n,m)tan\beta_i(n)]\Gamma_m = \frac{V_a(n)}{V_s}\left(\frac{tan\beta_i(n)}{tan\beta(n)} - 1\right) \qquad n = 1,\ldots,M \quad (4.3)$$

Notice that $\bar{u}_a(n,m)$ and $\bar{u}_t(n,m)$ are the horseshoe influence functions. Moreover, the axial and tangential induced velocities at control point r_c can be computed by Equation 4.4 and 4.5.

$$u_a^*(r_c(n)) = \sum_{m=1}^{M}\Gamma_m\bar{u}_a(n,m) \quad (4.4)$$

$$u_t^*(r_c(n)) = \sum_{m=1}^{M}\Gamma_m\bar{u}_t(n,m) \quad (4.5)$$

The total inflow velocity at any radial position on the lifting line can be solved by Equation 4.6.

$$V^*(r) = \sqrt{(V_A(r) + u_a^*(r))^2 + (\omega r + V_T(r) + u_t^*(r))^2} \quad (4.6)$$

Finally, the generated thrust can be calculated by Equation 4.7.

$$Thrust = \rho Z\int_{r_h}^{R}\left[V^*\Gamma cos\beta_i - \frac{1}{2}V^{*2}cC_d sin\beta_i\right]dr \quad (4.7)$$

With the thrust calculated, the thrust loading coefficient can be solved and compared with the desired thrust loading coefficient. The multiplier in Equation 4.2 is then iteratively adjusted, and the calculation process is repeated until the desired thrust loading coefficient is achieved.

MPVL basically shares the same algorithm with PVL, but new features such as the GUIs and data visualization are included in MPVL. MPVL is intended to perform two tasks: the parametric analysis and the single propeller design. The parametric analysis takes combinations of the number of blades, propeller speed and propeller diameter, then calculates the efficiency for each combination and finally plots the efficiency curves. The efficiency curves show the change in propeller efficiency and thus help determine the optimum propeller parameters. On the other hand, the single propeller design produces

[20] (Kerwin 2001, 163)

a complete propeller design with detailed input information. Both applications were introduced in the following subsections. The MATLAB® code of MPVL was listed in Appendix C.

4.1.1 MPVL Program Flow Chart

The MPVL program flow chart is shown in Figure 4.3. After initiating the MPVL GUI (See Figure 4.4.) in MATLAB®, users select either the parametric analysis or the single propeller design option in the option menu. The selected GUI is then launched, and the default input values are present. When the users are satisfied with the input fields, they can click the "Run MPVL" button at the bottom of the GUI (See Figure 4.6) to perform the calculations. The "Print Screen" option in the option menu allows users to print the current GUI. The "Exit" option terminates the GUIs.

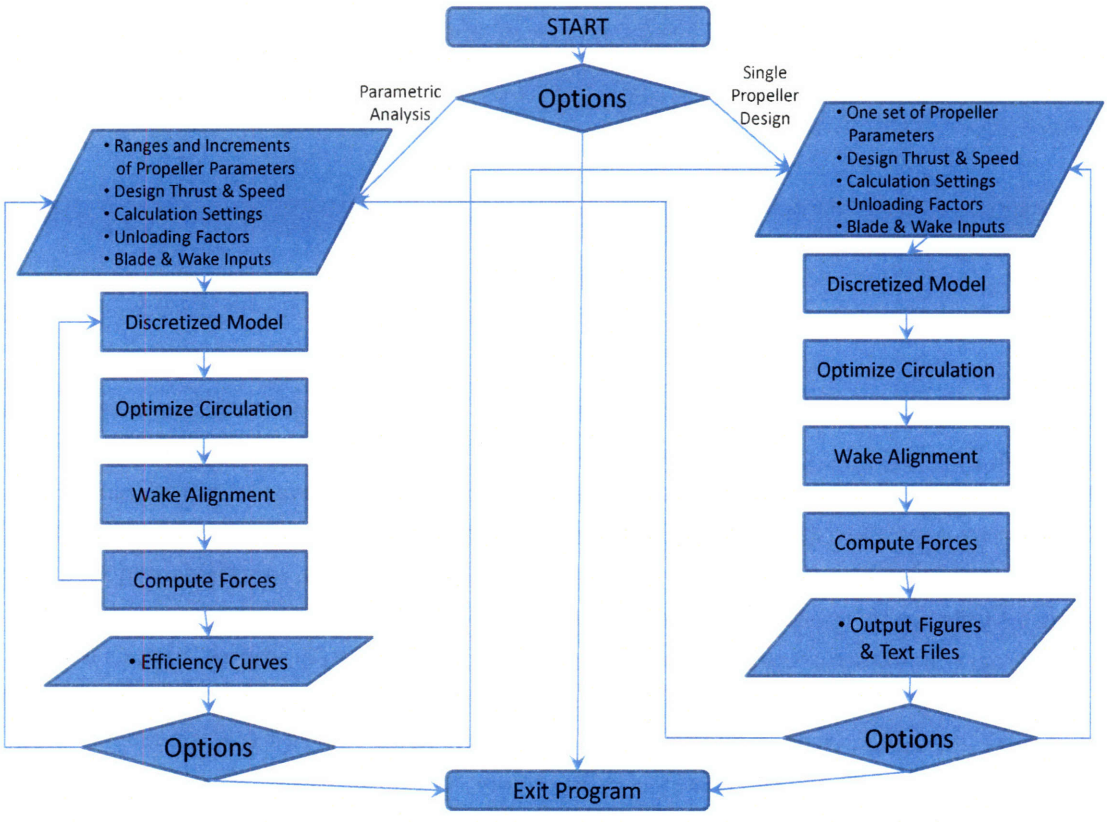


Figure 4. 3 MPVL Program Flow Chart

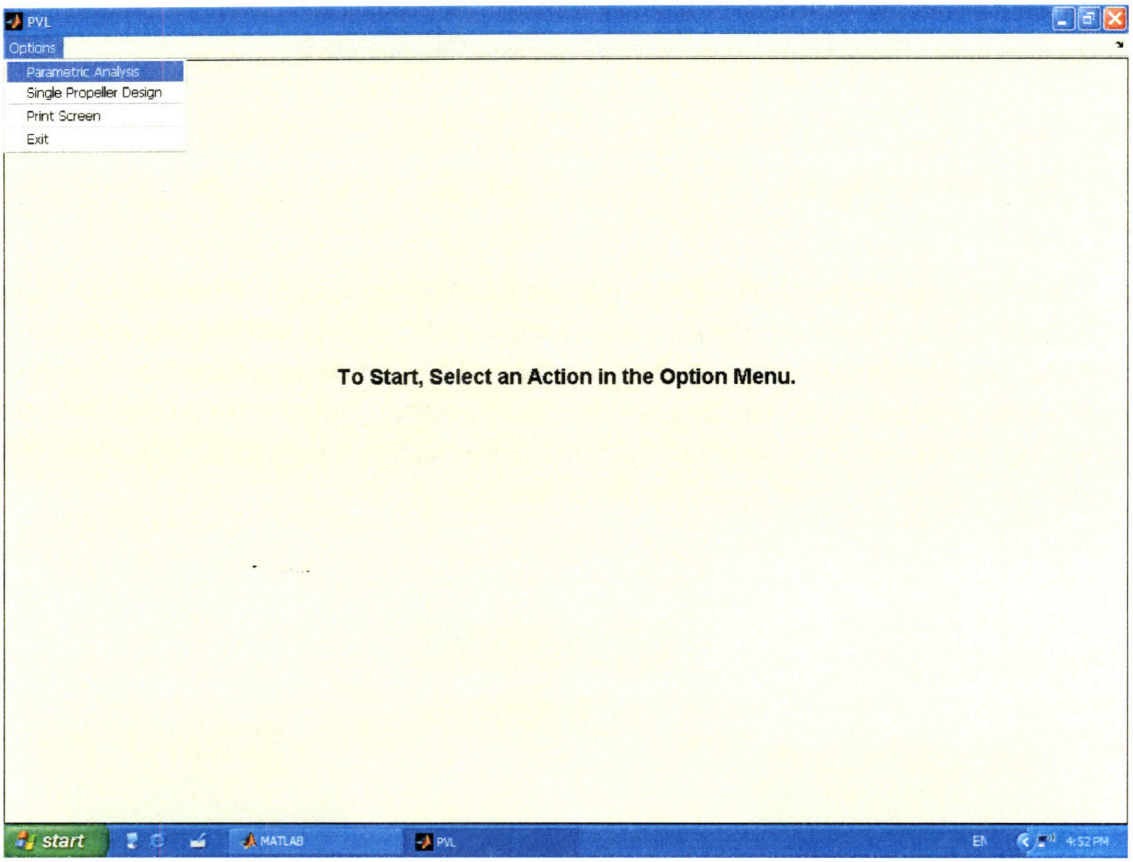


Figure 4. 4 MPVL Graphical User Interface

Both the algorithms for the parametric analysis and the single propeller design were contained in the **MPVL.m** file. This feature was enabled by adopting a specific programming structure called the "switchyard programming[21]". Figure 4.5 shows the structure of the switchyard programming. In this programming structure, the GUI callbacks call the same function **MPVL** repeatedly with different input arguments, and the **if** statements are then used to invoke different actions for different input arguments. Thus a single M-file is able to contain all the codes for performing different functions. By using the switchyard programming structure in MPVL, the codes for initiating the parametric analysis GUI and the single propeller design GUI, switching between the two GUIs and performing algorithms for the two GUIs were integrated and placed in one single M-file. Thus no additional M-file was needed though the users can still add their own add-on M-files as external functions in the future. Passing, operating and debugging the program was made easier by minimizing the number of M-files required for execution.

[21] (Hanselman and Littlefield 2005, 551)

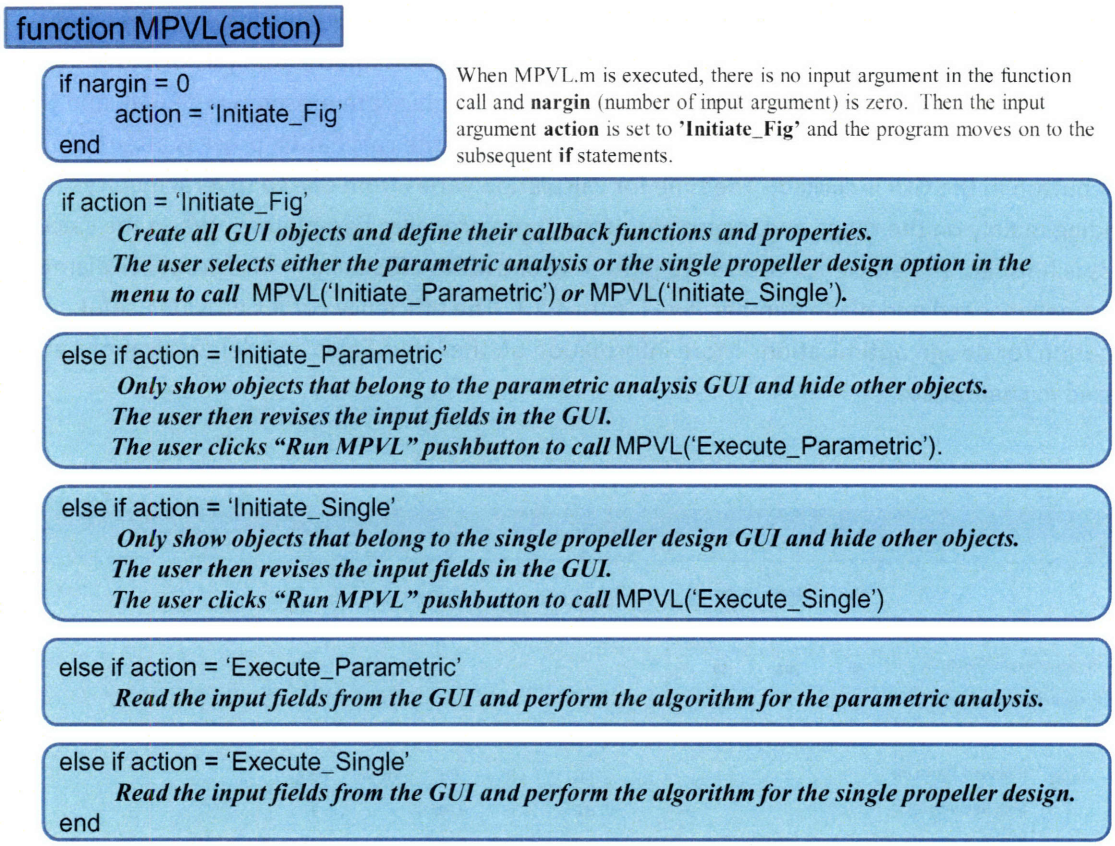


Figure 4. 5 Switchyard Programming Structure in MPVL

For the parametric analysis, only the efficiency curves are plotted after the calculation process is complete. For the single propeller design, the graphical report, the two-dimensional blade profile and the three-dimensional propeller image are plotted. Four text files are automatically created in the single propeller design. Detailed information of the output figures and text files were provided in the following subsections.

When the calculation process is complete, the GUIs do not terminate but wait for further instructions from the users. The users can choose modifying the previous run and executing the program again, or switching to the other GUI (either the parametric analysis GUI or the single propeller design GUI), or printing out the current GUI, or exiting the program.

4.2.2 Parametric Analysis GUI

The three major parameters in the propeller parametric analysis are the number of blades, the propeller speed (RPM) and the propeller diameter. The parametric analysis GUI uses combinations of

these three parameters to generate the efficiency curves to help optimize the design. When the "Parametric Analysis" option in the option menu is selected, the Parametric Analysis GUI (See Figure 4.6.) prompts and waits for actions from the users. Default input values are present in the GUI, and the users can either keep or modify them. MPVL then proceeds to the calculation process after the "Run MPVL" button in the GUI is clicked. The time for calculation varies from one to several minutes depending mainly on the range and increment of the three propeller parameters. When the calculation process is finished, the efficiency curves are plotted to show the efficiency vs. the propeller diameter for each propeller speed and blade number (See Figure 4.7.). The efficiency curves provide useful information for design optimization. More information on the input fields and demonstrations are discussed in chapter six.

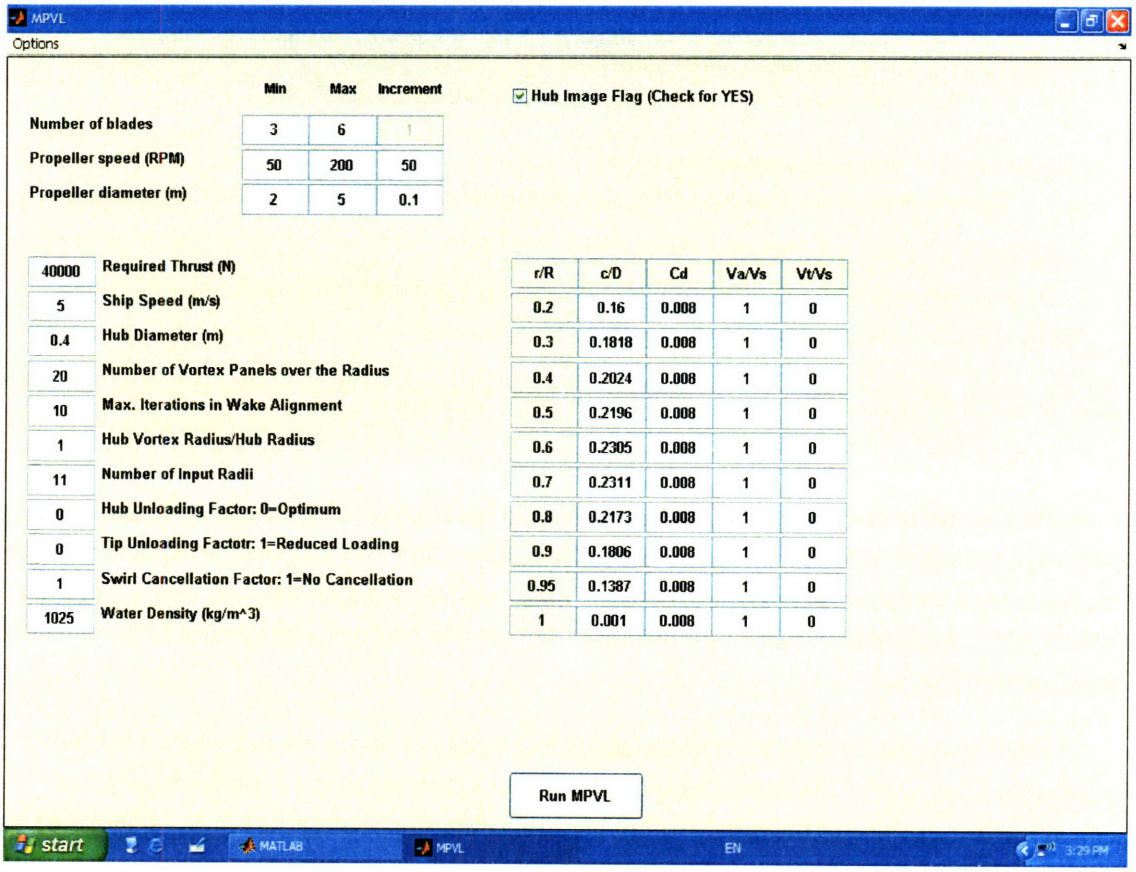


Figure 4. 6 MPVL Parametric Analysis GUI

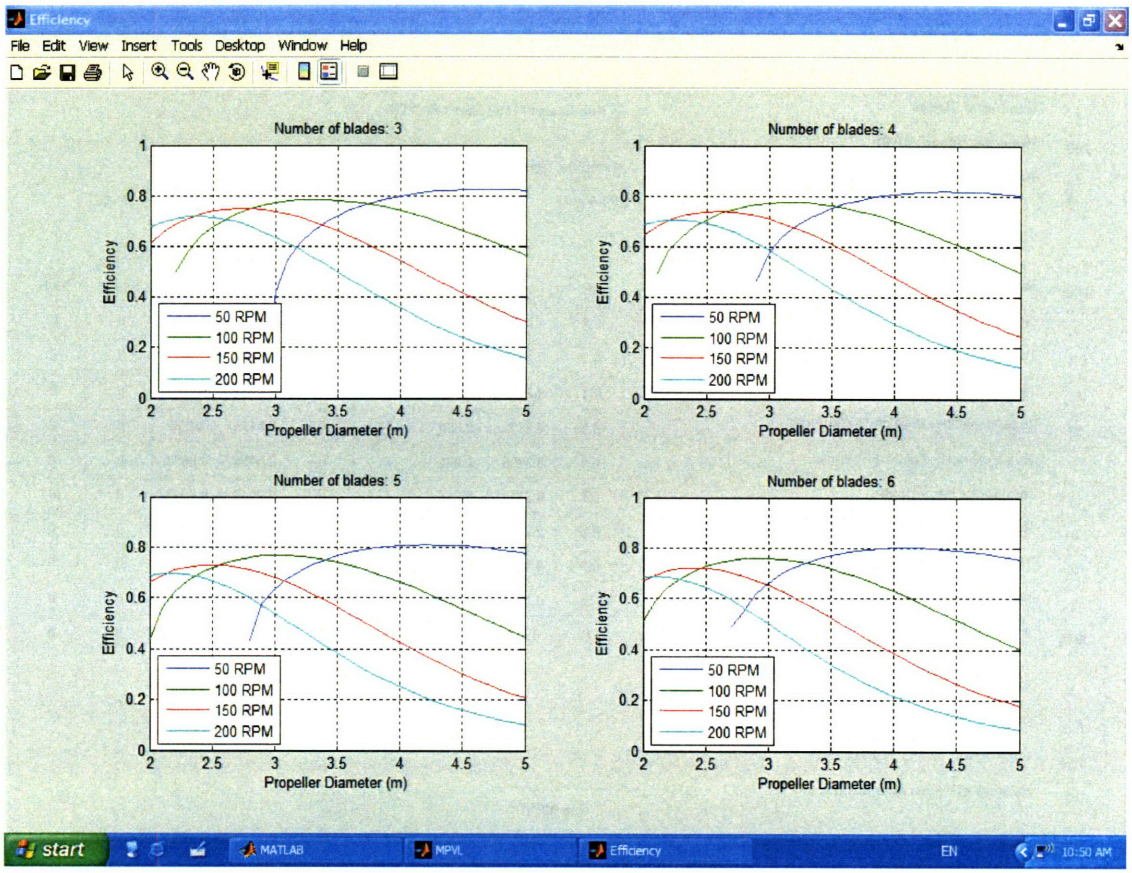


Figure 4. 7 Efficiency Curves in the MPVL Parametric Analysis

4.2.3 Single Propeller Design GUI

While the parametric analysis GUI facilitates the optimization of the propeller parameters, the single propeller design GUI performs a complete propeller design. The layout of the single propeller design GUI is similar to that of the parametric analysis GUI except that there are more input fields. Figure 4.8 shows an example of the single propeller design GUI. More information on the input fields and demonstrations are discussed in chapter six.

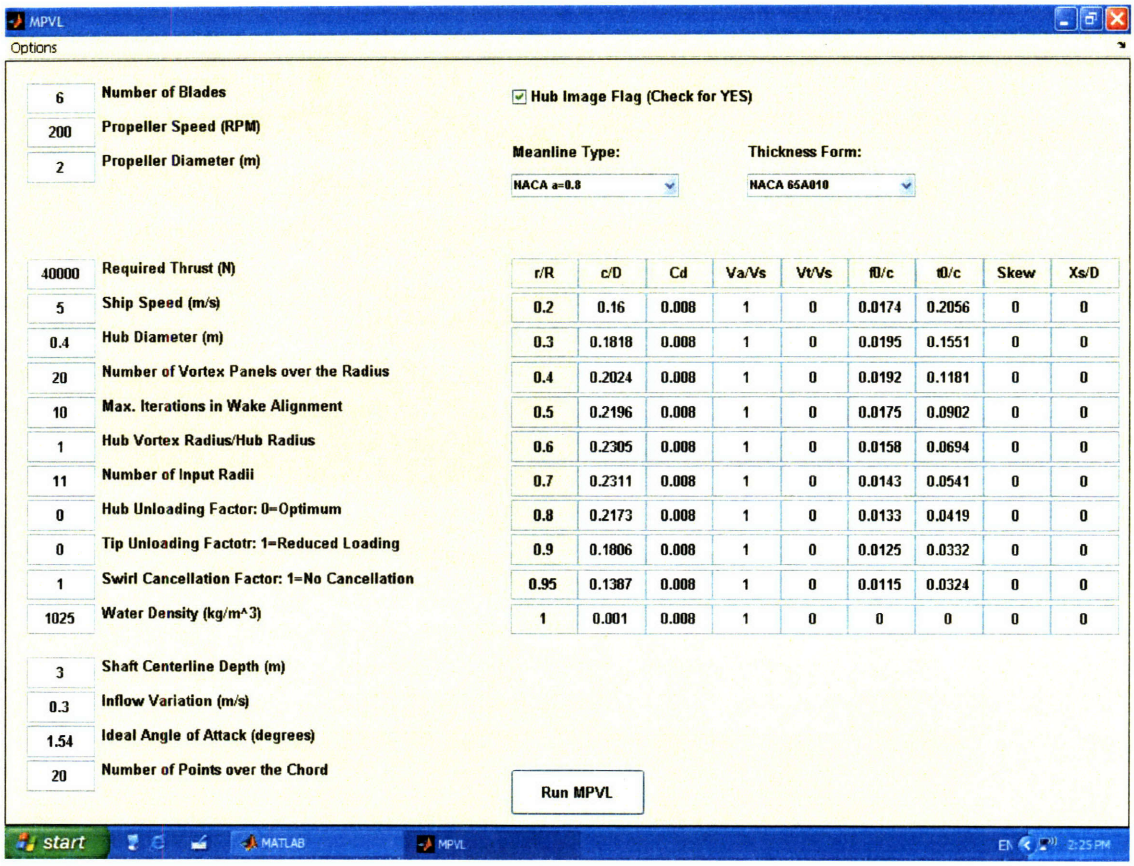


Figure 4. 8 MPVL Single Propeller Design GUI

Default input values are present when the single propeller design GUI is launched. The users can either keep or modify the default input values. The calculation process is initiated by clicking the "Run MPVL" button at the bottom of the GUI. When the calculation process is complete, three figures and four text files are created.

Figure 4.9 is the first figure to be created and contains the plots of the non-dimensional circulation distribution, the inflow and induced velocities, the undisturbed flow angle β and the hydrodynamic pitch angle β_i and the chord distribution. Figure 4.10 is the second figure to be created and contains the two-dimensional propeller blade profile. The last figure created by the single propeller design GUI is Figure 4.11, which contains the three-dimensional propeller image.

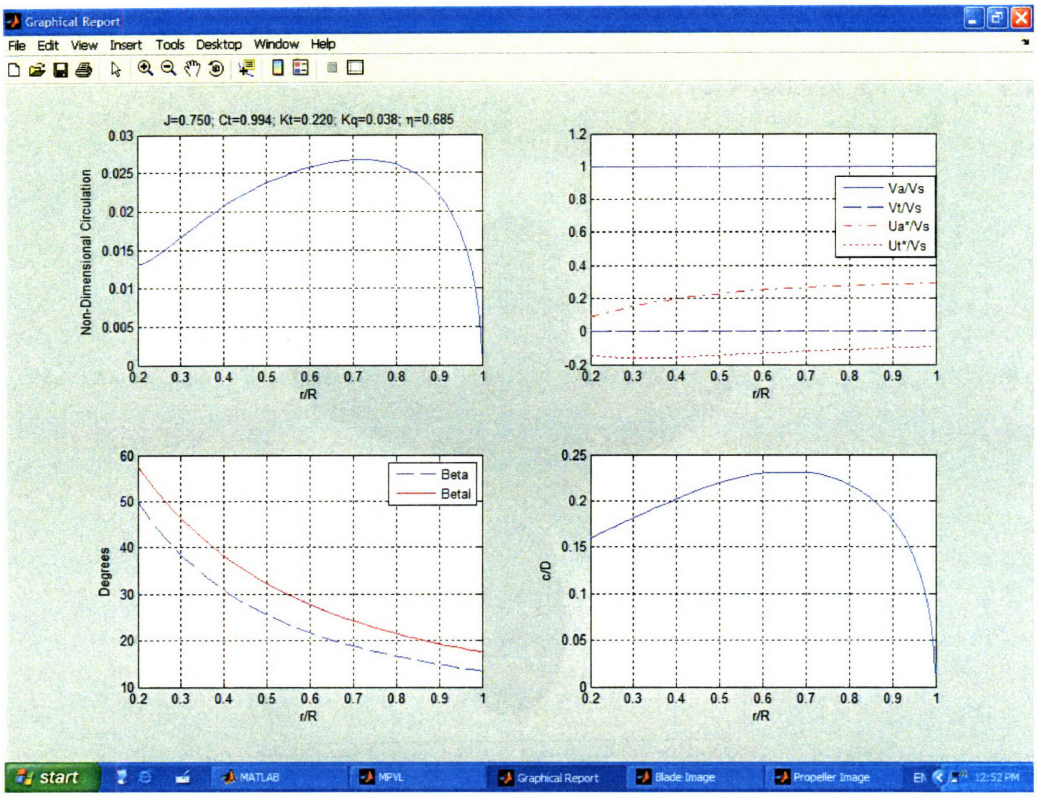


Figure 4. 9 Graphical Report in MPVL Single Propeller Design

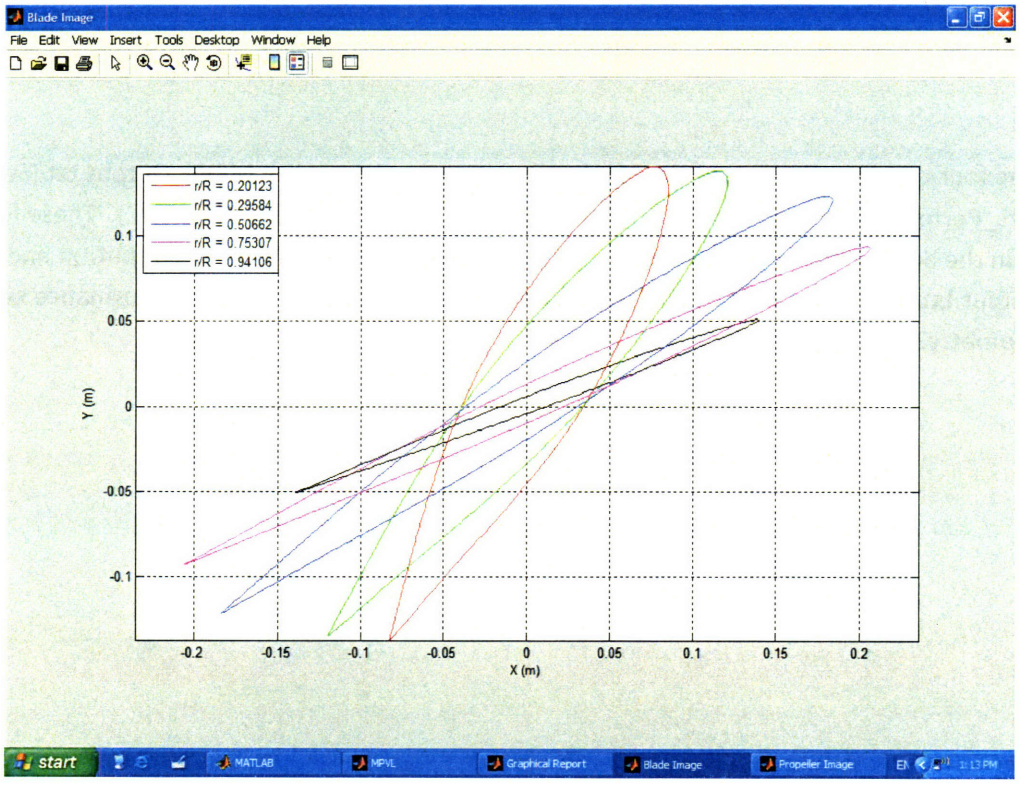


Figure 4. 10 2D Blade Profile in MPVL Single Propeller Design

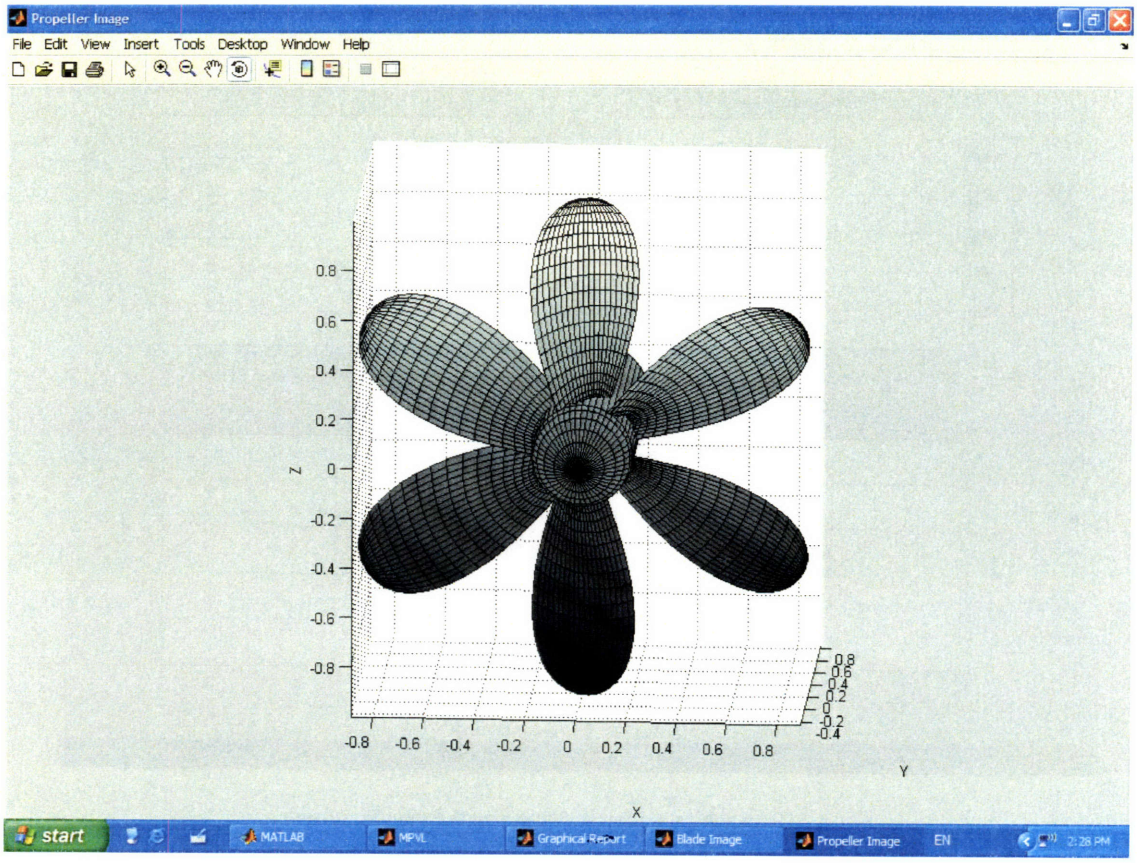


Figure 4. 11 3D Propeller Image in MPVL Single Propeller Design

The four text files created are the **MPVL_input.txt** (See Table 4.1.), **MPVL_Output.txt** (See Table 4.2.), **MPVL_Performance.txt** (See Table 4.3.) and **MPVL_Geometry.txt** (See Table 4.4.). These four text files contain the detailed information of the inputs and outputs for the run. **MPVL_Input.txt** and **MPVL_Output.txt** are consistent with the input and output files of PVL[22]. **MPVL_Performance.txt** and **MPVL_Geometry.txt** are add-ons to MPVL.

[22] (Kerwin 2001, 182)

```
2007-03-06 11:10:03 MPVL_Input.txt
20        Number of Vortex Panels over the Radius
10        Max. Iterations in Wake Alignment
1         Hub Image Flag: 1=YES, 0=NO
1.0       Hub Vortex Radius/Hub Radius
11        Number of Input Radii
6         Number of Blades
0.750     Advance Coef., J, Based on Ship Speed
0.994     Desired Thrust Coef., Ct
0         Hub Unloading Factor: 0=optimum
0         Tip Unloading Factor: 1=Reduced Loading
1         Swirl Cencellation Factor: 1=No Cancellation
r/R         C/D       XCD       Va/Vs   Vt/Vs
0.20000   0.16000   0.00800     1.00    0.0000
0.25000   0.17105   0.00800     1.00    0.0000
0.30000   0.18180   0.00800     1.00    0.0000
0.40000   0.20240   0.00800     1.00    0.0000
0.50000   0.21960   0.00800     1.00    0.0000
0.60000   0.23050   0.00800     1.00    0.0000
0.70000   0.23110   0.00800     1.00    0.0000
0.80000   0.21730   0.00800     1.00    0.0000
0.90000   0.18060   0.00800     1.00    0.0000
0.95000   0.13870   0.00800     1.00    0.0000
1.00000   0.00100   0.00800     1.00    0.0000
```

Table 4. 1 MPVL Single Propeller Design Input Text File

MPVL_Output.txt
MPVL Output Table

```
Ct=  0.9937
Cp=  1.4504
Kt=  0.2195
Kq=  0.0382
Va/Vs=  1.0000
Efficiency= 0.6852
```

r/R	G	Va	Vt	Ua	Ut	Beta	BetaI	c/D	Cd
0.20123	0.013198	1.00000	0.0000	0.09277	-0.14532	49.872	57.446	0.16028	0.00800
0.21105	0.013297	1.00000	0.0000	0.09917	-0.14812	48.522	56.196	0.16247	0.00800
0.23045	0.013790	1.00000	0.0000	0.11160	-0.15265	46.012	53.831	0.16677	0.00800
0.25894	0.014854	1.00000	0.0000	0.12909	-0.15715	42.674	50.598	0.17300	0.00800
0.29584	0.016439	1.00000	0.0000	0.15005	-0.15988	38.903	46.817	0.18093	0.00800
0.34022	0.018349	1.00000	0.0000	0.17240	-0.15974	35.057	42.816	0.19031	0.00800
0.39100	0.020349	1.00000	0.0000	0.19419	-0.15655	31.407	38.876	0.20069	0.00800
0.44693	0.022242	1.00000	0.0000	0.21396	-0.15091	28.110	35.196	0.21111	0.00800
0.50662	0.023897	1.00000	0.0000	0.23098	-0.14372	25.231	31.890	0.22049	0.00800
0.56862	0.025233	1.00000	0.0000	0.24508	-0.13587	22.775	29.003	0.22862	0.00800
0.63138	0.026192	1.00000	0.0000	0.25647	-0.12805	20.712	26.531	0.23080	0.00800
0.69338	0.026706	1.00000	0.0000	0.26552	-0.12071	18.999	24.447	0.23110	0.00800
0.75307	0.026674	1.00000	0.0000	0.27263	-0.11412	17.589	22.713	0.22639	0.00800
0.80900	0.025955	1.00000	0.0000	0.27817	-0.10839	16.441	21.288	0.21535	0.00800
0.85978	0.024387	1.00000	0.0000	0.28244	-0.10355	15.518	20.135	0.19960	0.00800
0.90416	0.021846	1.00000	0.0000	0.28568	-0.09960	14.791	19.220	0.17820	0.00800
0.94106	0.018296	1.00000	0.0000	0.28808	-0.09650	14.235	18.519	0.14840	0.00800
0.96955	0.013818	1.00000	0.0000	0.28977	-0.09421	13.833	18.010	0.11257	0.00800
0.98895	0.008605	1.00000	0.0000	0.29085	-0.09271	13.572	17.679	0.07190	0.00800
0.99877	0.002922	1.00000	0.0000	0.29138	-0.09196	13.443	17.516	0.02600	0.00800

Table 4. 2 MPVL Single Propeller Design Output Text File

Performance.txt

Propeller Performance Table

r/R	V*	beta	betai	Gamma	Cl	Sigma	dBetai
0.201	4.213	49.87	57.45	0.4146	0.614	13.919	2.36
0.211	4.411	48.52	56.20	0.4177	0.583	12.689	2.41
0.230	4.804	46.01	53.83	0.4332	0.541	10.681	2.48
0.259	5.386	42.67	50.60	0.4666	0.501	8.480	2.56
0.296	6.145	38.90	46.82	0.5165	0.465	6.495	2.60
0.340	7.064	35.06	42.82	0.5765	0.429	4.898	2.60
0.391	8.121	31.41	38.88	0.6393	0.392	3.691	2.54
0.447	9.289	28.11	35.20	0.6988	0.356	2.808	2.45
0.507	10.539	25.23	31.89	0.7508	0.323	2.171	2.34
0.569	11.839	22.77	29.00	0.7927	0.293	1.712	2.21
0.631	13.156	20.71	26.53	0.8228	0.271	1.379	2.08
0.693	14.457	19.00	24.45	0.8390	0.251	1.136	1.96
0.753	15.710	17.59	22.71	0.8380	0.236	0.957	1.85
0.809	16.884	16.44	21.29	0.8154	0.224	0.825	1.76
0.860	17.949	15.52	20.13	0.7661	0.214	0.727	1.68
0.904	18.881	14.79	19.22	0.6863	0.204	0.654	1.62
0.941	19.655	14.23	18.52	0.5748	0.197	0.602	1.57
0.970	20.253	13.83	18.01	0.4341	0.190	0.566	1.53
0.989	20.660	13.57	17.68	0.2703	0.182	0.543	1.51
0.999	20.866	13.44	17.52	0.0918	0.169	0.532	1.49

Table 4. 3 Propeller Performance Text File

Geometry.txt

Propeller Geometry Table

Propeller Diameter = 2.0 m
Number of Blades = 6
Propeller Speed= 200 RPM
Propeller Hub Diameter = 0.40 m
Meanline Type: NACA a=0.8
Thickness Type: NACA 65A010

r/R	P/D	Skew	Xs/D	c/D	f0/c	t0/c
0.201	1.05	0.0	0.000	0.160	0.0107	0.2049
0.211	1.05	0.0	0.000	0.162	0.0104	0.1993
0.230	1.05	0.0	0.000	0.167	0.0099	0.1887
0.259	1.05	0.0	0.000	0.173	0.0095	0.1741
0.296	1.05	0.0	0.000	0.181	0.0091	0.1569
0.340	1.05	0.0	0.000	0.190	0.0083	0.1389
0.391	1.05	0.0	0.000	0.201	0.0075	0.1210
0.447	1.05	0.0	0.000	0.211	0.0066	0.1040
0.507	1.05	0.0	0.000	0.220	0.0056	0.0886
0.569	1.05	0.0	0.000	0.229	0.0048	0.0753
0.631	1.06	0.0	0.000	0.231	0.0041	0.0641
0.693	1.06	0.0	0.000	0.231	0.0036	0.0550
0.753	1.07	0.0	0.000	0.226	0.0032	0.0472
0.809	1.07	0.0	0.000	0.215	0.0030	0.0410
0.860	1.07	0.0	0.000	0.200	0.0028	0.0357
0.904	1.08	0.0	0.000	0.178	0.0025	0.0331
0.941	1.08	0.0	0.000	0.148	0.0023	0.0326
0.970	1.08	0.0	0.000	0.113	0.0017	0.0257
0.989	1.08	0.0	0.000	0.072	0.0007	0.0105
0.999	1.08	0.0	0.000	0.026	0.0001	0.0012

Table 4. 4 Propeller Geometry Text File

The **MPVL_Performance.txt** file contains a propeller performance table which lists the total inflow velocity (V*), the undisturbed flow angle β, the hydrodynamic pitch angle β_i, the vortex sheet strength (Gamma), the lift coefficient (C_L), the cavitation number (sigma), the pitch angle variation $\delta\beta_i$, maximum camber ratio and maximum thickness ratio over the radius. This information can be utilized for cavitation consideration. Details of these output fields were introduced in chapter six.

Finally, the **MPVL_Geometry.txt** file contains a summary of the propeller geometry. It includes the propeller diameter, number of blades, propeller speed, hub diameter, meanline type, thickness form, pitch, skew, rake, chord distribution, camber distribution and thickness distribution. This propeller geometry data is essential for producing the propeller images and also for manufacturing.

5 MPVL Validation

5.1 Input Variants and Assumptions

In this chapter, MPVL was validated by comparing its outputs with PVL by using identical inputs. Study cases were defined, and the results were compared. Several assumptions were made to simplify the cases. First, the propeller was assumed to operate in the open water, and thereby the advance velocity was equal to the ship velocity ($V_a = V_s$). Second, the inflow was assumed to be uniform. Therefore, the axial inflow velocity ratio (V_a / V_s) was one, and the tangential inflow velocity ratio (V_t / V_s) was zero everywhere along the radius of the propeller. Moreover, several coefficients were required to define the validation cases. The advance coefficient (J), the thrust loading coefficient (C_T) and the thrust coefficient (K_T) were defined in Equation 5.1, 5.2 and 5.3 respectively. Alternatively, the thrust loading coefficient can be rewritten in terms of the advance coefficient and the thrust coefficient as shown in Equation 5.4[23]. Thus MPVL was validated by varying the advance coefficient (J) and the thrust loading coefficient (C_T). The ship velocity and the required thrust were varied to derive different advance coefficients and thrust loading coefficients for each validation case.

Advance Coefficient: $$J = \frac{V_S}{n\,D} \qquad (5.1)$$

Thrust Loading Coefficient: $$C_T = \frac{T}{\frac{1}{2}\rho\,{V_S}^2\frac{\pi}{4}D^2} \qquad (5.2)$$

Thrust Coefficient: $$K_T = \frac{T}{\rho\,n^2\,D^4} \qquad (5.3)$$

Equation (5.2) can be rewritten as $$C_T = \frac{8\,K_T}{\pi\,J^2} \qquad (5.4)$$

Vs is the ship velocity, and *n* is the propeller speed in revolution per second. *D* is the propeller diameter. *T* is the thrust, and ρ is the water density.

The advance coefficient was held constant, and the thrust loading coefficient was varied and vice versa to test MPVL under different loading conditions. Table 5.1 shows the summary of the five

[23] (Woud and Stapersma 2003, 397)

validation cases. In the first three validation cases, the advance coefficient (J) was held constant. By varying the thrust loading coefficient (C_T), three loading conditions were defined: moderately, heavily and lightly loaded. In case four and five, the thrust loading coefficient (C_T) was maintained moderate, and the advance coefficient (J) was varied to examine the results under extremely high and low advance coefficients. Throughout this chapter, a six-bladed, 120 RPM and three-meter diameter propeller was used for validation. The geometry of the N4148[24] propeller was adopted in each case. The selected meanline type was NACA a=0.8, and the thickness form was NACA 65A010.

Case	1	2	3	4	5
Condition	Moderately Loaded	Heavily Loaded	Lightly Loaded	High Advance Coefficient	Low Advance Coefficient
J	0.75	0.75	0.75	2.00	0.083
C_T	0.613	2.726	0.082	0.613	0.662
K_T	0.1354	0.6022	0.0181	0.9629	0.0018
K_Q	0.0228	0.1414	0.0054	0.6591	0.0027
η	0.7094	0.5082	0.3983	0.4650	0.0087

Table 5. 1 Summary of Validation Cases

5.2 Case One – Moderately Loaded

The first validation case was the moderately loaded propeller. All the input fields for this specific case are shown in Figure 5.1. The resultant advance coefficient (J) was 0.75, the thrust loading coefficient (C_T) was 0.613, the thrust coefficient (K_T) was 0.1354, and the torque coefficient (K_Q) was 0.0228. The same inputs were also fed to the MIT Propeller Lifting Line Program (PLL)[25] for double-check, and the outputs were compared with PVL and MPVL. The hub unloading factor was set to 0.08 in PVL and MPVL while the hub was not unloaded in PLL. Figure 5.2 shows the non-dimensional circulation vs. the radial position. Figure 5.3 shows the axial and tangential induced velocities vs. the radial position. Finally, Figure 5.4 shows the undisturbed flow angle β and the hydrodynamic pitch angle β_i vs. the radial position. It was shown that the circulation curves from MPVL and PVL match perfectly. On the other hand, PLL uses a slightly different strategy on circulation optimization and therefore gave slightly different results.

[24] (Lee, Gu and Kinnas 2004)
[25] (Coney, Hsin and Kerwin 1986)

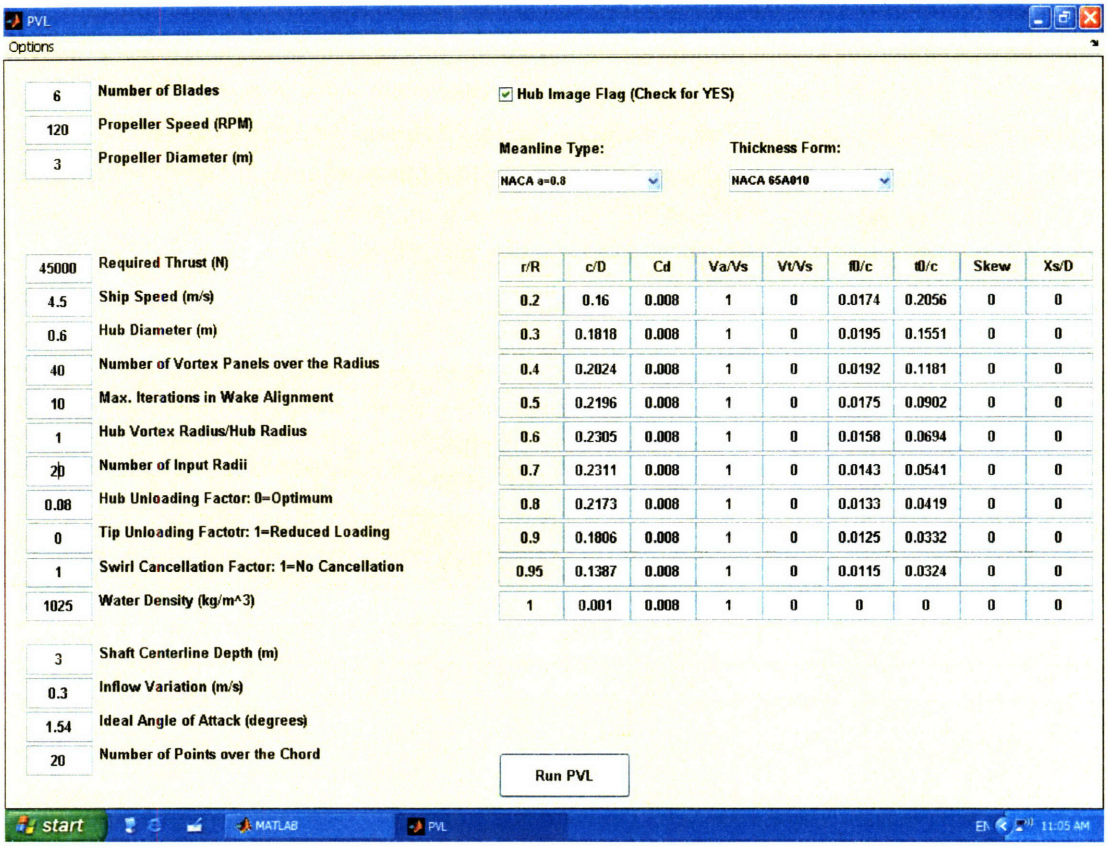


Figure 5. 1 Inputs for Moderately Loaded Case

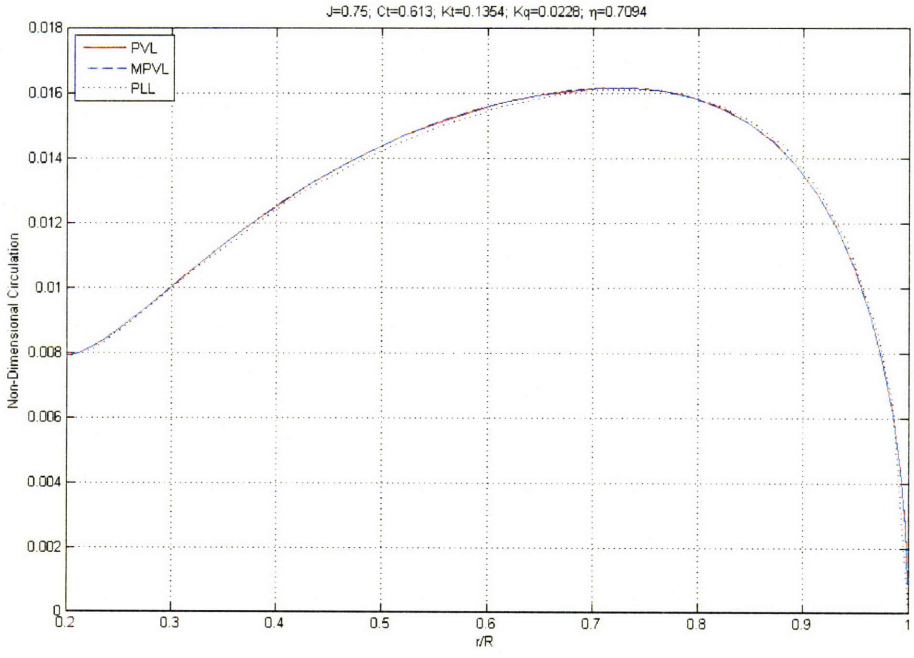


Figure 5. 2 Non-Dimensional Circulation Distribution

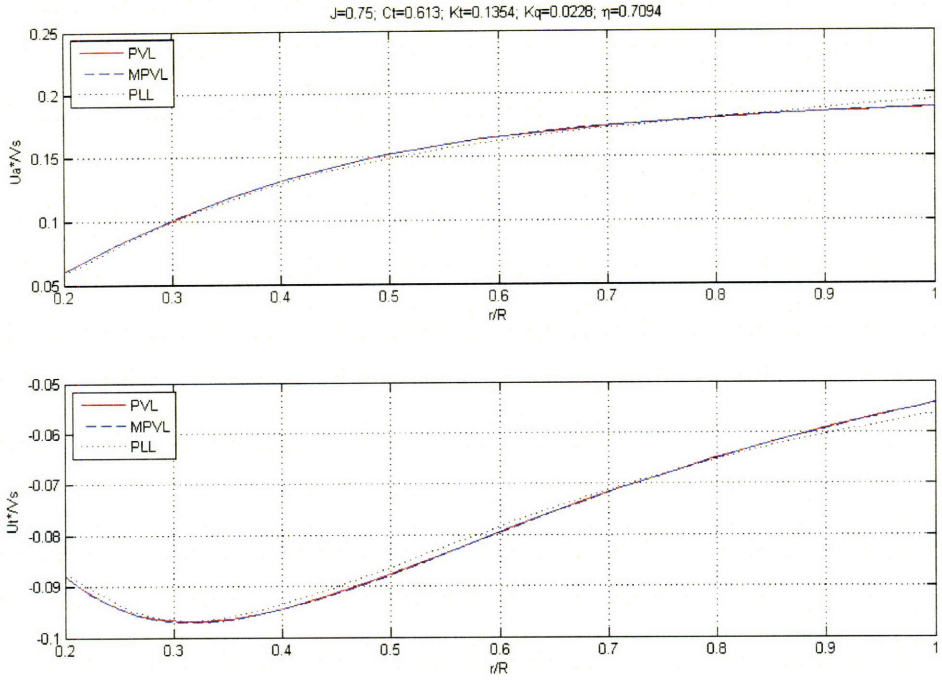


Figure 5. 3 Induced Velocities

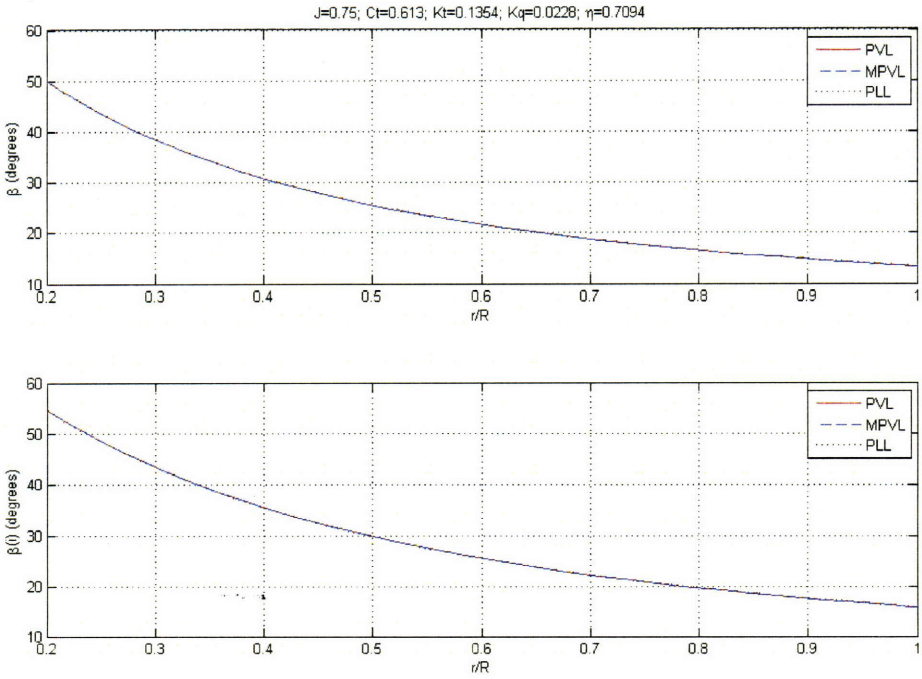
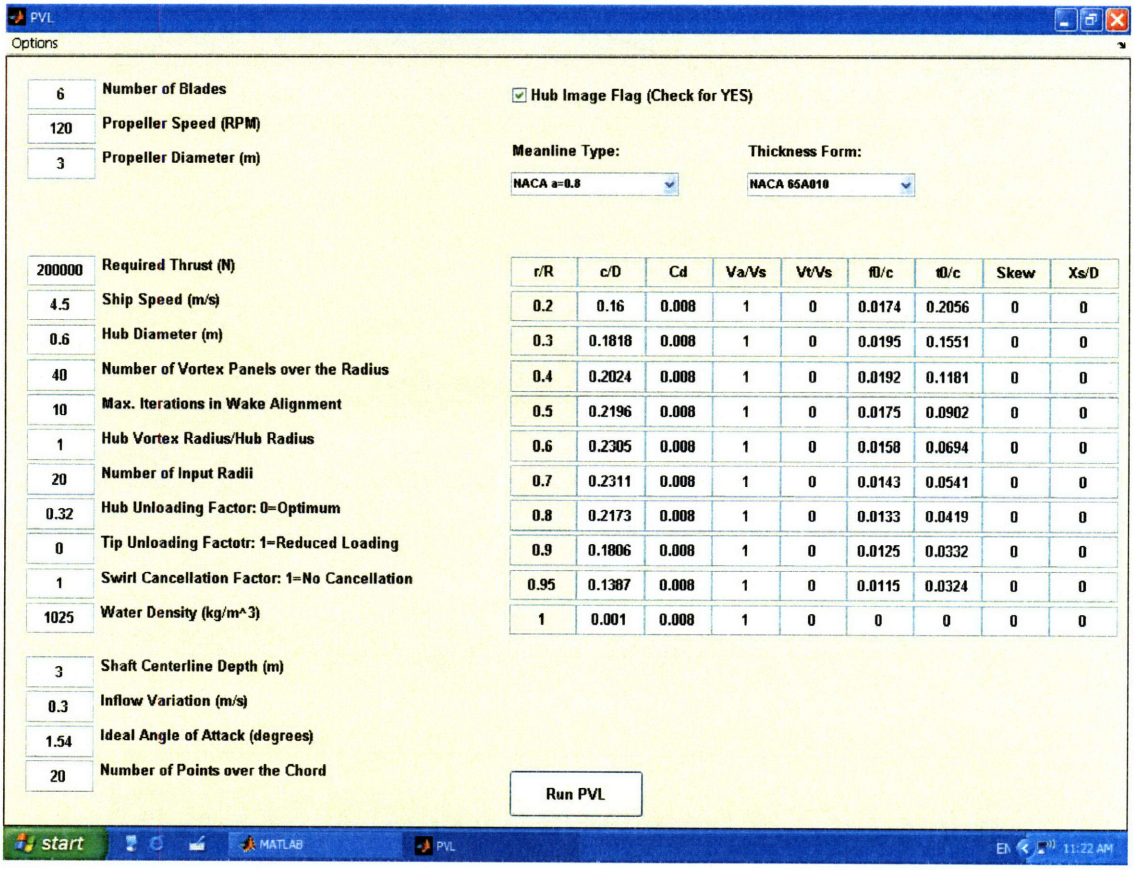


Figure 5. 4 Undisturbed Flow Angle β and Hydrodynamic Pitch Angle β_i

5.3 Case Two – Heavily Loaded

A heavily loaded propeller was presented in this section. All the input fields for this case were shown in Figure 5.5. The resultant advance coefficient (J) was 0.75, the thrust loading coefficient (C_T) was 2.726, the thrust coefficient (K_T) was 0.6022, and the torque coefficient (K_Q) was 0.1414. Only the results from MPVL and PVL were compared. Figure 5.6 shows the non-dimensional circulation vs. the radial position. Figure 5.7 shows the axial and tangential induced velocities vs. the radial position. Figure 5.8 shows the undisturbed flow angle β and the hydrodynamic pitch angle β_i vs. the radial position. It was shown that for the heavily loaded propeller the results from MPVL match the ones from PVL perfectly.



r/R	c/D	Cd	Va/Vs	Vt/Vs	f0/c	f0/c	Skew	Xs/D
0.2	0.16	0.008	1	0	0.0174	0.2056	0	0
0.3	0.1818	0.008	1	0	0.0195	0.1551	0	0
0.4	0.2024	0.008	1	0	0.0192	0.1181	0	0
0.5	0.2196	0.008	1	0	0.0175	0.0902	0	0
0.6	0.2305	0.008	1	0	0.0158	0.0694	0	0
0.7	0.2311	0.008	1	0	0.0143	0.0541	0	0
0.8	0.2173	0.008	1	0	0.0133	0.0419	0	0
0.9	0.1806	0.008	1	0	0.0125	0.0332	0	0
0.95	0.1387	0.008	1	0	0.0115	0.0324	0	0
1	0.001	0.008	1	0	0	0	0	0

Figure 5. 5 Inputs for Heavily Loaded Case

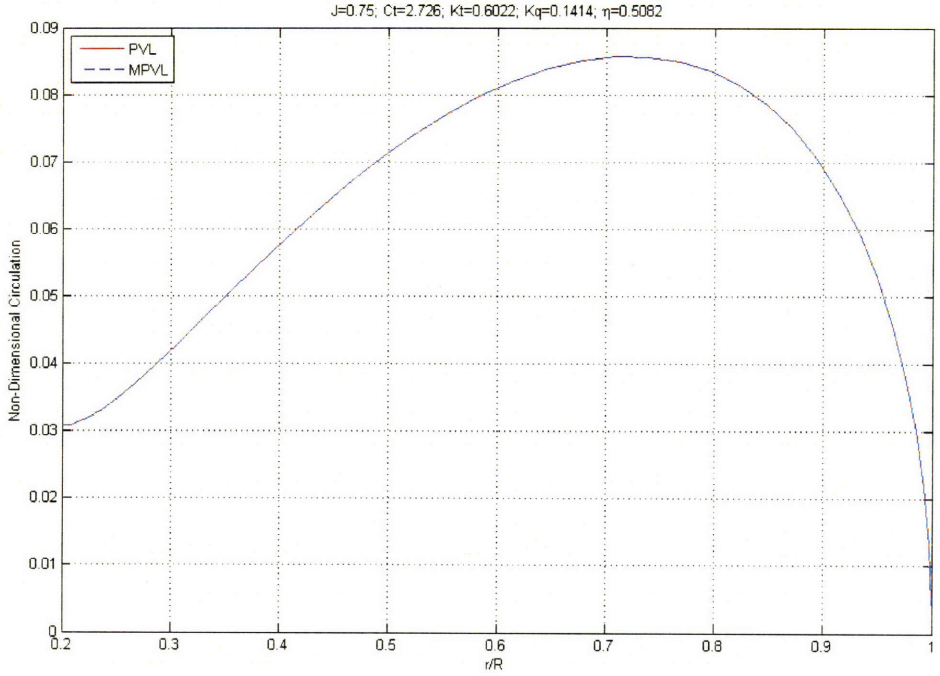


Figure 5. 6 Non-Dimensional Circulation Dsitribution

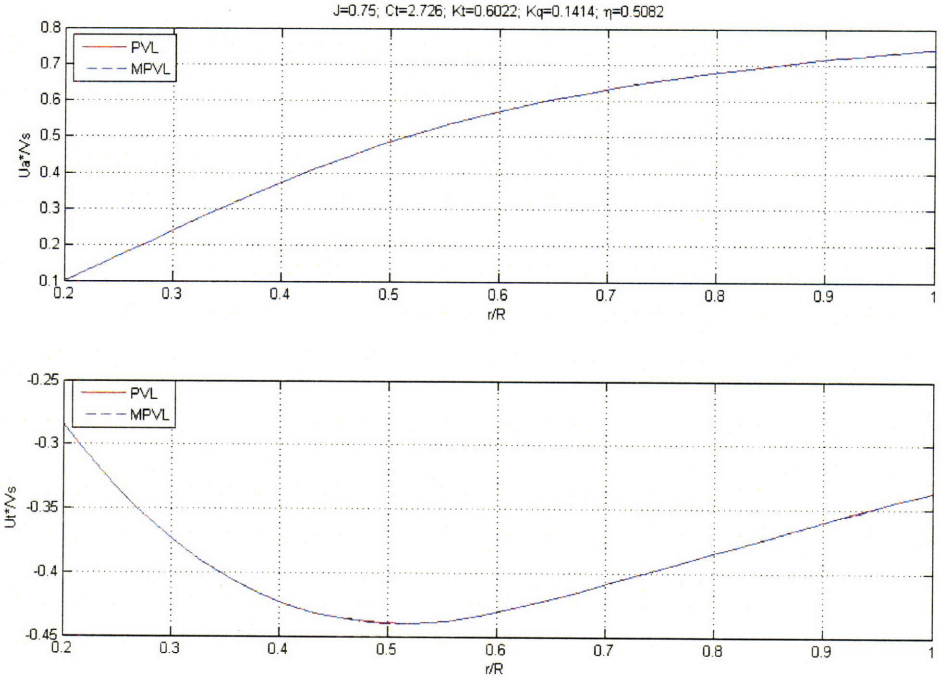


Figure 5. 7 Induced Velocities

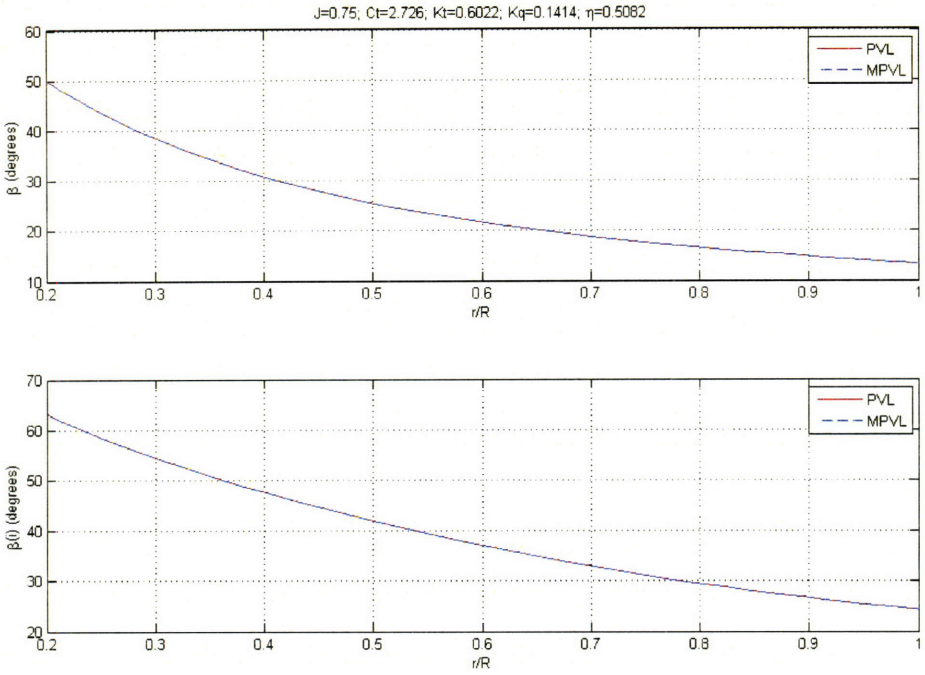


Figure 5. 8 Undisturbed Flow Angle β and Hydrodynamic Pitch Angle β_i

5.4 Case Three – Lightly Loaded

A lightly loaded propeller was presented in this section. All the input fields were shown in Figure 5.9. The resultant advance coefficient (J) was 0.75, the thrust loading coefficient (C_T) was 0.082, the thrust coefficient (K_T) was 0.0181, and the torque coefficient (K_Q) was 0.0054. Figure 5.10 shows the non-dimensional circulation vs. the radial position. Figure 5.11 shows the axial and tangential induced velocities vs. the radial position. Figure 5.12 shows the undisturbed flow angle β and the hydrodynamic pitch angle β_i vs. the radial position. The error in the circulation distribution can hardly be observed. Therefore, it was concluded that MPVL was still consistent with PVL under the lightly loaded condition.

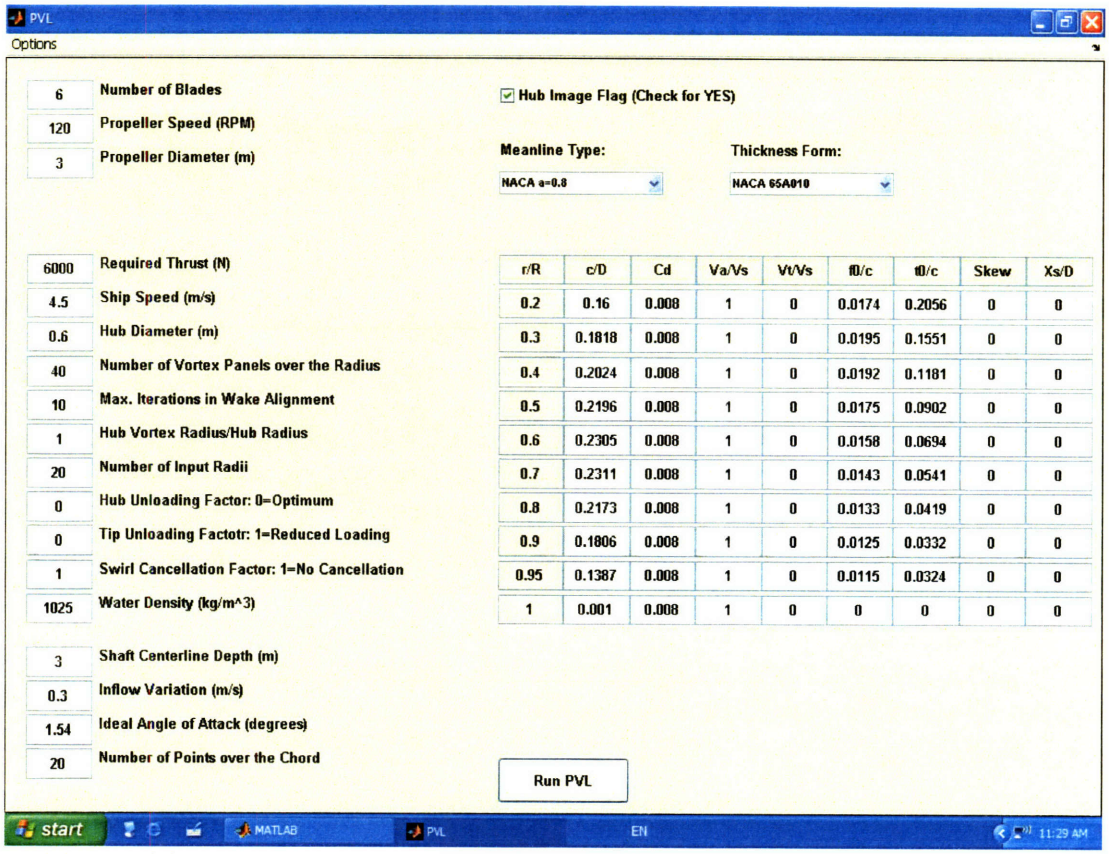


Figure 5. 9 Inputs for Lightly Loaded Case

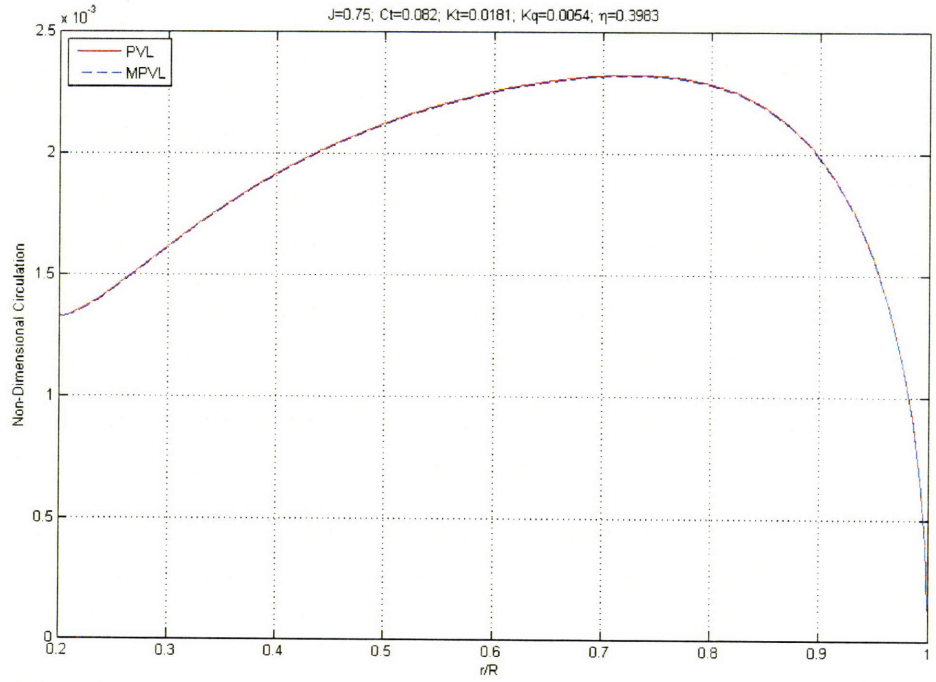


Figure 5. 10 Non-Dimensional Circulation Distribution

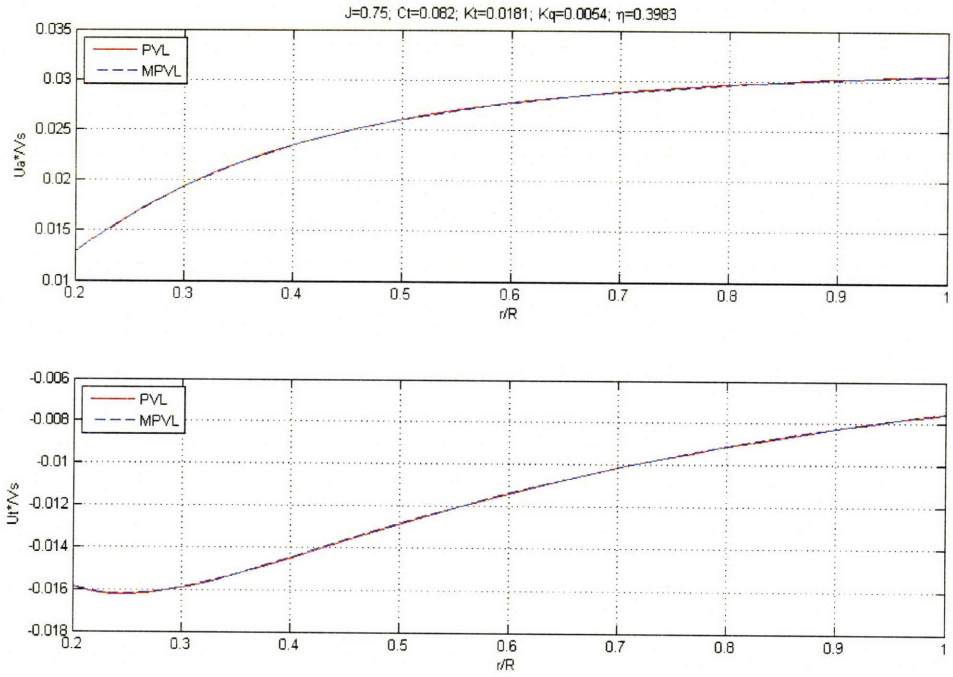


Figure 5. 11 Induced Velocities

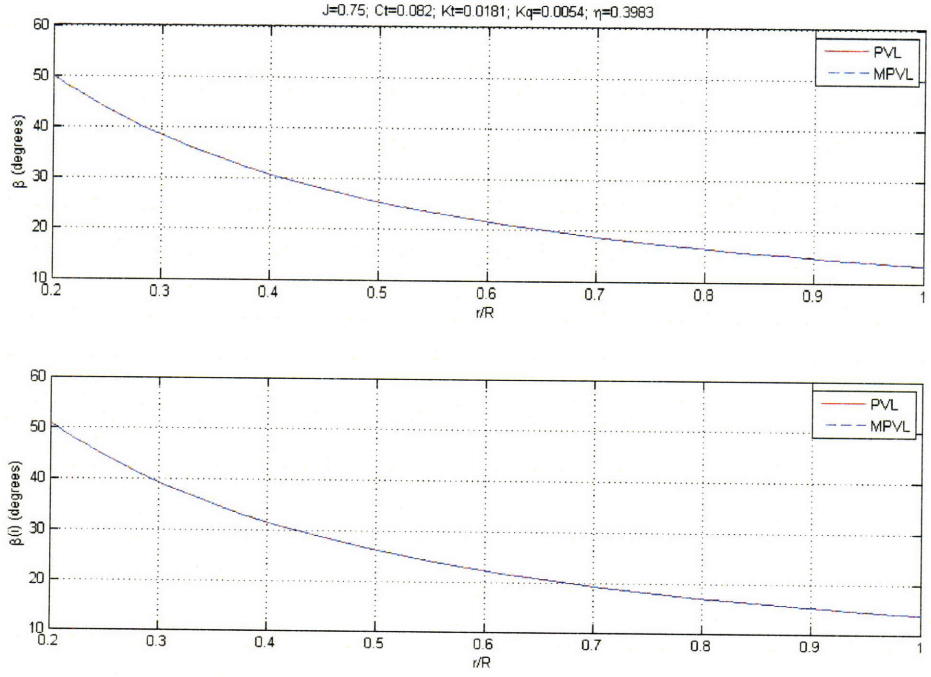


Figure 5. 12 Undisturbed Flow Angle β and Hydrodynamic Pitch Angle β_i

5.5 Case Four – High Advance Coefficient

A high advance coefficient (J) was presented in this validation case. The thrust loading coefficient was maintained moderate by increasing the required thrust. All the input fields were shown in Figure 5.13. The resultant advance coefficient (J) was 2, the thrust loading coefficient (C_T) was 0.613, the thrust coefficient (K_T) was 0.9629, and the torque coefficient (K_Q) was 0.6591. Figure 5.14 shows the non-dimensional circulation vs. the radial position. Figure 5.15 shows the axial and tangential induced velocities vs. the radial position. Figure 5.16 shows the undisturbed flow angle β and the hydrodynamic pitch angle β_i vs. the radial position. It was shown that the circulation distribution from PVL and MPVL matched perfectly, and thus it was concluded that the MPVL was consistent with PVL in the presence of a high advance coefficient.

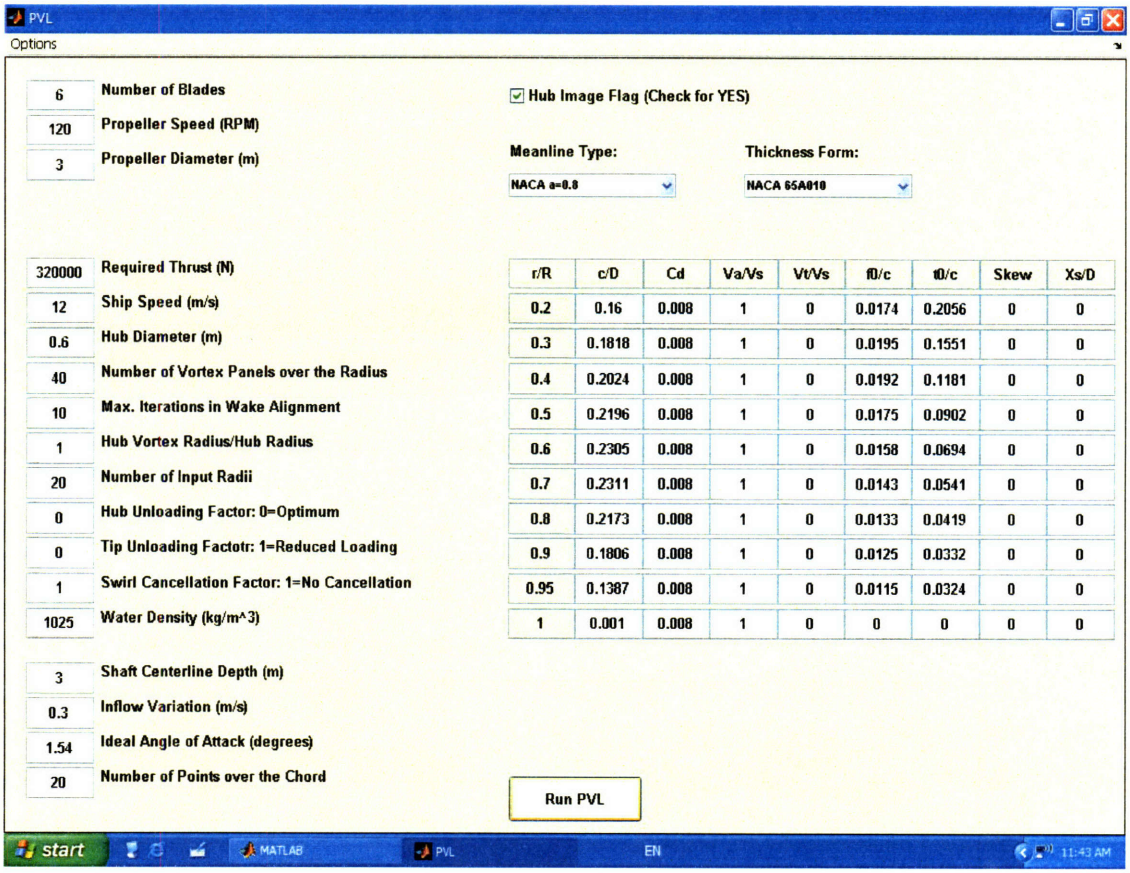


Figure 5. 13 Inputs for High Advance Coefficient Case

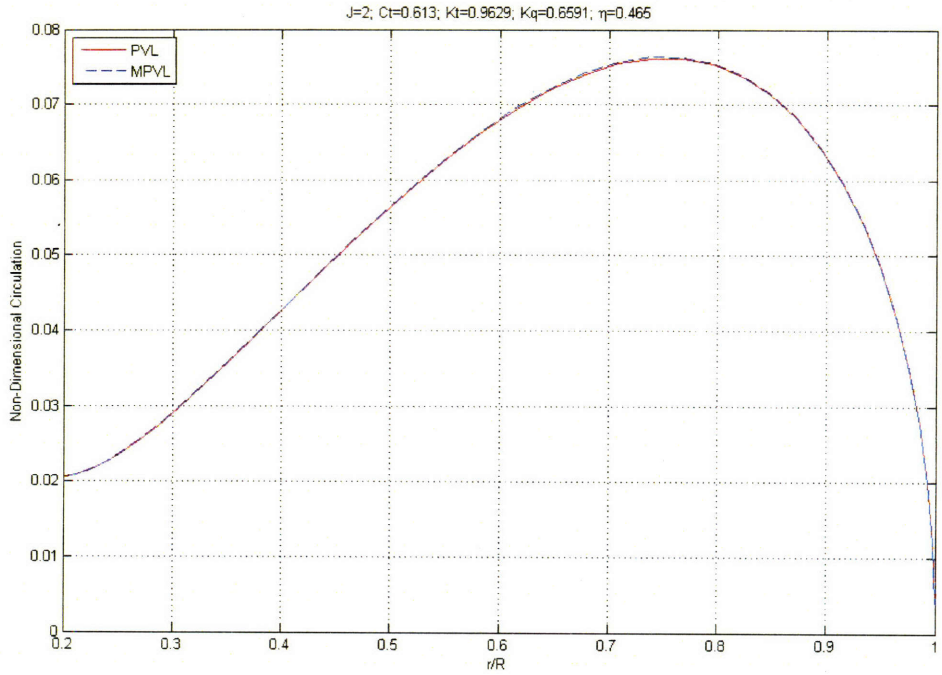


Figure 5. 14 Non-Dimensional Circulation Distribution

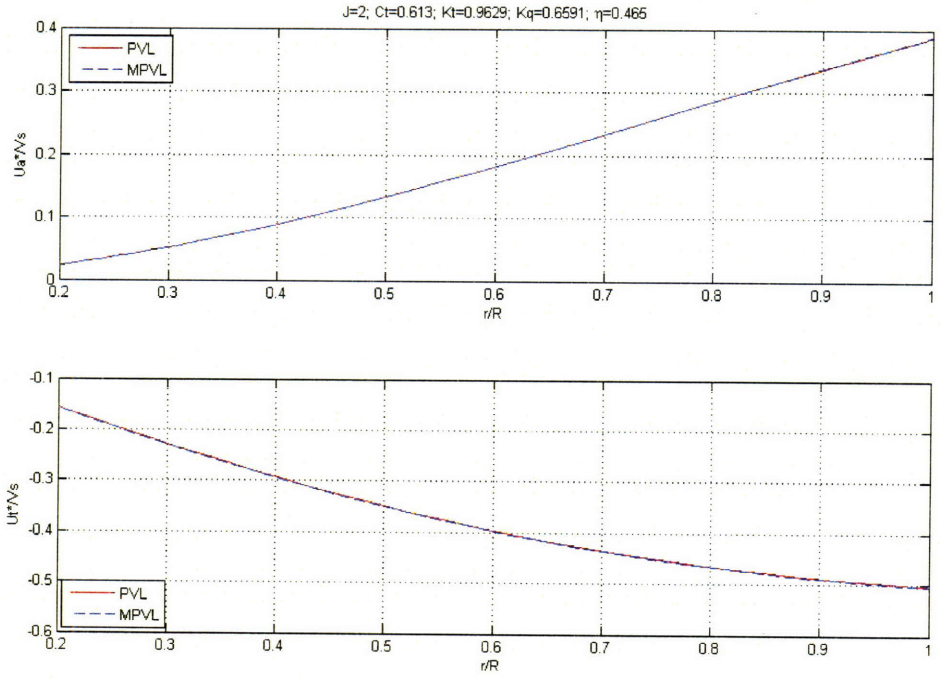


Figure 5. 15 Induced Velocities

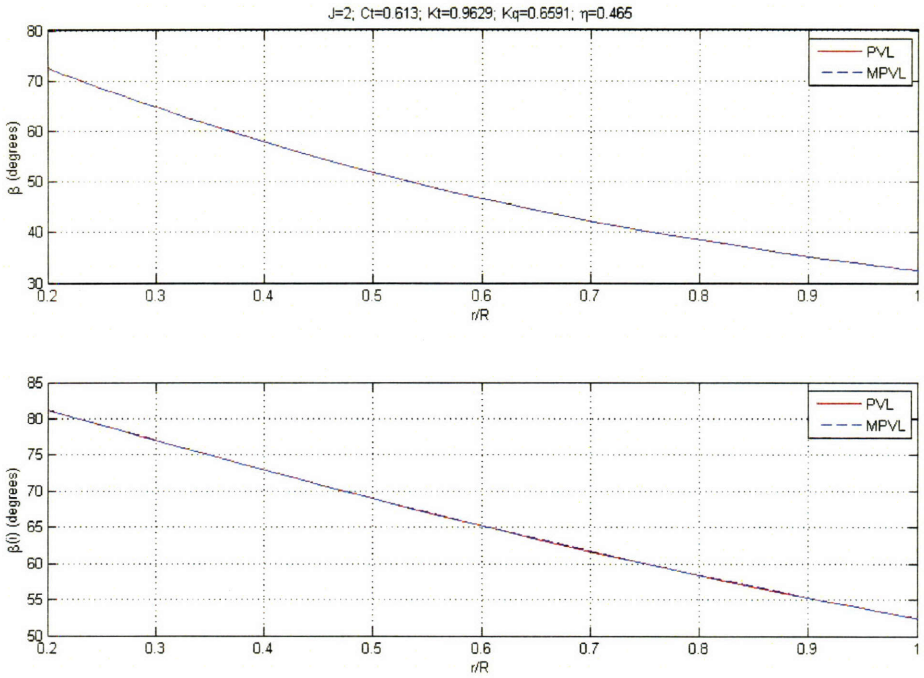
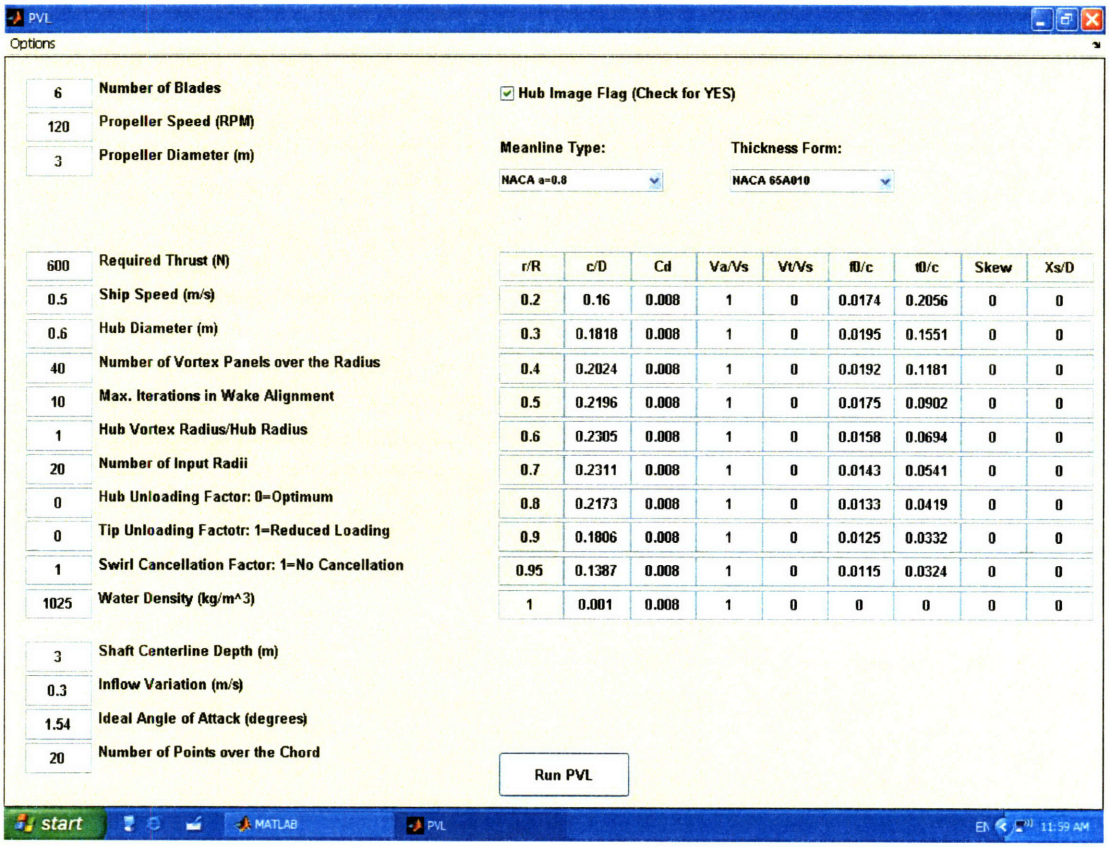


Figure 5. 16 Undisturbed Flow Angle β and Hydrodynamic Pitch Angle β_i

5.6 Case Five – Low Advance Coefficient

This validation case presented a low advance coefficient (J) with a moderate thrust loading coefficient (C_T). All the input fields were shown in Figure 5.17. The resultant advance coefficient (J) was 0.083, the thrust loading coefficient (C_T) was 0.662, the thrust coefficient (K_T) was 0.0018, and the torque coefficient (K_Q) was 0.0027. Figure 5.18 shows the non-dimensional circulation vs. the radial position. Figure 5.19 shows the axial and tangential induced velocities vs. the radial position. Figure 5.20 shows the undisturbed flow angle β and hydrodynamic pitch angle β_i vs. the radial position. The error in the circulation distribution can be observed simply because the value of the circulation was so small that any error could be amplified on this scale. Therefore, it was concluded that MPVL was still consistent with PVL in the presence of a low advance coefficient.



r/R	c/D	Cd	Va/Vs	Vt/Vs	f0/c	f0/c	Skew	Xs/D
0.2	0.16	0.008	1	0	0.0174	0.2056	0	0
0.3	0.1818	0.008	1	0	0.0195	0.1551	0	0
0.4	0.2024	0.008	1	0	0.0192	0.1181	0	0
0.5	0.2196	0.008	1	0	0.0175	0.0902	0	0
0.6	0.2305	0.008	1	0	0.0158	0.0694	0	0
0.7	0.2311	0.008	1	0	0.0143	0.0541	0	0
0.8	0.2173	0.008	1	0	0.0133	0.0419	0	0
0.9	0.1806	0.008	1	0	0.0125	0.0332	0	0
0.95	0.1387	0.008	1	0	0.0115	0.0324	0	0
1	0.001	0.008	1	0	0	0	0	0

Figure 5. 17 Inputs for Low Advance Coefficient Case

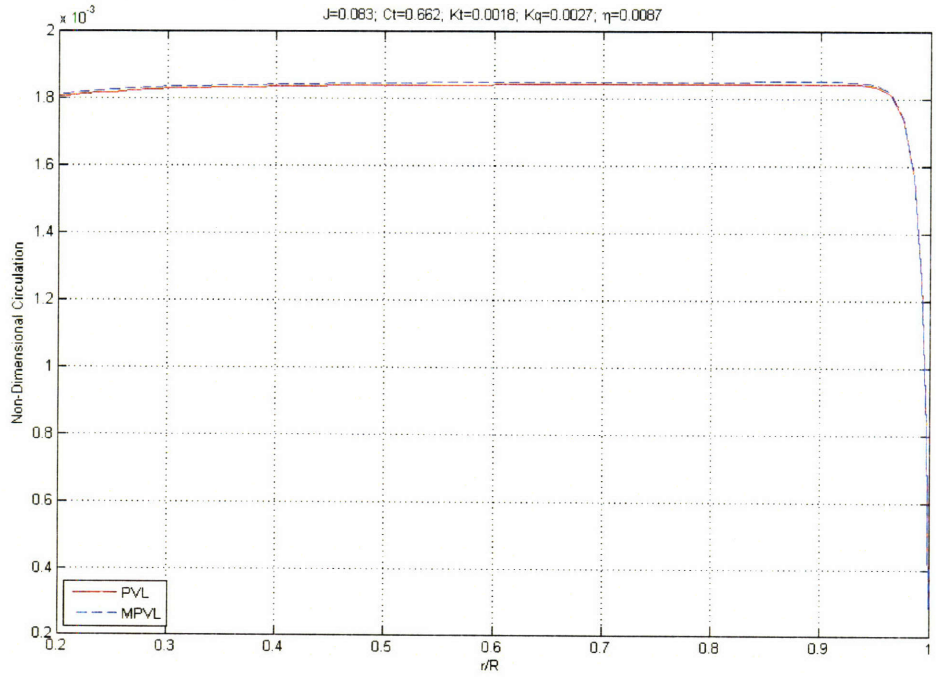


Figure 5. 18 Non-Dimensional Circulation Distribution

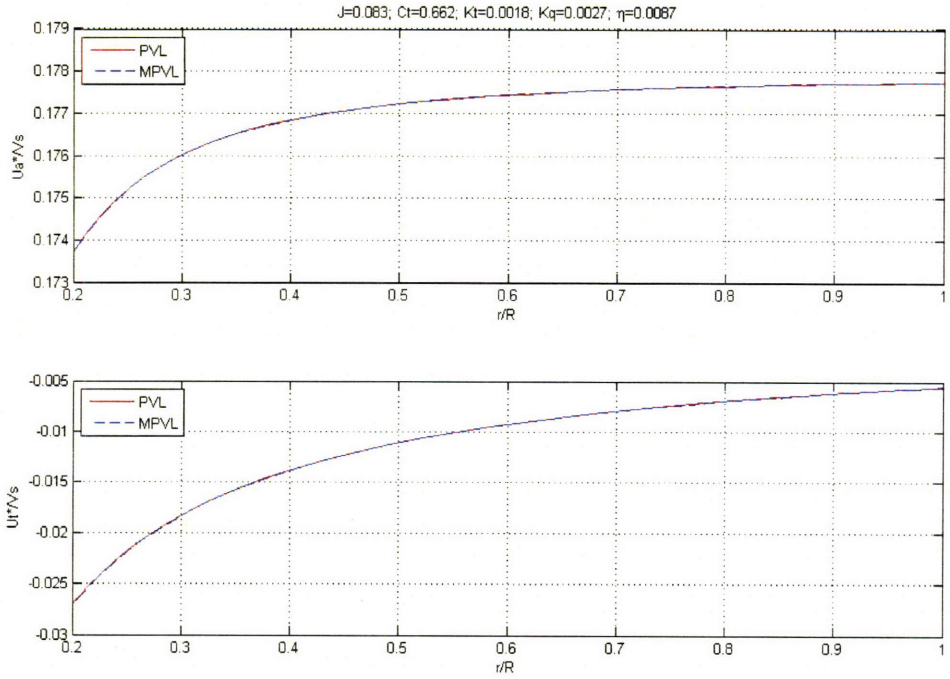


Figure 5. 19 Induced Velocities

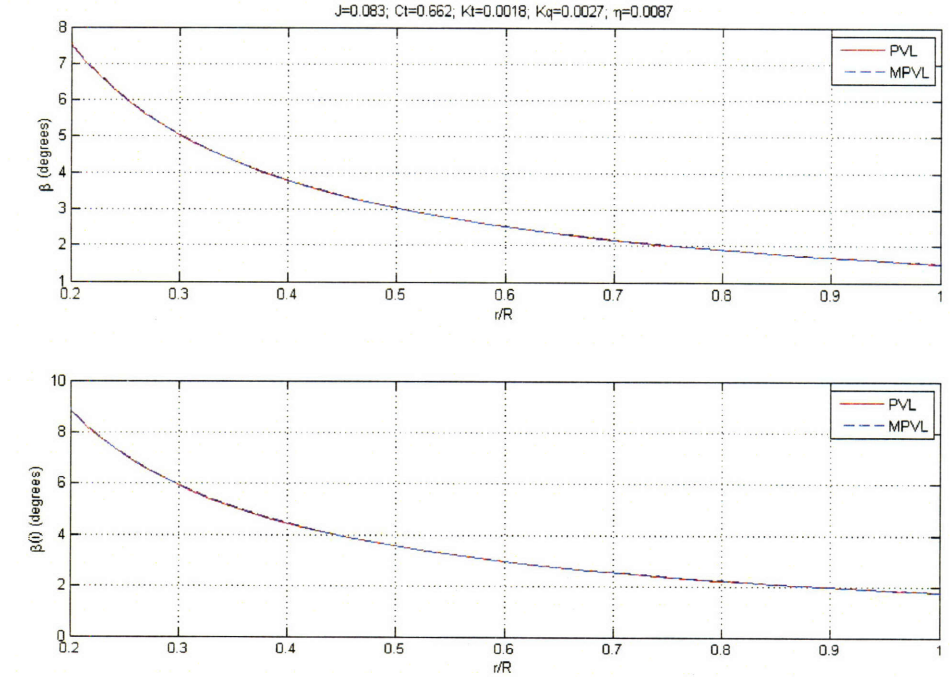


Figure 5. 20 Undisturbed Flow Angle β and Hydrodynamic Pitch Angle β_i

6 Design Example and Demonstration

In this chapter, a design example was introduced to demonstrate the applications of MPVL. This design example also serves as user's manual and shows the users how to start with the parametric analysis GUI to optimize the propeller parameters. With the optimized propeller parameters, the users can then produce a complete propeller design by using the single propeller design GUI.

6.1 Parametric Analysis and Optimization

A propeller design consists of three fundamental global parameters: the number of blades, the propeller speed and the propeller diameter. Different combinations of those three parameters result in different propeller efficiencies. Therefore, it is necessary to perform a parametric study to determine the optimum propeller parameters. The parametric analysis GUI is programmed to calculate the propeller efficiency of each combination of the three propeller parameters and to graphically present the efficiency curves.

Before conducting the parametric analysis in MPVL, input fields in the GUI must be introduced first. In the following paragraphs, the input fields were introduced from the left to the right and from the top to the bottom of the GUI. Also see Figure 6.1 for reference.

Number of Blades: The minimum and maximum numbers of blades are two and six respectively. Propellers normally have number of blades within this range. Users are able to modify this constraint in the **MPVL.m** file.

Propeller Speed: The unit of this input filed is in RPM. The restrictions are that the value must always be positive, the maximum value must be greater than or equal to the minimum value, and the increment should never be negative.

Propeller Diameter: The unit of this input field is in meter. The restrictions are that the value must always be positive, the maximum value must be greater than or equal to the minimum value, and the increment should never be negative. Also, this field should always be greater than the hub diameter.

Required Thrust: The required thrust is related to the calculation of the thrust loading coefficient (C_T) and the thrust coefficient (K_T). The unit of this input field is in Newton.

Ship Speed: This input field is related to the calculation of the advance coefficient (J) and the thrust loading coefficient (C_T). The unit of this field is in meter per second.

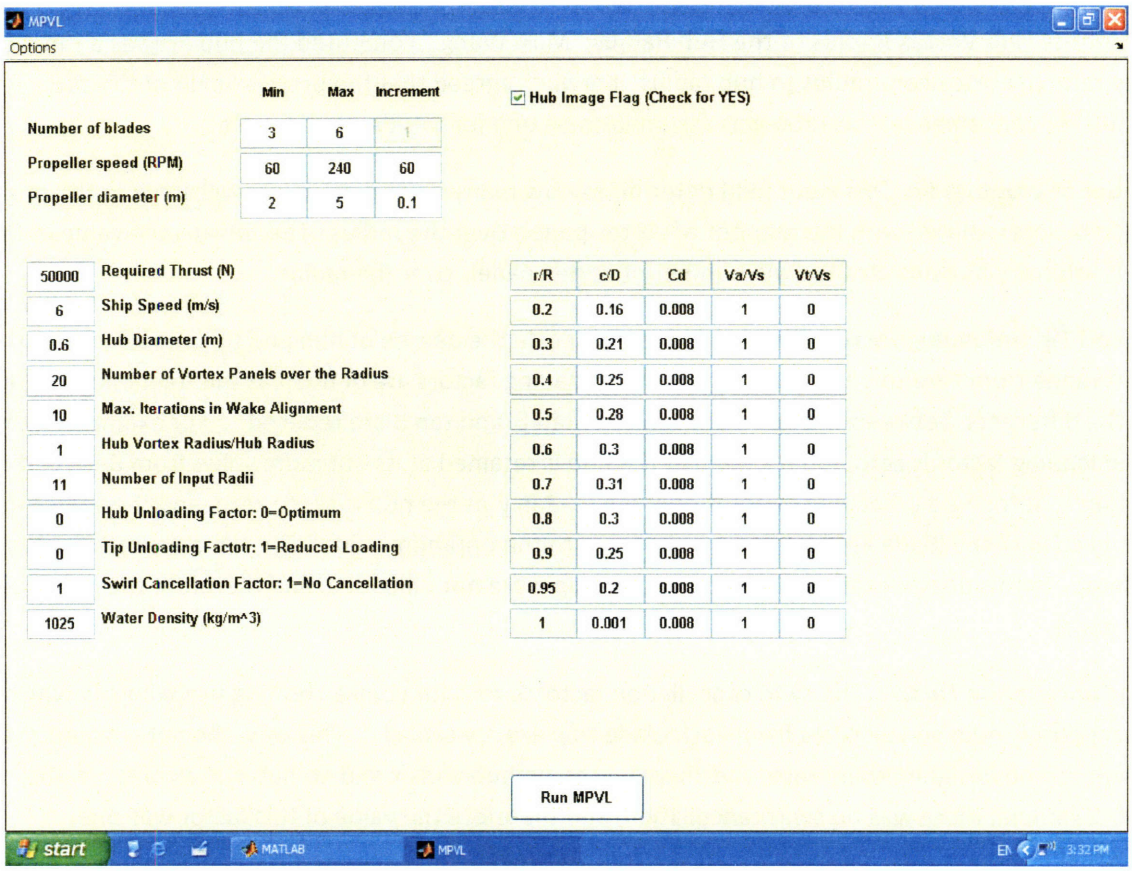


Figure 6. 1 Inputs for Parametric Analysis

Hub Diameter: Most propellers have a hub diameter which is greater than 15% of the propeller diameter. Normally, 20% to 30% of the propeller diameter would be a reasonable guess for this input field. The unit of this input field is in meter.

Number of Vortex Panels over the Radius: This input field represents how many vortex panels the blade will be divided into and thus effects the resolution of the propeller blade. The Cubic Hermite Polynomial Function[26] is then adopted to interpolate the chord distribution, the drag coefficient and the inflow velocities on each vortex panel. Usually, twenty panels would provide sufficient resolution. Increasing the number of panels will cost more time on unnecessary data interpolations.

Maximum Number of Iterations in Wake Alignment: This input field determines how many iterations MPVL is allowed to align the wake. Like PVL, MPVL is a wake-adapted method which calculates the circumferential volumetric mean inflow and then optimizes circulation and pitch distribution. Ten iterations are usually sufficient for the program to converge and align the wake.

[26] (Hanselman and Littlefield 2005, 340)

Ratio of the Hub Vortex Radius to the Hub Radius: M.H. Wang[27] computed the hub drag as a function of the ratio of vortex core radius to hub radius. Kerwin[28] agreed that the precise value of this ratio is not critical. For convenience, this ratio was assumed to be one for this design example.

Number of Input Radii: This input field determines how many input data points will serve as the original data to be interpolated with the number of vortex panels over the radius. That means the value of this field is arbitrary but smaller than the number of vortex panels over the radius.

Hub and Tip Unloading Factor: These two factors decide the degree of hub and tip unloading, and their values range from zero to one. The hub and tip unloading factors are defined as the fractional amount that the difference between the optimum values of tan β_i and tan β are reduced[29]. For example, if the hub unloading factor is zero, tan β_i - tan β at the hub is retained at its optimum value from Betz/Lerbs criterion[30]. If the hub unloading factor is one, tan β_i - tan β at the hub is set to zero, and the values up to the mid span of the blade are blended parabolically to the optimum value. The same procedure applies to the tip. In this design example, the hub and the tip were not unloaded, and therefore the two factors were zero.

Swirl Cancellation Factor: The swirl cancellation factor is zero for contra-rotating propellers in which the tangential induced velocities from each blade row exactly cancel[31]. This way, the net circulation at the hub can be designed to be zero, and there will be no hub vortex and no hub vortex drag. In this design example, there was no swirl cancellation, and therefore the value of this factor was one.

Water Density: The water density depends on the users' preference. The default value is 1,025 kg/m^3 for seawater.

Hub Image Flag: This input field is located on the upper right side of the GUI. By checking this option, a hub image is present, and the circulation has a finite value at the hub[32]. Unchecking this option will lead to a zero circulation at the root of the blade.

The tabulated editable textboxes on the right of the GUI contain four input fields. The first column of the table is the non-dimensional chord distribution (*c/D*) over the radius. The second column is the drag coefficient (Cd) over the radius. The third and the fourth columns are the ratios of the axial and tangential inflow velocities to the advance velocity over the radius. For simplicity, these four input fields are set to start at 20% of propeller radius (r/R=0.2) and end at the blade tip (r/R=1). MPVL automatically interpolates these four input fields in accordance with the number of input radii and the hub radius.

---

[27] (Wang 1985)
[28] (Kerwin 2001, 184)
[29] (Kerwin 2001, 185)
[30] (Kerwin 2001, 162)
[31] (Kerwin 2001, 184)
[32] (Kerwin 2001, 181)

Figure 6.1 shows a screen shot of the parametric analysis GUI and the value of each input field for this design example. Several assumptions must be mentioned. First, this propeller was assumed to operate in the open water and also uniform inflow. Thus the axial inflow velocity ratio (V_a/V_s) was set to one and the tangential inflow velocity ratio (V_t/V_s) was set to zero. Second, the drag coefficient was assumed to be 0.008 everywhere along the radius of the propeller. Finally, the non-dimensional chord distribution was arbitrary in this example. After all the input fields were revised, the parametric analysis was performed by clicking the "Run MPVL" pushbutton at the bottom of the GUI.

After the calculation process was complete, the propeller efficiency curves were plotted. See Figure 6.2. The data cursors were used to mark desired data points as shown in the figure. As expected, the propeller with a larger diameter, slower speed and less number of blades had the higher efficiency. However, in reality the propeller parameters must be restricted in size and speed. It is the purpose of the parametric analysis to reveal the efficiencies under different combinations of the propeller parameters to help the designers determine the optimum parameters for the design.

For this design example, it was assumed that the optimum propeller diameter was restricted to three meters due to the aft body geometry. An increase in the number of blades reduces the thrust per blade, thereby reducing the intensity of the disturbing forces[33] which cause propeller induced vibration. For that reason, a five-bladed design was adopted. Finally, the propeller speed was to be determined. The propeller speed can vary by means of mechanical gearing and therefore was flexible. It was shown in Figure 6.2 that a three-meter in diameter and five-bladed propeller had the highest efficiency at 120 RPM (green line). After all, the users are free to decide the constraints of the three propeller parameters. Now, the three major propeller parameters had been determined, and the single propeller design was ready to be conducted.

[33] (Edward 1988, 167)

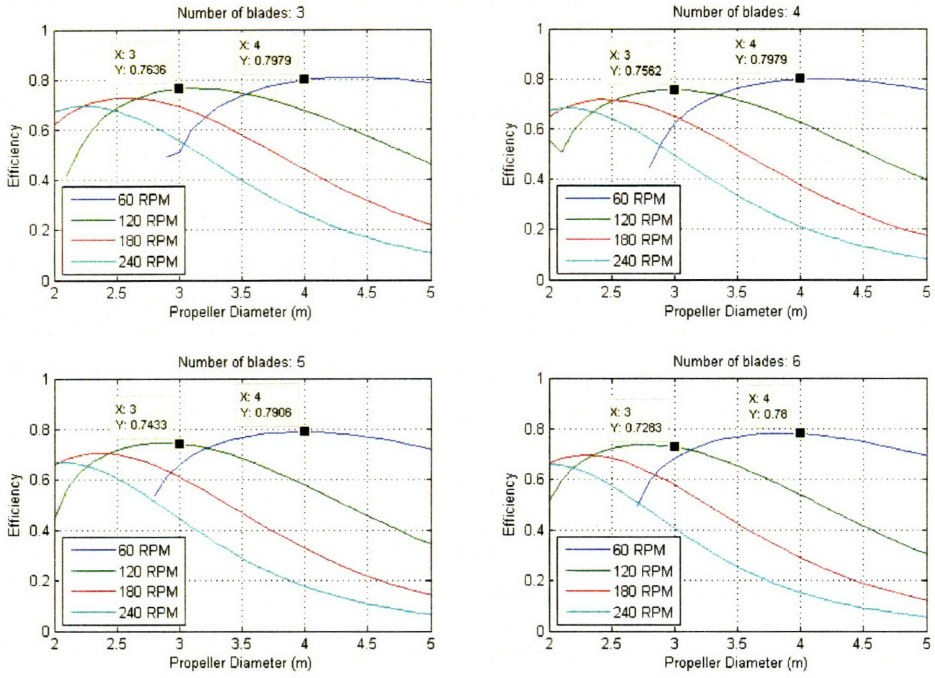


Figure 6. 2 Efficiency Curves of the Design Example

6.2 Single Propeller Design

In the previous section, the three major propeller parameters were determined: five-bladed, three-meter in diameter and with the optimum speed of 120 RPM. The single propeller design GUI was then launched. Most input fields were identical to the ones in the parametric analysis GUI. However, several more input fields were required in the single propeller design GUI. Figure 6.3 shows a screen shot of the single propeller design GUI and the inputs for this design example.

At the lower left corner of the GUI, four additional input fields are shown. The first one is the shaft centerline depth in meters. It is required for the calculation of cavitation number (σ). The second input field is the inflow velocity variation. It is required for the calculation of pitch angle variation ($\delta\beta_i$) which is related to the cavitation buckets. The third input field is the ideal angle of attack. It is required for specific meanline type to calculate the pitch angle. The fourth input field is the number of points over the chord. It is required to decide the resolution of the propeller blade.

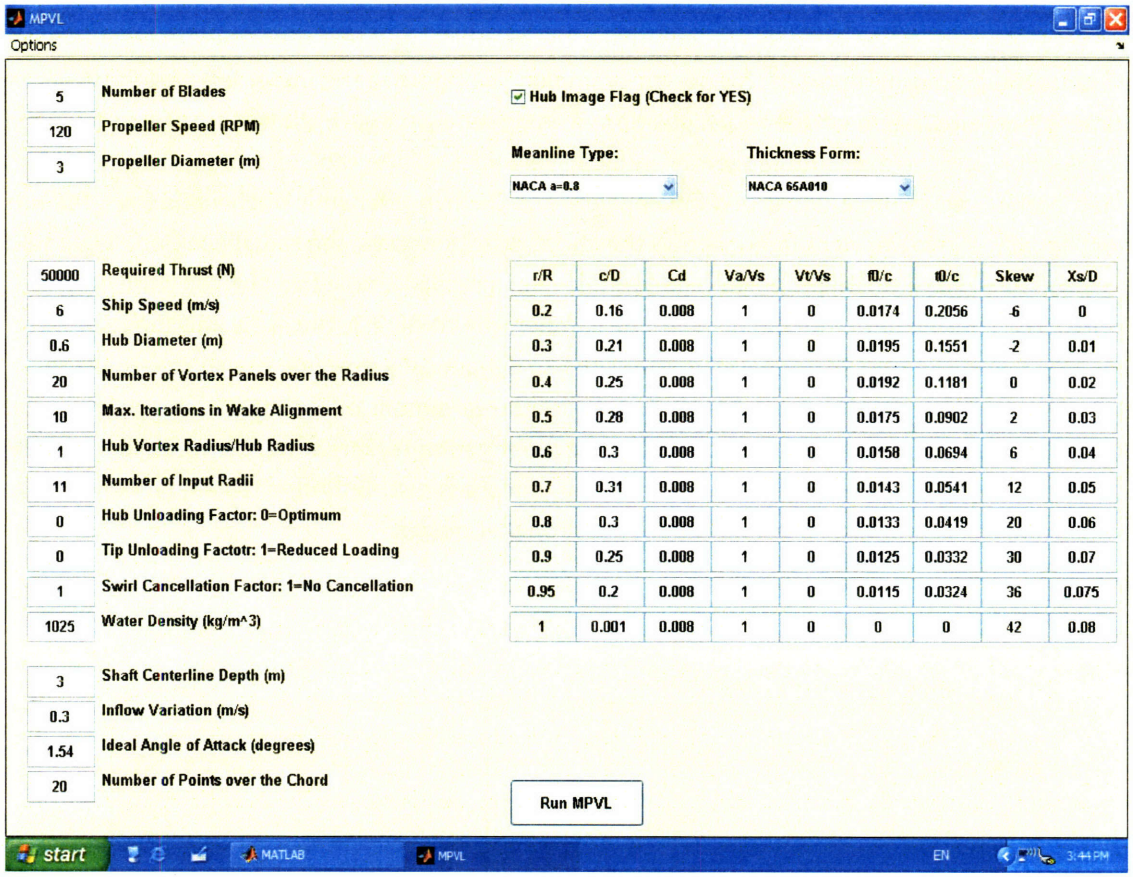


Figure 6. 3 Inputs for Single Propeller Design Example

On the upper right side of the GUI, two pop-up menus are shown. One of them is for selecting the meanline type, and the other one is for selecting the thickness form. Two options are available for the meanline type: the NACA a=0.8 and the parabolic meanline. Three thickness forms are available: the NACA 65 A010, the elliptical and the parabolic thickness forms. For this design example, the NACA a=0.8 meanilne and the NACA 65 A010 thickness form were selected.

Finally, the maximum camber distribution (f_0/c), the maximum thickness distribution (t_0/c), the skew in degrees and the non-dimensional rake (X_s/D) were required for propeller geometry calculations. Skew is often applied to reduce the propeller-induced unsteady forces and susceptibility to cavitation when operating in a wake[34]. Rake is often adopted to increase the clearance from the hull thereby reducing the propeller induced vibration. After all input fields were revised, the single propeller design was performed by clicking "Run MPVL" pushbutton at the bottom of the GUI.

[34] (Edward 1988, 184)

After the calculation process was complete, the graphical and text reports were created. Figure 6.4 presents the graphical report of this design example. The figure in the upper left corner shows the non-dimensional circulation vs. the radial position. The upper right figure shows the axial and tangential inflow velocities and the axial and tangential induced velocities vs. the radial position. The figure in the lower left corner shows the undisturbed flow angle β and the hydrodynamic pitch angle β_i vs. the radial position. Finally, the lower right figure shows the chord distribution vs. the radial position. Figure 6.5 presents the two-dimensional propeller blade profile. For simplicity, only five profiles were plotted in the figure. The chord length, the pitch angle, the camber, the thickness, the skew and the rake were shown. This figure can also be animated in MATLAB® to dynamically examine the change in blade profile from the root to the tip of the blade. Finally, the three-dimensional propeller image was created. See Figure 6.6. This figure presents the resultant propeller geometry including the propeller diameter, hub diameter, number of blades, camber distribution, thickness distribution, pitch angle, skew and rake. It provides an instant graphical presentation of the propeller design.

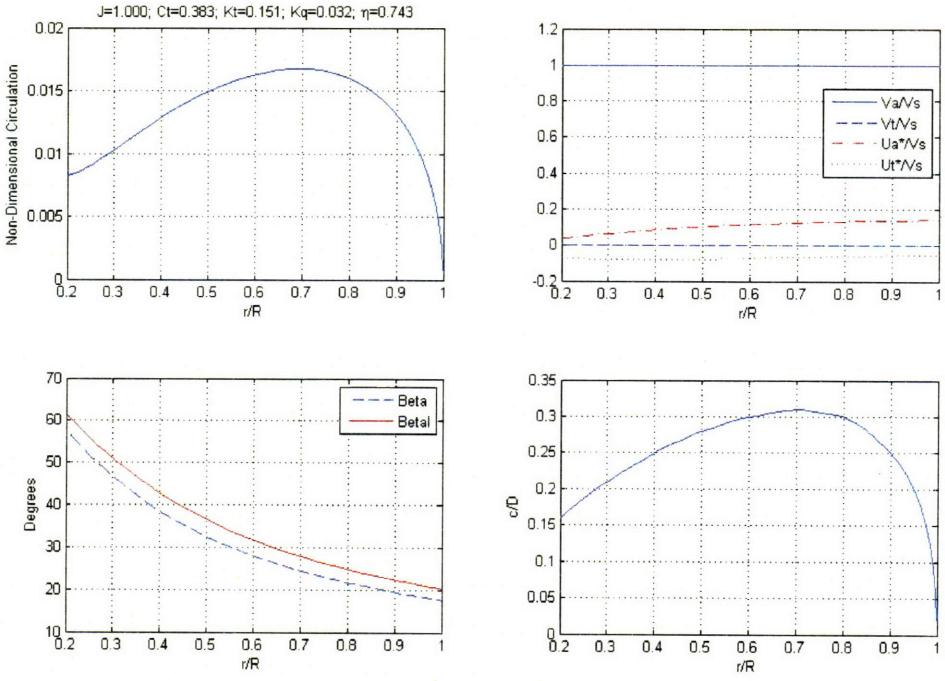


Figure 6. 4 Graphical Report of the Design Example

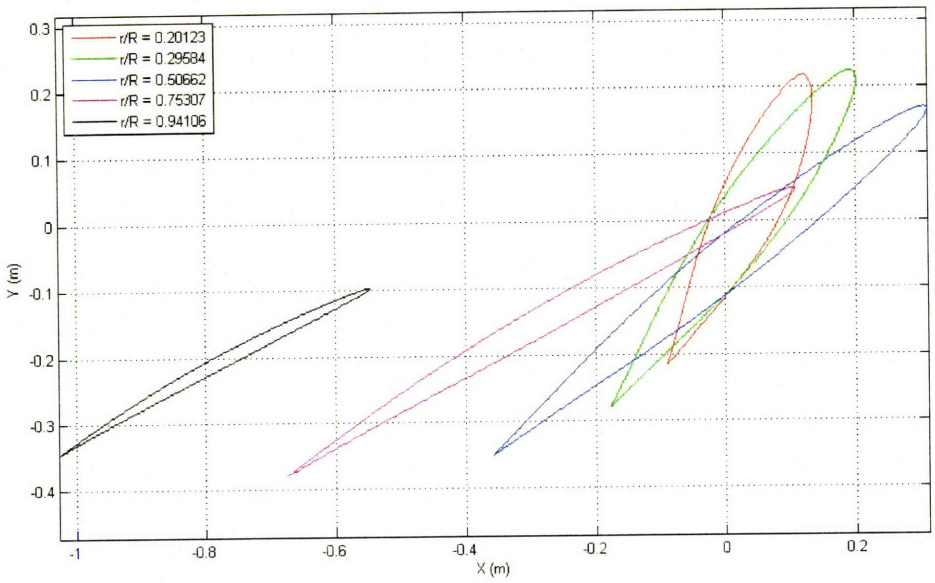


Figure 6. 5 2D Blade Profile of the Design Example

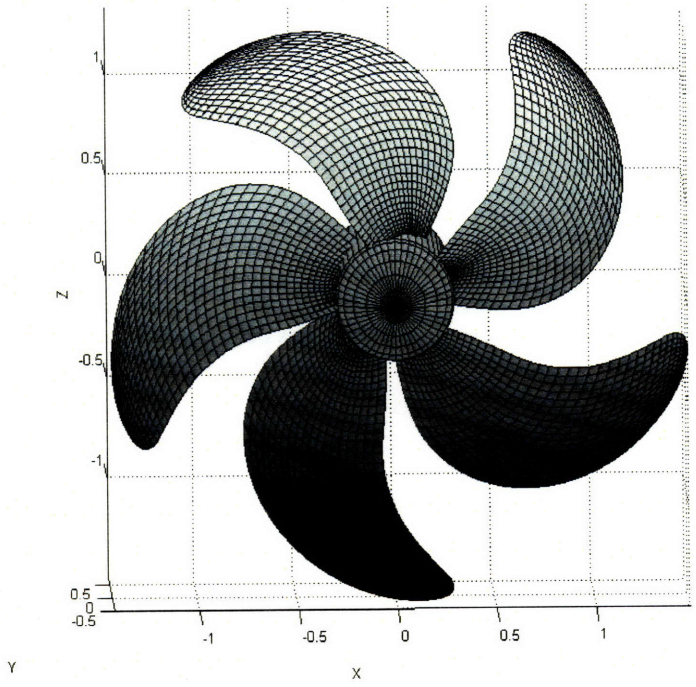


Figure 6. 6 3D Propeller Image of the Design Example

Four text files were also created to present and store the important information of this specific trial. Table 6.1 was the first table created. It contains the input data of MPVL and was identical to the input file of PVL. Table 6.2 was the second table created. It contains the outputs of MPVL and was identical to the output file of PVL. Table 6.3 was the propeller performance table which was the third table created. This table was intended for the cavitation calculations. It provides information on the sectional lift coefficient (C_L), the cavitation number (σ) and the pitch angle variation ($\delta\beta_i$). With this information, a Brockett diagram[35] can be used to check for cavitation for NACA foils. Table 6.4 was the last table created. It contains the essential propeller geometry data which can be used for manufacturing purpose.

```
2007-03-07 15:45:49 MPVL_Input.txt
20      Number of Vortex Panels over the Radius
10      Max. Iterations in Wake Alignment
1       Hub Image Flag: 1=YES, 0=NO
1.0     Hub Vortex Radius/Hub Radius
11      Number of Input Radii
5       Number of Blades
1.000   Advance Coef., J, Based on Ship Speed
0.383   Desired Thrust Coef., Ct
0       Hub Unloading Factor: 0=optimum
0       Tip Unloading Factor: 1=Reduced Loading
1       Swirl Cencellation Factor: 1=No Cancellation
r/R       C/D      XCD      Va/Vs  Vt/Vs
0.20000  0.16000  0.00800    1.00  0.0000
0.25000  0.18632  0.00800    1.00  0.0000
0.30000  0.21000  0.00800    1.00  0.0000
0.40000  0.25000  0.00800    1.00  0.0000
0.50000  0.28000  0.00800    1.00  0.0000
0.60000  0.30000  0.00800    1.00  0.0000
0.70000  0.31000  0.00800    1.00  0.0000
0.80000  0.30000  0.00800    1.00  0.0000
0.90000  0.25000  0.00800    1.00  0.0000
0.95000  0.20000  0.00800    1.00  0.0000
1.00000  0.00100  0.00800    1.00  0.0000
```

Table 6. 1 MPVL Input Text File of the Design Example

[35] (Edward 1988, 210)

MPVL_Output.txt
MPVL Output Table

Ct= 0.3834
Cp= 0.5158
Kt= 0.1506
Kq= 0.0322
Va/Vs= 1.0000
Efficiency= 0.7433

r/R	G	Va	Vt	Ua	Ut	Beta	BetaI	c/D	Cd
0.20123	0.008304	1.00000	0.0000	0.03698	-0.06796	57.699	61.449	0.16068	0.00800
0.21105	0.008357	1.00000	0.0000	0.03977	-0.06968	56.454	60.288	0.16604	0.00800
0.23045	0.008631	1.00000	0.0000	0.04527	-0.07266	54.096	58.073	0.17635	0.00800
0.25894	0.009246	1.00000	0.0000	0.05325	-0.07605	50.872	55.002	0.19075	0.00800
0.29584	0.010201	1.00000	0.0000	0.06316	-0.07896	47.096	51.343	0.20818	0.00800
0.34022	0.011398	1.00000	0.0000	0.07420	-0.08066	43.094	47.388	0.22713	0.00800
0.39100	0.012700	1.00000	0.0000	0.08544	-0.08081	39.149	43.407	0.24687	0.00800
0.44693	0.013970	1.00000	0.0000	0.09608	-0.07951	35.459	39.608	0.26533	0.00800
0.50662	0.015094	1.00000	0.0000	0.10560	-0.07709	32.141	36.130	0.28157	0.00800
0.56862	0.015986	1.00000	0.0000	0.11375	-0.07399	29.240	33.041	0.29498	0.00800
0.63138	0.016574	1.00000	0.0000	0.12052	-0.07060	26.755	30.360	0.30420	0.00800
0.69338	0.016797	1.00000	0.0000	0.12602	-0.06722	24.658	28.075	0.30992	0.00800
0.75307	0.016593	1.00000	0.0000	0.13042	-0.06405	22.913	26.156	0.30687	0.00800
0.80900	0.015904	1.00000	0.0000	0.13389	-0.06121	21.478	24.568	0.29814	0.00800
0.85978	0.014689	1.00000	0.0000	0.13660	-0.05876	20.316	23.275	0.27676	0.00800
0.90416	0.012935	1.00000	0.0000	0.13867	-0.05672	19.395	22.246	0.24696	0.00800
0.94106	0.010670	1.00000	0.0000	0.14022	-0.05511	18.688	21.455	0.21196	0.00800
0.96955	0.007965	1.00000	0.0000	0.14131	-0.05390	18.175	20.879	0.16555	0.00800
0.98895	0.004921	1.00000	0.0000	0.14201	-0.05311	17.842	20.504	0.10815	0.00800
0.99877	0.001664	1.00000	0.0000	0.14235	-0.05271	17.677	20.319	0.03954	0.00800

Table 6. 2 MPVL Output Text File of the Design Example

Performance.txt
Propeller Performance Table

r/R	V*	beta	betai	Gamma	Cl	Sigma	dBetai
0.201	3.867	57.70	61.45	0.4696	0.504	16.394	2.07
0.211	4.044	56.45	60.29	0.4726	0.469	14.968	2.12
0.230	4.397	54.10	58.07	0.4881	0.420	12.633	2.22
0.259	4.919	50.87	55.00	0.5229	0.371	10.061	2.32
0.296	5.599	47.10	51.34	0.5768	0.330	7.730	2.41
0.340	6.423	43.09	47.39	0.6446	0.295	5.843	2.46
0.391	7.370	39.15	43.41	0.7182	0.263	4.410	2.46
0.447	8.417	35.46	39.61	0.7900	0.236	3.358	2.42
0.507	9.537	32.14	36.13	0.8536	0.212	2.596	2.35
0.569	10.702	29.24	33.04	0.9040	0.191	2.046	2.25
0.631	11.884	26.75	30.36	0.9373	0.173	1.646	2.15
0.693	13.051	24.66	28.07	0.9499	0.157	1.354	2.05
0.753	14.176	22.91	26.16	0.9383	0.144	1.139	1.95
0.809	15.230	21.48	24.57	0.8994	0.132	0.980	1.86
0.860	16.188	20.32	23.28	0.8306	0.124	0.862	1.79
0.904	17.024	19.39	22.25	0.7314	0.116	0.774	1.73
0.941	17.720	18.69	21.45	0.6034	0.107	0.711	1.68
0.970	18.257	18.18	20.88	0.4504	0.099	0.668	1.64
0.989	18.623	17.84	20.50	0.2782	0.092	0.640	1.62
0.999	18.808	17.68	20.32	0.0941	0.084	0.627	1.61

Table 6. 3 MPVL Propeller Performance Text File of the Design Example

Geometry.txt

Propeller Geometry Table

Propeller Diameter = 3.0 m
Number of Blades = 5
Propeller Speed= 120 RPM
Propeller Hub Diameter = 0.60 m
Meanline Type: NACA a=0.8
Thickness Type: NACA 65A010

r/R	P/D	Skew	Xs/D	c/D	f0/c	t0/c
0.201	1.24	-5.9	0.000	0.161	0.0088	0.2049
0.211	1.24	-5.5	0.001	0.166	0.0083	0.1993
0.230	1.23	-4.5	0.003	0.176	0.0077	0.1887
0.259	1.23	-3.4	0.006	0.191	0.0071	0.1741
0.296	1.23	-2.1	0.010	0.208	0.0064	0.1569
0.340	1.23	-1.1	0.014	0.227	0.0057	0.1389
0.391	1.23	-0.2	0.019	0.247	0.0051	0.1210
0.447	1.23	0.9	0.025	0.265	0.0044	0.1040
0.507	1.23	2.2	0.031	0.282	0.0037	0.0886
0.569	1.23	4.5	0.037	0.295	0.0031	0.0753
0.631	1.23	7.6	0.043	0.304	0.0026	0.0641
0.693	1.24	11.5	0.049	0.310	0.0023	0.0550
0.753	1.24	16.0	0.055	0.307	0.0020	0.0472
0.809	1.25	20.8	0.061	0.298	0.0017	0.0410
0.860	1.25	25.7	0.066	0.277	0.0016	0.0357
0.904	1.25	30.5	0.070	0.247	0.0014	0.0331
0.941	1.25	34.9	0.074	0.212	0.0013	0.0326
0.970	1.26	38.3	0.077	0.166	0.0009	0.0257
0.989	1.26	40.7	0.079	0.108	0.0003	0.0105
0.999	1.26	41.9	0.080	0.040	0.0000	0.0012

Table 6. 4 MPVL Geometry Text File of the Design Example

Equations 6.1 to 6.5 were the formulas used in the propeller performance calculation.

Total Inflow Velocity: $$V^* = \sqrt{(V_a + u_a^*)^2 + (\omega r + V_t + u_t^*)^2} \quad (6.1)$$

Circulation: $$\Gamma = G \times 2\pi R V_A \quad (6.2)$$

Sectional Lift Coefficient: $$C_L = \frac{2\,\Gamma}{V^* c} \quad (6.3)$$

Sectional Cavitation number: $$\sigma = \frac{P_\infty - P_{vapor}}{\frac{1}{2}\rho V^{*2}} \quad (6.4)$$

Pitch Angle Variation: $$\delta\beta_i = tan^{-1}\left(\frac{\tan(\beta_i)\cdot\omega r + \delta V_A}{\omega r}\right) - tan^{-1}\left(\frac{\tan(\beta_i)\cdot\omega r - \delta V_A}{\omega r}\right) \quad (6.5)$$

V_a is the axial inflow velocity. V_t is the tangential inflow velocity. u_a^* is the induced axial velocity. u_t^* is the induced tangential velocity. ω is the propeller angular velocity. r is the local radius. G is the non-dimensional circulation. R is the propeller radius. V_A is the advance velocity. c is the chord length. P_∞ is the hydrostatic pressure. P_{vapor} is the vapor pressure. β_i is the hydrodynamic pitch angle, and δV_A is the axial inflow variation.

7 Conclusions and Recommendations

7.1 Conclusions

The FORTRAN version of PVL was successfully implemented in MATLAB® and presented with GUIs in this thesis. The enhanced program, MPVL, was proved to be consistent with PVL under various loading conditions in chapter five. Several advantages of MPVL over PVL were revealed.

First, MATLAB® is a high-level technical computing language, and its built-in functions relieve the programmers from writing lengthy and trivial functions. Therefore, the MPVL code is more compact and easier to understand than the PVL code. Moreover, the MPVL code can be easily modified by the users according to their specific needs while the PVL code requires a special compiler. This advantage gives MPVL the potential to be extended for complex propeller design applications.

Second, MPVL's simple and intuitive GUIs allow the propeller designers to conduct both the parametric analysis and the single propeller design. The efficiency curves generated by the parametric analysis GUI help the designers optimize the propeller parameters. The single propeller design GUI then performs a complete design including the propeller images and geometry data in just seconds.

Third, the MPVL GUIs are user-friendly and demand so little from the users that even people with little experiences and knowledge in propeller design can navigate the program without difficulty. Also, the data visualization capability of MATLAB® converts the texts into colorful graphs. Instead of reading tedious texts, users now are able to visually examine the propeller design and its data instantly. This advantage makes MPVL a great tool for teaching basic propeller design.

Finally, the low-level GUI developing approach had reduced the amount of work required for creating the GUIs. Especially the switchyard programming structure enabled the codes of different functions to be integrated into one single M-file. Thus no additional file was required, and efforts for managing the program were minimal.

In summary, it can be concluded that MPVL can replace PVL and serve as a tool for preliminary propeller design. The advantages mentioned above enable MPVL to serve as an excellent base program for further extension. With further efforts, add-ons can be attached to MPVL to perform more sophisticated propeller designs such as ducted propellers and contra-rotating propellers.

7.2 Recommendations for Further Work

MPVL can serve as a base program for complex propeller designs, and thus further efforts are required to extend its applications. The following are some potential topics for further work.

- Replacing former FORTRAN subroutines with MATLAB built-in functions to further simplify and accelerate MPVL.
- Adding features such as ducted and contra-rotating propellers to the program.
- Adding functions that handle the wake field of the ship as the inflow to the propeller.
- Incorporating MPVL with Propeller Blade Design Program (PBD) to conduct detailed propeller blade design.
- Adding features for calculating the structural strength of the propeller blade.
- Incorporating VLM and the Brockett diagrams to enable cavitation calculations for NACA foils.
- Adding help functions to the GUIs.

Bibliography

Abbott, Ira H., and Albert E. Von Doenhoff. *Theory of Wing Sections.* New York: Dover Publications, 1959.

Carlton, J. S. *Marine Propellers and Propulsion.* Oxford: Butterworth-Heinemann Ltd., 1994.

Coney, William B., Ching-Yeh Hsin, and Justin E. Kerwin. *MIT-PLL-2 Report and User's Manual.* Cambridge, MA: Massachusetts Institute of Technology, 1986.

Edward, V. Lewis. *Principles of Naval Architecture, Vol II - Resistance, Propulsion and Vibration.* Jersy City, NY: The Society of Naval Architect and Marine Engineers, 1988.

Faulkner, V. M. *The Solutionof Lifting-Panel Problems by Vortex Lattice Theory.* Aeronautical Research Council, 1947.

Glauert, Hermann. *Elements of Airfoil Airscrew Theory.* Cambridge University Press, 1926.

Hanselman, Duane, and Bruce Littlefield. *Mastering MATLAB 7.* Upper Saddle River, NJ: Pearson Education Inc., 2005.

Kerwin, Justin E. *2.23 Lecture Notes - Hydrofoils and Propellers.* Cambridge, Massachusetts: Massachusetts Institute of Technology, 2001.

Lan, C. E. "A Quasi-Vortex Lattice Method in Thin Wing Theory." *Aircraft, Vol 11, No. 9*, September 1974.

Lee, H. S., H. Gu, and S. A. Kinnas. *MPUF3A User's Manual and Documentation Ocean Engineering Report 04-2.* Austin, Texas: Ocean Engineering Group, University of Texas, 2004.

Marchand, Patrick, and Thomas Holland. *Graphics and GUIs with MATLAB 3rd Edition.* BocaRaton: CRC Press, 2002.

Wang, M-H. *Hub Effects in Propeller Design and Analysis, PhD Thesis, Department of Ocean Engineering.* Cambridge, MA: Massachusetts Institute of Technology, 1985.

Woud, Hans Klein, and Douwe Stapersma. *Design of Propulsion and Electric Power Generation Systems.* London: The Institute of Marine Engineering, Science and Technology, 2003.

Appendix A. Hydrofoil Vortex Lattice Lifting Line (VLL) Code

A.1 VLL.m

```
% ========================== Last modified: 04/27/07 by Hsin-Lung Chung
% ================================================================================
% Copyright 2007 Hsin-Lung Chung
% This program is free software; you can redistribute it and/or modify
% it under the terms of the GNU General Public License version 2 as
% published by the Free Software Foundation.
% This program is distributed in the hope that it will be useful, but
% WITHOUT ANY WARRANTY; without even the implied warranty of
% MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
% See the GNU General Public License for more details.
% ================================================================================

% This file contains the callback functions for the objects in VLL GUI.
% This program only executes when "VLL.fig" is present in the same
% directory.

function varargout = VLL(varargin)

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @VLL_OpeningFcn, ...
                   'gui_OutputFcn',  @VLL_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);

if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT

function VLL_OpeningFcn(hObject, eventdata, handles, varargin)
    handles.output = hObject;
    guidata(hObject, handles);

function varargout = VLL_OutputFcn(hObject, eventdata, handles)
    varargout{1} = handles.output;

% ================================================ Editable Text Boxes
function inset_Callback(hObject, eventdata, handles)
function inset_CreateFcn(hObject, eventdata, handles)
if ispc && isequal(get(hObject,'BackgroundColor'), ...
    get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
```

```
function N_Callback(hObject, eventdata, handles)
function N_CreateFcn(hObject, eventdata, handles)
if ispc && isequal(get(hObject,'BackgroundColor'), ...
    get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function a1_Callback(hObject, eventdata, handles)
function a1_CreateFcn(hObject, eventdata, handles)
if ispc && isequal(get(hObject,'BackgroundColor'), ...
    get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function a2_Callback(hObject, eventdata, handles)
function a2_CreateFcn(hObject, eventdata, handles)
if ispc && isequal(get(hObject,'BackgroundColor'), ...
    get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function a3_Callback(hObject, eventdata, handles)
function a3_CreateFcn(hObject, eventdata, handles)
if ispc && isequal(get(hObject,'BackgroundColor'), ...
    get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function a4_Callback(hObject, eventdata, handles)
function a4_CreateFcn(hObject, eventdata, handles)
if ispc && isequal(get(hObject,'BackgroundColor'), ...
    get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function a5_Callback(hObject, eventdata, handles)
function a5_CreateFcn(hObject, eventdata, handles)
if ispc && isequal(get(hObject,'BackgroundColor'), ...
    get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

% ============================================================ UIPanel1
function uipanel1_SelectionChangeFcn(hObject,eventdata,handles)
    if get(handles.Vcosine,'Value')==0 && get(handles.Vconstant,'Value')==0
        set(handles.Vcosine,'Value',1)
    end

    if get(handles.Vcosine,'Value')==1
        set(handles.inset,'Visible','off')
    else
        set(handles.inset,'Visible','on')
    end

% ============================================================ UIPanel2
function uipanel2_SelectionChangeFcn(hObject,eventdata,handles)
    if get(handles.CPcosine,'Value')==0 && get(handles.CPMid,'Value')==0
```

```
        set(handles.CPcosine,'Value',1)
    end

% ================================================== Run Push Button
function Run_Callback(hObject, eventdata, handles)
    if get(handles.Vcosine,'Value') == 1
        input(1,1) = 0;
    else
        input(1,1) = 1;
    end

    input(1,2) = str2num(get(handles.inset,'String'));

    if get(handles.CPcosine,'Value') == 1
        input(1,3) = 0;
    else
        input(1,3) = 1;
    end

    input(1,4) = str2num(get(handles.N,'String'));
    input(1,5) = str2num(get(handles.a1,'String'));
    input(1,6) = str2num(get(handles.a2,'String'));
    input(1,7) = str2num(get(handles.a3,'String'));
    input(1,8) = str2num(get(handles.a4,'String'));
    input(1,9) = str2num(get(handles.a5,'String'));

    % Call "Main" function and pass inputs
    [yc,W,Wnum,G,Gnum] = Main(input);

    axes(handles.Plots);
    plot(yc,G,':*r',yc,Gnum,':ob',yc,W,'-*r',yc,Wnum,'-ob')
    legend('G-exact','G-numerical','W-exact','W-numerical');
    title('VLL Graphical Report');          grid on;
    xlabel('Spanwise Position y/s');
    ylabel('Gamma/(U*s) and  W*/U');
    clear;  clc;

% ================================================== Print Pushbutton
function Print_Callback(hObject, eventdata, handles)
    printpreview;

% ================================================== Exit Pushbutton
function Exit_Callback(hObject, eventdata, handles)
    clear;  clc;    close all;
```

A.2 Main.m

```
% ========================== Last modified: 03/18/07 by Hsin-Lung Chung
% HYDROFOIL VORTEX LATTICE LINE PROGRAM(VLL)
% This function contains the algorithm for VLL.

function [yc,W,Wnum,G,Gnum] = Main(input)

% ========================================================= Define inputs
Vspacing = input(1,1);      inset = input(1,2);     CPspacing =input(1,3);
N = input(1,4);             a1 = input(1,5);        a2 = input(1,6);
a3 = input(1,7);            a4 = input(1,8);        a5 = input(1,9);
U = 1;       span = 1;      rho = 1; % (For non-dimensional Calculation)

% ============================================================ Algorithm
if Vspacing == 0            % Cosine spacing is selected for vortices
    delta = [0:pi/N:pi];
    yv = -.5.* cos(delta);      % Location of vortices
else                            % Constant spacing is selected
    yv = [-.5+inset/(N+inset*2):1/(N+inset*2):.5-inset/(N+inset*2)];
    delta = acos(yv/-.5);
end

for i = 1:N
    % Convert the locations into spanwise
    if CPspacing == 0   % Cosine spacing is selected for control points
        angle(i) = (delta(i+1)+delta(i)) / 2;
        yc(i) = -.5.* cos(angle(i));
    else                    % Mid-Point spacing is selected for CPs
        yc(i) = (yv(i)+yv(i+1)) / 2;
        angle(i) = acos(yc(i)/-.5);
    end

    % Glauert's Method for circulation
    G(i) = 2*U*span*(a1*sin(angle(i)) + a2*sin(2*angle(i)) + ...
    a3*sin(3*angle(i)) + a4*sin(4*angle(i)) + a5*sin(5*angle(i)));

    % Calculate Downwash Velocity
    W(i) = -U*( a1*sin(angle(i))+ 2*a2*sin(2*angle(i))+3*a3*sin(3*angle(i))...
          +4*a4*sin(4*angle(i))+5*a5*sin(5*angle(i)))/sin(angle(i));

    for j = 1:N    % Influence Matrix (i: CP Index; j: Vortex Index)
        I(i,j) = 1/(4*pi*(yv(j)-yc(i))) - 1/(4*pi*(yv(j+1)-yc(i)));
    end
end

Lift = pi/2*rho*U^2*span^2*a1;
Drag = pi/2*rho*U^2*span^2*( a1^2 + 2*a2^2 + 3*a3^2 + 4*a4^2 + 5*a5^2 );

DLRatio = Drag/Lift^2;

Wnum = G*I' ;          % w = G/(4*pi*y);  Numerical Washdown Velocity
Gnum = I\W';           % G = I\W;  Numerical Circulation
```

```
for k = 1:N
    % Lift and Drag based on the given circulation
    L(k) = rho * U * G(k) * (yv(k+1) - yv(k));
    D(k) = -rho * ( Wnum(k) * G(k) * (yv(k+1)-yv(k)) );

    % Lift and Drag based on the given washdown
    Lc(k) = rho*U * Gnum(k) * (yv(k+1) - yv(k));
    Dc(k) = -rho * ( W(k)*Gnum(k) * (yv(k+1)-yv(k)) );
end

Lnum = sum(L);
Lcirc = sum(Lc);
Dnum = sum(D);
Dcirc = sum(Dc);
DLRatioNum = Dnum / Lnum^2;
DLRatioCirc = Dcirc / Lcirc^2;

% Percent Error in Numerical Solution
NumLiftError = 100*(Lnum-Lift) / Lift;
NumDragError = 100*(Dnum-Drag) / Drag;
NumDLRatioError = 100*(DLRatioNum-DLRatio) / DLRatio;

% Percent Error in Circulation Solution
CircLiftError = 100*(Lcirc-Lift) / Lift;
CircDragError = 100*(Dcirc-Drag) / Drag;
CircDLRatioError = 100*(DLRatioCirc-DLRatio) / DLRatio;

% ========================================================== Text Report
fid = fopen('Report.txt','w');
fprintf(fid,'VORTEX LIFTING LINE SOLUTION WITH %g ELEMENTS\n\n',N);
fprintf(fid,'CIRCULATION COEFFICIENTS\n');
fprintf(fid,'%6.3f  %6.3f  %6.3f  %6.3f  %6.3f\n',a1,a2,a3,a4,a5);

if (Vspacing == 0) && (CPspacing == 0)
    fprintf(fid,'Cosine spaced vortices and control points\n');
elseif (Vspacing == 1) && (CPspacing == 0)
    fprintf(fid,'Constant spaced vortices and cosine spaced control
        points\n');
elseif (Vspacing == 0) && (CPspacing == 1)
    fprintf(fid,'Cosine spaced vortices and Mid-point spaced control
        points\n');
else
    fprintf(fid,'Constant spaced vortices and Mid-point spaced control
        point\n');
end

fprintf(fid,'N\t\tYV\t\t\tYC\t\t  GAMMA\t\t  W(EXACT)\t  W(NUM)\t  G(NUM)\n');

for m = 1:N
    fprintf(fid,'%g\t %+5.4f\t %+5.4f\t %+5.4f\t %+5.4f\t %+5.4f\t %+5.4f\n',
        m,yv(m),yc(m),G(m),W(m),Wnum(m),Gnum(m));
end

fprintf(fid,'\t %+5.4f\n',yv(m+1));
fprintf(fid,'PERCENT ERROR IN NUMERICAL SOLUTION:\n');
fprintf(fid,'COMPUTING DOWNWASH FOR GIVEN CIRCULATION\n');
fprintf(fid,'LIFT: %5.2f  DRAG: %5.2f  DRAG/LIFT^2: %5.2f\n\n'...
```

```
    ,NumLiftError,NumDragError,NumDLRatioError);
fprintf(fid,'COMPUTING DOWNWASH FOR GIVEN DOWNWASH\n');
fprintf(fid,'LIFT: %5.2f  DRAG: %5.2f  DRAG/LIFT^2: %5.2f'...
    ,CircLiftError,CircDragError,CircDLRatioError);

fclose(fid);
fid = fopen('Report.txt','r');
open('Report.txt')
```

Appendix B. Two-Dimensional Vortex Lattice Method (VLM) Code

B.1 VLM.m

```
% ========================= Last modified: APR/27/07 by Hsin-Lung Chung
% ============================================================================
% Copyright 2007 Hsin-Lung Chung
% This program is free software; you can redistribute it and/or modify
% it under the terms of the GNU General Public License version 2
% as published by the Free Software Foundation.
% This program is distributed in the hope that it will be useful, but
% WITHOUT ANY WARRANTY; without even the implied warranty of
% MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
% See the GNU General Public License for more details.
% ============================================================================

% This file contains the callbacks functions for objects in VLM GUI.

function varargout = VLM(varargin)

% ============================== Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @VLM_OpeningFcn, ...
                   'gui_OutputFcn',  @VLM_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);

if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end

% ================================ End initialization code - DO NOT EDIT
function VLM_OpeningFcn(hObject, eventdata, handles, varargin)
    handles.output = hObject;
    guidata(hObject, handles);

function varargout = VLM_OutputFcn(hObject, eventdata, handles)
    varargout{1} = handles.output;

% ======================================================== Run Pushbutton
function Run_Callback(hObject, eventdata, handles)

global N CL Alpha TOC;

N = str2double(get(handles.N,'String'));
CL = str2double(get(handles.CL,'String'));
Alpha = str2double(get(handles.Alpha,'String'));
TOC = str2double(get(handles.TOC,'String'));
[xt,CPU, CPL, CLNum] = Main;     % Call "Main" function and pass inputs
```

```
% ============================================================ Plotting
set(handles.CLNum,'String',num2str(CLNum));
axes(handles.PressureCoeff);
plot(xt,CPU,'-r','LineWidth',2);        hold on;
plot(xt,CPL,':b','LineWidth',2);        hold off;
grid on;            xlim([0 1]);
xlabel('X/C');      ylabel('-Cp');
legend('Upper Surface','Lower Surface');
clear;

% ========================================================== Print Button
function Print_Callback(hObject, eventdata, handles)
    printpreview;

% ========================================================== Exit Button
function Exit_Callback(hObject, eventdata, handles)
    clc;        clear;      close all;

% ============================================ Editable Text Components
function N_Callback(hObject, eventdata, handles)    % Number of Panels
function N_CreateFcn(hObject, eventdata, handles)
if ispc && isequal(get(hObject,'BackgroundColor'),...
    get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function CL_Callback(hObject, eventdata, handles)   % Ideal Lift Coeff
function CL_CreateFcn(hObject, eventdata, handles)
if ispc && isequal(get(hObject,'BackgroundColor'),...
    get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function Alpha_Callback(hObject, eventdata, handles) % Angle of Attack
function Alpha_CreateFcn(hObject, eventdata, handles)
if ispc && isequal(get(hObject,'BackgroundColor'),...
    get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function TOC_Callback(hObject, eventdata, handles) % Thickness/Chord
function TOC_CreateFcn(hObject, eventdata, handles)
if ispc && isequal(get(hObject,'BackgroundColor'),...
    get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
```

B.2 Main.m

```
% ========================= Last modified: APR/27/07 by Hsin-Lung Chung
% 2D Vortex/Source Lattice with Lighthill Correction Program (VLM)
% This file contains the algorithms for VLM.

function [xt, CPU, CPL, CLNum] = Main;

global N CL Alpha TOC;

% ========================================================================
U = 1;          c = 1;          % Free Stream Velocity and Chord Length

for i = 1:N
   xv(i) = c/2 * (1-cos((i-1/2)*pi/N));     % Vortex Position
   xc(i) = c/2 * (1-cos(i*pi/N));           % CP Position
   dx(i) = pi * sqrt(xv(i)*(c-xv(i))) / N;  % Interval between vortices
end

for i = 1:N          % Influence Matrix A (i:CP; j:vortex)
    for j = 1:N
        A(i,j) = 1/(2*pi*(xv(j)-xc(i)));
    end
end

% ============================================================ Camber Term
[B,F,Gexact] = MeanLine(xv,xc);     % Function for NACA a=0.8 Mean Line

for i = 1:N
   B(i) = CL*B(i) - Alpha*pi/180;
   F(i) = CL*F(i);                  % Camber F
end

Gamma = (B/A');                     % Point Vortex Strength
G = Gamma./dx;                      % Vortex Sheet Strength
CLNum = 2*sum(Gamma);               % Numerical Lift Coeff

% ========================================================= Thickness Term
xt(1) = 0;                          % Thickness at the leading edge

for i = 1:length(xc)
   xt(i+1) = xc(i);
end

[RLE,yt,dydx] = Thickness(TOC, xt, xv);

for i = 1:N                         % i for CP; j for Vortices
    for j = 1:N
        ut(i,j) = (yt(j+1)-yt(j))/(xc(i)-xv(j))/(2*pi);
    end
    UT(i) = sum(ut(i,:));           % UT @ Control Points
end

UTVP = spline(xc,UT,xv);            % UT @ Vortex Points
```

```
% ======================================== Leading Edge Surface Velocity
QU = Alpha*pi/180*sqrt(2*c/RLE);  % Surface Velocity
CPU(1) = QU^2-1;                  % Minus Cp on the upper surface at LE
CPL(1) = CPU(1);                  % Minus Cp on the lower surface at LE

% ================================================== Surface Velocity
for i = 1:N
    if dydx(i)>0
        FLH(i) = 1/sqrt(1+dydx(i)^2);
    else
        FLH(i) = 1;
    end

    QU(i) = (1+UT(i)+1/2*G(i))*FLH(i);  % Velocity on Upper Surface
    CPU(i+1) = QU(i)^2-1;               % -Cp
    QL(i) = (1+UT(i)-1/2*G(i))*FLH(i);  % Velocity on Lower Surface
    CPL(i+1) = QL(i)^2-1;               % -Cp
end

% ============================================================ Plotting
figure(2);
subplot(2,2,1); plot(xc,Gamma,'-b','LineWidth',2);
xlabel('X/C');  ylabel('Non-Dimensional Circulation');     grid on;
title('Circulation Distribution');
subplot(2,2,2);
plot(xc,UT,'-b','LineWidth',2);
xlabel('X/C');  ylabel('Ut');   grid on;
title('Thickness Induced Velocity vs X')
subplot(2,2,3); plot(xv,F,'-b','LineWidth',2);
ylabel('Camber f/c');   xlabel('X/C');  grid on;
title('Camber Distribution');
subplot(2,2,4);
plot(xt,yt,'-b','LineWidth',2);
xlabel('X/C');  ylabel('Thickness t/c');    grid on;
title('Thickness Distribution')

% ========================================================== Text Report
fid = fopen('Report.txt','w');
fprintf(fid,'2D Vortex Lattice with Lighthill Correction Program (VLM)\n\n');
fprintf(fid,'Number of Panels: %g\n',N);
fprintf(fid,'Ideal Lift Coefficient: %5.2f\n',CL);
fprintf(fid,'Angle of Attack (Alpha-AlphaIdeal): %5.1f degree\n',Alpha);
fprintf(fid,'Maximum Thickness/Chord Ratio (T/C): %5.2f\n',TOC);
fprintf(fid,'Radius of Leading Edge (RLE/C): %5.5f\n',RLE);
fprintf(fid,'Calculated Lift Coefficient (CLNum): %5.5f\n\n',CLNum);
fprintf(fid,'  x/c\t  f/c\t  t/c\t Gamma\t   G\t\t UT\t\t UTVP\t  CPU\t
    CPL\n');

for i = 1:N
    fprintf(fid,'%6.4f  %6.4f  %6.4f  %+6.4f  %+6.4f  %+6.4f  %+6.4f  %+6.4f
        %+6.4f\n',xv(i),F(i),yt(i),Gamma(i),G(i),UT(i),UTVP(i),CPU(i),CPL(i));
end

fclose(fid);
open('Report.txt')
```

B.3 MeanLine.m

```
% Former FORTRAN Subroutine "AEIGHT" ========== APR/27/07 by H.L. Chung
function [B,F,Gexact] = MeanLine(xv,xc)

global U c N CL Alpha TOC;

a = 0.8;              % For NACA a=0.8
MC = length(xv);
g = -1/(1-a) * (a^2*(log(a)/2-1/4)+1/4);
h = 1/(1-a) * ((1-a)^2*log(1-a)/2 - (1-a)^2/4) + g;
AlphaIdeal = -h / (2*pi*(a+1));

for i = 1:MC
    C1 = max(1- xv(i),1e-6);
    CA = a - xv(i);
    if (abs(CA)<1e-6)
        CA = CA+1e-5;
    end

        P = 1/2*CA^2*log(abs(CA))-1/2*C1^2*log(C1)+1/4*(C1^2-CA^2);
        F(i)=(P/(1-a)-xv(i)*log(xv(i))+ g-h*xv(i))/(2*pi*(a+1))
              +C1*AlphaIdeal;

    if (xv(i)<=a)
        Gexact(i) = 1/(a+1);
    else
        Gexact(i) = 1/(a+1) * (1-xv(i))/(1-a);
    end
end

for j = 1:MC
    C1 = max(1-xc(j),1e-6);
    CA = a - xc(j);

    if (abs(CA)<1e-6)
        CA = CA+1e-5;
    end

    R = -(a-xc(j))*log(abs(CA))-1/2*CA+C1*log(C1)+1/2*C1;
    S = -1/2*C1+1/2*CA;
    T = -log(xc(j))-1-h;
    B(j) = ((R+S)/(1-a)+T)/(2*pi*(a+1)) - AlphaIdeal;
end
```

B.4 Thickness.m

```
% Function for "NACA66-010" ==================== APR/27/07 by H.L.Chung

function [RLE,YT,DYDX] = Thickness(thk, xt, xv)

PC = [0.000, 0.010, 0.025, 0.050, 0.100, 0.200, 0.300, 0.400, 0.450,...
      0.500, 0.600, 0.700, 0.800, 0.900, 0.950, 0.975, 0.990, 1.000];

% NACA 66(mod)from PNA vol. II(p136)
T66 = [0.0000, 0.1870, 0.2932, 0.4132, 0.5814, 0.8000, 0.9274,...
       0.9904, 1.0000, 0.9917, 0.9256, 0.7934, 0.5950, 0.3306,...
       0.1736, 0.0888, 0.0360, 0.0000];

RLE_CONST = 0.448;
NT = length(xt);
RLE = RLE_CONST*thk^2;
PSQ = sqrt(PC);
TRLE = 2*sqrt(2*RLE_CONST);
XSQ = sqrt(xt);
YSPLN = spline(PSQ,T66,XSQ);
YT = thk.*YSPLN;

for N=1:NT-1
   DYDX(N) = (YT(N+1)-YT(N))/(xt(N+1)-xt(N))/2;
end
```

Appendix C. MATLAB Propeller Vortex Lattice Lifting Line (MPVL) Code

C.1 MPVL.m

```
% MPVL Version 1.0 ====== Last modified: APR/27/2007 by Hsin-Lung Chung

% ======================================================================
% Copyright 2007 Hsin-Lung Chung
% This program is free software; you can redistribute it and/or modify
% it under the terms of the GNU General Public License version 2
% as published by the Free Software Foundation.
% This program is distributed in the hope that it will be useful, but
% WITHOUT ANY WARRANTY; without even the implied warranty of
% MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
% See the GNU General Public License for more details.
% ======================================================================

function MPVL(action)

if nargin==0                    % nargin is "Number of Input Arguments"
   action = 'Initiate_Fig';
end

% ------------------------------------------- Set up global variables
global Fig_Main Parametric Single Parametric_Flag Single_Flag XR0 ...
       XCHD_in XCD_in XVA_in XVT_in Text1_P Text2_P Text3_P NBLADE_in ...
       N_in D_in Input1 Hub_Flag Input1_S Input2_S Input3_S Text1_S ...
       Text2_S Text3_S f0oc_in t0oc_in skew_in rake_in ...
       Run_Parametric Run_Single Fig_P Fig1_S Fig2_S Fig3_S ...
       New_Parametric New_Single Err_Parametric Err_Single Label_Col ...
       Label_Row Welcome;

% ================================================ INITIATE MAIN FIGURE
if strcmp(action,'Initiate_Fig')==1

    % ================================= Declare Default Input Variables
    % Common_def: thrust,Vs,Dhub,MT,ITER,RHV,NX,HR,HT,CRP,rho
    Common_Def=[40000 5 .4 20 10 1 11 0 0 1 1025];
    XR0=[.2 .3 .4 .5 .6 .7 .8 .9 .95 1];                    % r/R
    % c/D
    XCHD_def=[.1600 .1818 .2024 .2196 .2305 .2311 .2173 .1806 .1387 .0010];
    XCD_def=ones(length(XR0)).*.008;                          % Cd
    XVA_def=ones(length(XR0));                                % Va/Vs
    XVT_def=zeros(length(XR0));                               % Vt/Vs
    % f0/c
    f0oc_def=[.0174 .0195 .0192 .0175 .0158 .0143 .0133 .0125 .0115 .0000];
    % t0/c
    t0oc_def=[.2056 .1551 .1181 .0902 .0694 .0541 .0419 .0332 .0324 .0000];
    skew_def=zeros(length(XR0));                            % Skew (degrees)
    rake_def=zeros(length(XR0));                            % Rake (Xs/D)
    Parametric_def=[3 6 1 50 200 50 2 5 .5];  % Z,N,D (Min,Max,Increment)
    Single_def1=[6 200 2];                    % NBLADE, N and D
    Single_def2=[3 .3 1.54 20];               % H, dV, AlphaI and NP

    % -------------- Parameters that define the layout of UI components
    ew = .06;     eh = .04;    tw1 = .15;    tw2 = .35;    ph = .06;
    X1 = .02;     X2 = .085;   X3 = .45;     Y1 = .93;     Y2 = .7;
```

```
Y3 = .18;    Y4 = .015;  h1 = eh+.005;

% ---------------------------------------- Create Figure for GUI
close all;

Fig_Main=figure('units','normalized','position',[.02 .3 .96 .6],...
    'numbertitle','off','name','MPVL','menubar','none');

Frame=uicontrol('parent',Fig_Main,'style','frame',...
    'units','normalized','position',[0 0 1 1]);

% ------------------------------------- Create UI menu components
Menu=uimenu(Fig_Main,'label','Options');
Parametric=uimenu(Menu,'label','Parametric Analysis','callback',...
    'MPVL(''Initiate_Parametric'')');
Single=uimenu(Menu,'label','Single Propeller Design','callback',...
    'MPVL(''Initiate_Single'')');
uimenu(Menu,'label','Print Screen','separator','on','callback', ...
    ['printpreview']);
uimenu(Menu,'label','Exit','separator','on','callback', ...
    ['clear all; close all; clc']);

% Flags that indicate the status of program
Parametric_Flag = 0;     Single_Flag = 0;
% ------------------------------------------------ Welcome message
Welcome=uicontrol('style','text','units','normalized', ...
    'FontSize',14,'FontWeight','bold','position', ...
    [.2 .4 .6 .2],'string', ...
    'To Start, Select an Action in the Option Menu.');

% ---------------------------------------------- Create Text Labels
Label1_P={'Min' 'Max' 'Increment'};
Label2_P={'Number of blades'
          'Propeller speed (RPM)'
          'Propeller diameter (m)'};
Label3_P={'Required Thrust (N)'                                  % THRUST
          'Ship Velocity (m/s)'                                  % V
          'Hub Diameter (m)'                                     % Dhub
          'Number of Vortex Panels over the Radius'              % MT
          'Max. Iterations in Wake Alignment'                    % ITER
          'Hub Vortex Radius/Hub Radius'                         % RHV
          'Number of Input Radii'                                % NX
          'Hub Unloading Factor: 0=Optimum'                      % HR
          'Tip Unloading Factotr: 1=Reduced Loading'             % HT
          'Swirl Cancellation Factor: 1=No Cancellation'         % CRP
          'Water Density (kg/m^3)'};                             % rho
Label1_S={'Number of Blades'                                     % NBLADE
          'Propeller Speed (RPM)'                                % N
          'Propeller Diameter (m)'};                             % D
Label2_S={'Shaft Centerline Depth (m)'                           % H
          'Inflow Variation (m/s)'                               % dV
          'Ideal Angle of Attack (degrees)'                      % AlphaI
          'Number of Points over the Chord'};                    % NP

% ---------------- Create UI components for Parametric Analysis GUI
for i=1:length(Label1_P)
    NBLADE_in(i)=uicontrol('style','edit','units','normalized', ...
```

```
        'FontSize',10, 'FontWeight','bold','backgroundcolor','w', ...
        'position',[tw1+ew*(i) Y1-h1 ew eh],'string', ...
        Parametric_def(i),'callback','MPVL(''Update'')','visible','off');
    N_in(i)=uicontrol('style','edit','units','normalized', ...
        'FontSize',10,'FontWeight','bold','backgroundcolor','w', ...
        'string',Parametric_def(i+3),'position', ...
        [tw1+ew*(i) Y1-h1*2 ew eh],'callback','MPVL(''Update'')', ...
        'visible','off');
    D_in(i)=uicontrol('style','edit','units','normalized', ...
        'FontSize',10,'FontWeight','bold','backgroundcolor','w', ...
        'position',[tw1+ew*(i) Y1-h1*3 ew eh],...
        'string',Parametric_def(i+6),'callback','MPVL(''Update'')', ...
        'visible','off');
    Text1_P(i)=uicontrol('style','text','units','normalized', ...
        'FontSize',9,'FontWeight','bold','position', ...
        [tw1+ew*(i) Y1 ew eh],'string',Label1_P(i),'visible','off');
    Text2_P(i)=uicontrol('style','text','units','normalized', ...
        'FontSize',10,'FontWeight','bold','HorizontalAlignment', ...
        'left','visible','off','position',[X1 Y1-h1*(i) tw1 eh], ...
        'string',Label2_P(i));
end

% ---------------------- Create UI components for Single Design GUI
for i = 1:length(Single_def1)
    Input1_S(i)=uicontrol('style','edit','units','normalized', ...
        'FontSize',10,'FontWeight','bold','backgroundcolor','w', ...
        'callback','MPVL(''Update'')','position', ...
        [X1 Y1-h1*(i-1) ew eh],'string',Single_def1(i),'visible','off');
    Text1_S(i)=uicontrol('style','text','units','normalized', ...
        'FontSize',10,'FontWeight','bold','HorizontalAlignment', ...
        'left','position',[X2 Y1-h1*(i-1) tw2 eh], ...
        'string',Label1_S(i),'visible','off');
end

for i = 1:length(Single_def2)
    Input2_S(i)=uicontrol('style','edit','units','normalized', ...
        'FontSize',10,'FontWeight','bold','backgroundcolor','w', ...
        'callback','MPVL(''Update'')','position', ...
        [X1 Y3-h1*(i-1) ew eh],'string',Single_def2(i),'visible','off');
    Text2_S(i)=uicontrol('style','text','units','normalized', ...
        'FontSize',10,'FontWeight','bold','HorizontalAlignment', ...
        'left','visible','off','position',[X2 Y3-h1*(i-1) tw2 eh], ...
        'string',Label2_S(i));
end

% ------------------------- Create UI components for Common Inputs
for i = 1:length(Common_Def)
    Input1(i)=uicontrol('style','edit','units','normalized', ...
        'FontSize',10,'FontWeight','bold','backgroundcolor','w', ...
        'callback','MPVL(''Update'')','position', ...
        [X1 Y2-h1*(i-1) ew eh],'string',Common_Def(i),'visible','off');
    Text3_P(i)=uicontrol('style','text','units','normalized', ...
        'FontSize',10,'FontWeight','bold','HorizontalAlignment', ...
        'left', 'visible','off','position',[X2 Y2-h1*(i-1) tw2 eh], ...
        'string',Label3_P(i));
end
```

```
Hub_Flag=uicontrol('style','checkbox','units','normalized', ...
    'FontSize',10,'FontWeight','bold','position',[X3 Y1 tw2 eh],...
    'value',1,'callback','MPVL(''Update'')','string', ...
    'Hub Image Flag (Check for YES)','visible','off');

% ------------------------------------------- Create POP-UP MENU
Input3_S(1)=uicontrol('style','popupmenu','units','normalized', ...
    'FontSize',8,'FontWeight','bold','backgroundcolor','w', ...
    'callback','MPVL(''Update'')','position',[X3 Y1-eh*3 tw1 eh], ...
    'string',{'NACA a=0.8' 'Parabolic'},'visible','off');
Text3_S(1)=uicontrol('style','text','units','normalized','FontSize',10,...
    'FontWeight','bold','HorizontalAlignment','left','position', ...
    [X3 Y1-eh*2 tw1 eh],'string','Meanline Type:','visible','off');
Input3_S(2)=uicontrol('style','popupmenu','units','normalized', ...
    'FontSize',8,'FontWeight','bold','backgroundcolor','w', ...
    'callback','MPVL(''Update'')','position',...
    [X3+tw1+ew Y1-eh*3 tw1 eh],'string', ...
    {'NACA 65A010' 'Elliptical' 'Parabolic'},'visible','off');
Text3_S(2)=uicontrol('style','text','units','normalized', ...
    'FontSize',10,'FontWeight','bold','HorizontalAlignment','left', ...
    'position',[X3+tw1+ew Y1-eh*2 tw1 eh],'string', ...
    'Thickness Form:','visible','off');

% --------- Create UI components for inputs on the right of the GUI
for j = 1:length(XR0)
    Label_Row(j)=uicontrol('style','edit','units','normalized', ...
        'FontSize',10,'FontWeight','bold','position', ...
        [X3 Y2-h1*j ew eh],'string',XR0(j), ...
        'visible','off','enable','inactive');
    XCHD_in(j)=uicontrol('style','edit','units','normalized', ...
        'FontSize',10,'FontWeight','bold','backgroundcolor','w', ...
        'string',XCHD_def(j),'position',[X3+ew Y2-h1*j ew eh], ...
        'callback','MPVL(''Update'')','visible','off');
    XCD_in(j)=uicontrol('style','edit','units','normalized', ...
        'FontSize',10,'FontWeight','bold','backgroundcolor','w', ...
        'string',XCD_def(j),'position',[X3+ew*2 Y2-h1*j ew eh], ...
        'callback','MPVL(''Update'')','visible','off');
    XVA_in(j)=uicontrol('style','edit','units','normalized', ...
        'FontSize',10,'FontWeight','bold','backgroundcolor','w', ...
        'string',XVA_def(j),'position',[X3+ew*3 Y2-h1*j ew eh], ...
        'callback','MPVL(''Update'')','visible','off');
    XVT_in(j)=uicontrol('style','edit','units','normalized', ...
        'FontSize',10,'FontWeight','bold','backgroundcolor','w', ...
        'string',XVT_def(j),'position',[X3+ew*4 Y2-h1*j ew eh], ...
        'callback','MPVL(''Update'')','visible','off');
    f0oc_in(j)=uicontrol('style','edit','units','normalized', ...
        'FontSize',10,'FontWeight','bold','backgroundcolor','w', ...
        'string',f0oc_def(j),'position',[X3+ew*5 Y2-h1*j ew eh], ...
        'callback','MPVL(''Update'')','visible','off');
    t0oc_in(j)=uicontrol('style','edit','units','normalized', ...
        'FontSize',10,'FontWeight','bold','backgroundcolor','w', ...
        'string',t0oc_def(j),'position',[X3+ew*6 Y2-h1*j ew eh], ...
        'callback','MPVL(''Update'')','visible','off');
    skew_in(j)=uicontrol('style','edit','units','normalized', ...
        'FontSize',10,'FontWeight','bold','backgroundcolor','w', ...
        'string',skew_def(j),'position',[X3+ew*7 Y2-h1*j ew eh], ...
        'callback','MPVL(''Update'')','visible','off');
```

```
        rake_in(j)=uicontrol('style','edit','units','normalized', ...
            'FontSize',10,'FontWeight','bold','backgroundcolor','w', ...
            'string',rake_def(j),'position',[X3+ew*8 Y2-h1*j ew eh], ...
            'callback','MPVL(''Update'')','visible','off');
    end

    ColName={'r/R' 'c/D' 'Cd' 'Va/Vs' 'Vt/Vs' 'f0/c' 't0/c' 'Skew' 'Xs/D'};

    for i = 1:length(ColName)
        Label_Col(i)=uicontrol('style','edit','units','normalized', ...
            'FontSize',10,'FontWeight','bold','position', ...
            [X3+ew*(i-1) Y2 ew eh], 'string',ColName(i),...
            'enable','inactive','visible','off');
    end

    % ------------------------------------------- Create Pushbuttons
    Run_Parametric = uicontrol('style','pushbutton','units','normalized', ...
        'FontSize',10,'FontWeight','bold','position', [X3 Y4 ph*2 ph], ...
        'string','Run MPVL','callback', ...
        'MPVL(''Execute_Parametric'')','visible','off');
    Run_Single = uicontrol('style','pushbutton','units','normalized', ...
        'FontSize',10,'FontWeight','bold','position',[X3 Y4 ph*2 ph], ...
        'string','Run MPVL','callback', ...
        'MPVL(''Execute_Single'')','visible','off');
    New_Parametric = uicontrol('style','pushbutton','units','normalized', ...
        'FontSize',10,'FontWeight','bold','position',[X3+ph*2 Y4 ph*2 ph], ...
        'string','Try Again','callback','MPVL(''Update'')','visible','off');
    New_Single = uicontrol('style','pushbutton','units','normalized', ...
        'FontSize',10,'FontWeight','bold','position',[X3+ph*2 Y4 ph*2 ph], ...
        'string','Try Again','callback','MPVL(''Update'')','visible','off');
    Err_Parametric = uicontrol('style','pushbutton','units','normalized', ...
        'FontSize',8,'Fontweight','bold','position',[X3+ph*4 Y4 ph*3 ph], ...
        'Foregroundcolor','r','callback','MPVL(''Update'')','visible','off');
    Err_Single = uicontrol('style','pushbutton','units','normalized', ...
        'FontSize',8,'Fontweight','bold','position',[X3+ph*4 Y4 ph*3 ph], ...
        'Foregroundcolor','r','callback','MPVL(''Update'')','visible','off');

% ================================== INITIATE PARAMETRIC ANALYSIS GUI
elseif strcmp(action,'Initiate_Parametric')==1
    set(Welcome,'visible','off');
    set(Parametric,'enable','off');
    set(Single,'enable','on');
    Parametric_Flag = 1;

    if Single_Flag ==1      % If Single Design GUI is already launched
        set(Input1_S,'visible','off');
        set(Text1_S,'visible','off');
        set(Input2_S,'visible','off');
        set(Text2_S,'visible','off');
        set(Input3_S,'visible','off');
        set(Text3_S,'visible','off');
        set(f0oc_in,'visible','off');
        set(t0oc_in,'visible','off');
        set(skew_in,'visible','off');
        set(rake_in,'visible','off');
        set(Run_Single,'visible','off');
        set(New_Single,'visible','off');
```

```
        set(Err_Single,'visible','off');
    end

    set(Text1_P,'visible','on');
    set(Text2_P,'visible','on');
    set(NBLADE_in,'enable','on','visible','on');
    set(NBLADE_in(3),'Enable','off');
    set(N_in,'enable','on','visible','on');
    set(D_in,'enable','on','visible','on');
    set(Input1,'enable','on','visible','on');
    set(Text3_P,'visible','on');
    set(Hub_Flag,'enable','on','visible','on');
    set(Label_Col(1:5),'visible','on');
    set(Label_Col(6:9),'visible','off');
    set(Label_Row,'visible','on');
    set(XCHD_in,'enable','on','visible','on');
    set(XCD_in,'enable','on','visible','on');
    set(XVA_in,'enable','on','visible','on');
    set(XVT_in,'enable','on','visible','on');
    set(Run_Parametric,'enable','on','visible','on');
    Single_Flag = 0;

% ================================================ UPDATE INPUTS FIELDS
elseif strcmp(action,'Update')==1
    % ------------------------------------ Update Common Input Fields
    set(Input1,'enable','on');
    set(Hub_Flag,'enable','on');
    set(XCHD_in,'enable','on');
    set(XCD_in,'enable','on');
    set(XVA_in,'enable','on');
    set(XVT_in,'enable','on');
    get(Input1,'string');
    get(Hub_Flag,'value');
    get(XCHD_in,'string');
    get(XCD_in,'string');
    get(XVA_in,'string');
    get(XVT_in,'string');

    % ---------------------------------- Update Respective Input Fields
    if Parametric_Flag==1
        set(NBLADE_in(1:2),'enable','on');
        set(N_in,'enable','on');
        set(D_in,'enable','on');
        set(Run_Parametric,'enable','on');
        set(New_Parametric,'visible','off');
        set(Err_Parametric,'visible','off');
        get(NBLADE_in,'string');
        get(N_in,'string');
        get(D_in,'string');
    elseif Single_Flag==1
        set(Run_Single,'enable','on');
        set(Input1_S,'enable','on');
        set(Input2_S,'enable','on');
        set(Input3_S,'enable','on');
        set(f0oc_in,'enable','on');
        set(t0oc_in,'enable','on');
        set(skew_in,'enable','on');
```

```
        set(rake_in,'enable','on');
        set(New_Single,'visible','off');
        set(Err_Single,'visible','off');
        get(Input1_S,'string');
        get(Input2_S,'string');
        get(Input3_S,'string');
        get(f0oc_in,'string');
        get(t0oc_in,'string');
        get(skew_in,'string');
        get(rake_in,'string');
    end

% ============================== PERFORM PARAMETRIC ANALYSIS ALGORITHM
elseif strcmp(action,'Execute_Parametric')==1
    tic;      % start stopwatch

    % --- Disable components to prevent intervention during calculation
    set(NBLADE_in,'enable','off');
    set(N_in,'enable','off');
    set(D_in,'enable','off');
    set(Input1,'enable','off');
    set(Hub_Flag,'enable','off');
    set(XCHD_in,'enable','off');
    set(XCD_in,'enable','off');
    set(XVA_in,'enable','off');
    set(XVT_in,'enable','off');
    set(Run_Parametric,'enable','off');
    set(New_Parametric,'visible','on','enable','on');

    % ------------------------------ Close figures previously created
    if ishandle(Fig_P)~=0
        close(Fig_P);
    end
    if ishandle(Fig1_S)~=0
       close(Fig1_S);
    end
    if ishandle(Fig2_S)~=0
       close(Fig2_S);
    end
    if ishandle(Fig3_S)~=0
        close(Fig3_S);
    end

    % ------------------------------------------------ Define Variables
    NBLADE=str2double(get(NBLADE_in(1),'string')):1: ...
        str2double(get(NBLADE_in(2),'string'));
    N=str2double(get(N_in(1),'string')):str2double(get(N_in(3),'string')):...
        str2double(get(N_in(2),'string'));
    D=str2double(get(D_in(1),'string')):str2double(get(D_in(3),'string')):...
        str2double(get(D_in(2),'string'));

    % ------------------------ Set constraints on propeller parameters
    if (max(NBLADE>6)) || (min(NBLADE)<2)
        set(Err_Parametric,'visible','on','enable','on','string', ...
            '2<=Number of Blades=<6');
        set(New_Parametric,'enable','off');          beep;          return;
    elseif (str2double(get(N_in(1),'string'))> ...
```

```
        str2double(get(N_in(2),'string')))
        set(Err_Parametric,'visible','on','enable','on', ...
            'string','Check Propeller Speed');
        set(New_Parametric,'enable','off');          beep;          return;
    elseif (str2double(get(D_in(1),'string'))> ...
        str2double(get(D_in(2),'string')))
        set(Err_Parametric,'visible','on','enable','on', ...
            'string','Check Propeller Diameter');
        set(New_Parametric,'enable','off');          beep;          return;
    end

    % ------------------------- Derive Common Input Fields in a String
    string = str2double(get(Input1,'string'));
    THRUST = string(1);                V = string(2);
    Dhub = string(3);                  MT = string(4);
    ITER = string(5);                  RHV = string(6);
    NX = string(7);                    HR = string(8);
    HT = string(9);                    CRP = string(10);
    rho = string(11);                  Rhub = Dhub/2;
    IHUB = get(Hub_Flag,'value');
    XCHD0 = str2double(get(XCHD_in,'string'));
    XCD0 = str2double(get(XCD_in,'string'));
    XVA0 = str2double(get(XVA_in,'string'));
    XVT0 = str2double(get(XVT_in,'string'));

%------------------------------------------------ Perform Algorithm
    for k=1:length(NBLADE)
        for i=1:length(N)
            n(i) = N(i)/60;
            for j=1:length(D)
                R(j) = D(j)/2;
                ADVCO = V/(n(i)*D(j));
                CTDES = THRUST/(rho*V^2*pi*R(j)^2/2);
                XR1 = Rhub/R(j):(1-Rhub/R(j))/(NX-3):1;
                half1 = (XR1(1)+XR1(2))/2;
                half2 = (XR1(NX-3)+XR1(NX-2))/2;
                XR = cat(2,XR1(1),half1,XR1(2:NX-3),half2,XR1(NX-2));
                XCHD = pchip(XR0,XCHD0,XR);
                XCD = pchip(XR0,XCD0,XR);
                XVA = pchip(XR0,XVA0,XR);
                XVT = pchip(XR0,XVT0,XR);

                % ================================== Call Main Function
                [CT,CP,KT,KQ,WAKE,EFFY0,RC,G,VAC,VTC, UASTAR,UTSTAR, ...
                TANBC,TANBIC,CDC,CD,KTRY] = Main(MT,ITER,IHUB,RHV,NX, ...
                NBLADE(k),ADVCO,CTDES,HR,HT,CRP,XR,XCHD,XCD,XVA,XVT);
                % ========================================================
                EFFY(i,j,k) = EFFY0(end);
            end
        end
    end

    Fig_P=figure('units','normalized','position',[0.01 .06 .4 .3], ...
        'name','Efficiency','numbertitle','off');

    for i = 1:length(NBLADE)
        if length(NBLADE)==4
```

```
            subplot(2,2,i);
        else
            subplot(length(NBLADE),1,i);
        end

        plot(D,EFFY(:,:,i));
        str_suffix={' RPM'};

        for j=1:length(N)
            str_legend(j)=strcat(num2str(N(j)),str_suffix);
        end

        legend(str_legend,'location','southwest');          grid on;
        xlabel('Propeller Diameter (m)');
        ylabel('Efficiency');
        title_prefix = {'Number of Blades: '};
        title(strcat(title_prefix,num2str(NBLADE(i))))
    end

    set(New_Parametric,'enable','on');
    figure(Fig_Main);
    toc     % Stop stopwatch

% ========================================== INITIATE SINGLE DESIGN GUI
elseif strcmp(action,'Initiate_Single')==1
    set(Welcome,'visible','off');
    set(Parametric,'enable','on');
    set(Single,'enable','off');
    Single_Flag = 1;

    % If Parametric Analysis GUI is already launched
    if Parametric_Flag ==1
        set(Text1_P,'visible','off');
        set(Text2_P,'visible','off');
        set(NBLADE_in,'visible','off');
        set(N_in,'visible','off');
        set(D_in,'visible','off');
        set(Run_Parametric,'visible','off');
        set(New_Parametric,'visible','off');
        set(Err_Parametric,'visible','off');
    end

    set(Input1_S,'visible','on','enable','on');
    set(Text1_S,'visible','on');
    set(Input1,'visible','on','enable','on');
    set(Text3_P,'visible','on');
    set(Input2_S,'visible','on','enable','on');
    set(Text2_S,'visible','on');
    set(Hub_Flag,'visible','on','enable','on');
    set(Input3_S,'visible','on','enable','on');
    set(Text3_S,'visible','on');
    set(Label_Col,'visible','on');
    set(Label_Row,'visible','on');
    set(XCHD_in,'visible','on','enable','on');
    set(XCD_in,'visible','on','enable','on');
    set(XVA_in,'visible','on','enable','on');
    set(XVT_in,'visible','on','enable','on');
```

```
    set(f0oc_in,'visible','on','enable','on');
    set(t0oc_in,'visible','on','enable','on');
    set(skew_in,'visible','on','enable','on');
    set(rake_in,'visible','on','enable','on');
    set(Run_Single,'visible','on','enable','on');
    Parametric_Flag = 0;

% ==================================== PERFORM SINGLE DESIGN ALGORITHM
elseif strcmp(action,'Execute_Single')==1
    tic;        % start stopwatch
    set(Run_Single,'enable','off');
    set(Input1,'enable','off');
    set(Input1_S,'enable','off');
    set(Input2_S,'enable','off');
    set(Input3_S,'enable','off');
    set(Hub_Flag,'enable','off');
    set(XCHD_in,'enable','off');
    set(XCD_in,'enable','off');
    set(XVA_in,'enable','off');
    set(XVT_in,'enable','off');
    set(f0oc_in,'enable','off');
    set(t0oc_in,'enable','off');
    set(skew_in,'enable','off');
    set(rake_in,'enable','off');

    % ------------------------------ Close figures previously created
    if ishandle(Fig_P)~=0
        close(Fig_P);
    end
    if ishandle(Fig1_S)~=0
       close(Fig1_S);
    end
    if ishandle(Fig2_S)~=0
       close(Fig2_S);
    end
    if ishandle(Fig3_S)~=0
        close(Fig3_S);
    end

    % ------------------------------------------- Derive Input Fields
    string1 = str2double(get(Input1_S,'string'));
    string2 = str2double(get(Input1,'string'));
    string3 = str2double(get(Input2_S,'string'));
    NBLADE = string1(1);        N = string1(2);
    D = string1(3);             THRUST = string2(1);
    V = string2(2);             Dhub = string2(3);
    MT = string2(4);            ITER = string2(5);
    RHV = string2(6);           NX = string2(7);
    HR = string2(8);            HT = string2(9);
    CRP = string2(10);          rho = string2(11);
    H = string3(1);             dV = string3(2);
    AlphaI = string3(3);        NP = string3(4);
    R = D/2;                    Rhub = Dhub/2;
    n = N/60;
    ADVCO = V/(n*D);
    CTDES = THRUST/(rho*V^2*pi*R^2/2);
    IHUB = get(Hub_Flag,'Value');
```

```
Meanline = get(Input3_S(1),'Value');
Thickness = get(Input3_S(2),'Value');
XCHD0 = str2double(get(XCHD_in,'String'));
XCD0 = str2double(get(XCD_in,'String'));
XVA0 = str2double(get(XVA_in,'String'));
XVT0 = str2double(get(XVT_in,'String'));
f0oc0 = str2double(get(f0oc_in,'String'));
t0oc0 = str2double(get(t0oc_in,'String'));
skew0 = str2double(get(skew_in,'String'));
rake0 = str2double(get(rake_in,'String'));

if Dhub/D < .15      % Warn if hub diameter is too small
    set(Err_Single,'visible','on','enable','on','string','Dhub/D>= 15%');
    return;
end

% ------------------------------------------- Make MPVL_Input.txt
XR1 = Rhub/R:(1-Rhub/R)/(NX-3):1;
half1 = (XR1(1)+XR1(2))/2;
half2 = (XR1(NX-3)+XR1(NX-2))/2;
XR = [XR1(1) half1 XR1(2:NX-3) half2 XR1(NX-2)];
XCHD = pchip(XR0,XCHD0,XR);
XCD = pchip(XR0,XCD0,XR);
XVA = pchip(XR0,XVA0,XR);
XVT = pchip(XR0,XVT0,XR);
Flag1 = datestr(now,31);
fid = fopen('MPVL_Input.txt','w');
fprintf(fid,'%s\tMPVL_Input.txt\n',Flag1);
fprintf(fid,'%.0f \t\tNumber of Vortex Panels over the Radius\n',MT);
fprintf(fid,'%.0f \t\tMax. Iterations in Wake Alignment\n',ITER);
fprintf(fid,'%.0f \t\tHub Image Flag: 1=YES, 0=NO\n',IHUB);
fprintf(fid,'%.1f \tHub Vortex Radius/Hub Radius\n',RHV);
fprintf(fid,'%.0f \t\tNumber of Input Radii\n',NX);
fprintf(fid,'%.0f \t\tNumber of Blades\n',NBLADE);
fprintf(fid,'%.3f \tAdvance Coef., J, Based on Ship Speed\n',ADVCO);
fprintf(fid,'%.3f \tDesired Thrust Coef., Ct\n',CTDES);
fprintf(fid,'%.0f \t\tHub Unloading Factor: 0=optimum\n',HR);
fprintf(fid,'%.0f \t\tTip Unloading Factor: 1=Reduced Loading\n',HT);
fprintf(fid,'%.0f \t\tSwirl Cencellation Factor:1=No Cancellation\n',CRP);
fprintf(fid,'r/R  \t  C/D  \t   XCD\t    Va/Vs  Vt/Vs\n');
for i = 1:NX
    fprintf(fid,'%6.5f  %6.5f  %6.5f  %6.2f  %6.4f\n', ...
        XR(i),XCHD(i),XCD(i),XVA(i),XVT(i));
end
fclose(fid);

% ============================================== Call Main Function
[CT,CP,KT,KQ,WAKE,EFFY,RC,G,VAC,VTC,UASTAR,UTSTAR,TANBC,TANBIC,CDC,CD,...
    KTRY]= Main(MT,ITER,IHUB,RHV,NX,NBLADE,ADVCO,CTDES,HR,HT,CRP,XR, ...
    XCHD,XCD,XVA,XVT);

% ---------------------------------------- Create Graphical Reports
Fig1_S = figure('units','normalized','position',[.01 .06 .4 .3],'name',...
    'Graphical Report','numbertitle','off');
subplot(2,2,1);
plot(RC,G);
xlabel('r/R');          ylabel('Non-Dimensional Circulation');
```

```
grid on;
TitleString=strcat('J=',num2str(ADVCO,'%10.3f'),'; Ct=', ...
    num2str(CT(KTRY),'%10.3f'),'; Kt=',num2str(KT(KTRY),'%10.3f'), ...
    '; Kq=',num2str(KQ(KTRY),'%10.3f'),'; \eta=', ...
    num2str(EFFY(KTRY),'%10.3f'));
title(TitleString);
subplot(2,2,2);
plot(RC,VAC,'-b',RC,VTC,'--b',RC,UASTAR,'-.r',RC,UTSTAR,':r');
xlabel('r/R');          legend('Va/Vs','Vt/Vs','Ua*/Vs','Ut*/Vs');
grid on;
subplot(2,2,3);
plot(RC,TANBC,'--b',RC,TANBIC,'-r');
xlabel('r/R');          ylabel('Degrees');      grid on;
legend('Beta','BetaI');
subplot(2,2,4);
plot(RC,CDC);
xlabel('r/R');          ylabel('c/D');          grid on;

% ------------------------------ Propeller Performance Calculation
w = 2*pi*n;          % Angular velocity w
for k = 1:MT
    Vstar(k) = sqrt((VAC(k)+UASTAR(k))^2+(w*R*RC(k)+VTC(k)+UTSTAR(k))^2);
    Gamma(k) = G(k)*2*pi*R*V;
    Cl(k) = 2*Gamma(k) / (Vstar(k)*CDC(k)*D);
    dBetai(k) = atand((tand(TANBIC(k))*w*RC(k)*R+dV)/(w*RC(k)*R))...
              -atand((tand(TANBIC(k))*w*RC(k)*R-dV)/(w*RC(k)*R));
    %Cavitation Number
    Sigma(k) = (101000+rho*9.81*(H-RC(k)*R)-2500)/(rho*Vstar(k)^2/2);
end

% Scale camber ratio with lift coefficient
f0oc = pchip(XR0,f0oc0,RC).*Cl;
t0oc = pchip(XR0,t0oc0,RC);
fid = fopen('Performance.txt','w');
fprintf(fid,'\t\t\t\t\t\tPerformance.txt\n');
fprintf(fid,'\t\t\t\t\tPropeller Performance Table\n');
fprintf(fid,' r/R\t\tV*\t beta\t betai\t  Gamma\t\tCl\t Sigma\tdBetai\n');
for k = 1:MT
  fprintf(fid,'%.3f\t %.3f\t %.2f\t %.2f\t %.4f\t %.3f\t %.3f\t %.2f\n'...
    ,RC(k),Vstar(k),TANBC(k),TANBIC(k),Gamma(k),Cl(k),Sigma(k),dBetai(k));
end
fclose(fid);

% ------------------------------------------ Geometry Calculation
skew = pchip(XR0,skew0,RC);
rake = pchip(XR0,rake0,RC);
fid = fopen('Geometry.txt','w');
fprintf(fid,'\t\t\tGeometry.txt\n');
fprintf(fid,'\t\tPropeller Geometry Table\n\n');
fprintf(fid,'Propeller Diameter = %.1f m\n',D);
fprintf(fid,'Number of Blades = %.0f\n',NBLADE);
fprintf(fid,'Propeller Speed= %.0f RPM\n',N);
fprintf(fid,'Propeller Hub Diameter = %.2f m\n',Dhub);

if Meanline==1
    fprintf(fid,'Meanline Type: NACA a=0.8\n');
elseif Meanline==2
```

```
        fprintf(fid,'Meanline Type: Parabolic\n');
    end

    if Thickness==1
        fprintf(fid,'Thickness Type: NACA 65A010\n\n');
    elseif Thickness==2
        fprintf(fid,'Thickness Type: Elliptical\n\n');
    elseif Thickness==3
        fprintf(fid,'Thickness Type: Parabolic\n\n');
    end

    fprintf(fid,' r/R\t P/D\t Skew\t Xs/D\t  c/D\t  f0/c\t  t0/c\n');

    for i = 1:MT
        ThetaP(i) = TANBIC(i) + AlphaI;                  % Pitch angle
        PitchOD(i) = tand(ThetaP(i))*pi*RC(i);           % P/D
        fprintf(fid, '%.3f\t %.2f\t %.1f\t %.3f\t %.3f\t %.4f\t %.4f\n'...
        ,RC(i),PitchOD(i),skew(i),rake(i),CDC(i),f0oc(i),t0oc(i));
    end

    fclose(fid);

    % ------------------------------------------------------ BASIC SHAPE
    c = CDC.*D;
    r = RC.*R;
    theta = 0:360/NBLADE:360;   % Angles between blades

    for i = 1:MT
        for j = 1:NP
            x1(i,j) = c(i)/2-c(i)/(NP-1)*(j-1);
            station(1,j) = 1/(NP-1)*(j-1);               % For NACA foil
            z1(i,j,1) = sqrt(r(i)^2-x1(i,j)^2);
        end
    end

% ------------------------------------------------- MEANLINE & THICKNESS
    x = [0 .5 .75 1.25 2.5 5 7.5 10 15 20 25 30 35 40 45 50 55 60 65 70 75 ...
         80 85 90 95 100]./100;

    if Meanline==1          % NACA a=0.8 meanline is chosen
        foc = [0 .287 .404 .616 1.077 1.841 2.483 3.043 3.985 4.748 5.367...
               5.863 6.248 6.528 6.709 6.79 6.77 6.644 6.405 6.037 5.514...
               4.771 3.683 2.435 1.163 0]./100;
        fscale = f0oc./max(foc);        % Scale for camber
        dfdx0 = [.48535 .44925 .40359 .34104 .27718 .23868 .21050 .16892...
            .13734 .11101 .08775 .06634 .04601 .02613 .00620 -.01433...
            -.03611 -.06010 -.08790 -.12311 -.18412 -.23921 -.25583...
            -.24904 -.20385];
        for i = 1:MT
            for j = 1:NP
                f(i,:) = pchip(x,foc.*fscale(i).*c(i),station);
                dfdx(i,:) = pchip(x(2:end),dfdx0,station);
            end
        end
    elseif Meanline==2      % Parabolic meanline is chosen
        for i = 1:MT
            for j = 1:NP
```

```
            f(i,j) = f0oc(i)*c(i)*(1-(2*x1(i,j)/c(i))^2);
            dfdx(i,j) = -8*f0oc(i)*x1(i,j)/c(i);
        end
    end
end

if Thickness==1          % NACA 65A010 thickness form is chosen
    toc_65 = [0 .765 .928 1.183 1.623 2.182 2.65 3.04 3.658 4.127...
              4.483 4.742 4.912 4.995 4.983 4.863 4.632 4.304 3.899...
              3.432 2.912 2.352 1.771 1.188 .604 .021]./100;
    tscale = t0oc./2./max(toc_65);      % Scale for thickness
    for i = 1:MT
        for j = 1:NP
            t(i,:) = pchip(x,toc_65.*tscale(i).*c(i),station);
        end
    end
elseif Thickness==2      % Elliptical thickness form is chosen
    for i = 1:MT
        for j = 1:NP
            t(i,j) = t0oc(i)*c(i)*real(sqrt(1-(2*x1(i,j)/c(i))^2));
        end
    end
elseif Thickness==3      % Parabolic thickness form is chosen
    for i = 1:MT
        for j = 1:NP
            t(i,j) = t0oc(i)*c(i)*(1-(2*x1(i,j)/c(i))^2);
        end
    end
end

% ------------------------------------------------ CAMBER & THICKNESS
for i = 1:MT
    for j = 1:NP
        xu(i,j) = x1(i,j)+t(i,j)*sin(atan(dfdx(i,j)));
        xl(i,j) = x1(i,j)-t(i,j)*sin(atan(dfdx(i,j)));
        yu(i,j) = f(i,j)+t(i,j)*cos(atan(dfdx(i,j)));
        yl(i,j) = f(i,j)-t(i,j)*cos(atan(dfdx(i,j)));
        if (isreal(yu(i,j))==0)||(isreal(yl(i,j))==0)
            if Single_Flag==1
                set(Err_Single,'visible','on','enable','on', ...
                    'string','Error in MPVL.');
                if ishandle(Fig1_S)~=0
                    close(Fig1_S);
                end
                return;
            end
        end
    end
end

% ------------------------------------------------ PITCH, SKEW & RAKE
yrake = rake.*D;  % Rake: translation in y direction
for i = 1:MT
    for j = 1:NP
        xup(i,j,1)=xu(i,j)*cosd(ThetaP(i))-yu(i,j)*sind(ThetaP(i));
        xlp(i,j,1)=xl(i,j)*cosd(ThetaP(i))-yl(i,j)*sind(ThetaP(i));
        yup(i,j,1)=xu(i,j)*sind(ThetaP(i))+yu(i,j)*cosd(ThetaP(i));
```

```
        ylp(i,j,1)=xl(i,j)*sind(ThetaP(i))+yl(i,j)*cosd(ThetaP(i));
        zlp(i,j,1)=zl(i,j);
        xus(i,j,1)=xup(i,j)*cosd(skew(i))-zlp(i,j)*sind(skew(i));
        xls(i,j,1)=xlp(i,j)*cosd(skew(i))-zlp(i,j)*sind(skew(i));
        yus(i,j,1)=yup(i,j);
        yls(i,j,1)=ylp(i,j);
        zls(i,j,1)=xl(i,j)*sind(skew(i))+zlp(i,j)*cosd(skew(i));
        xur(i,j,1)=xus(i,j);
        xlr(i,j,1)=xls(i,j);
        yur(i,j,1)=yus(i,j)-yrake(i);
        ylr(i,j,1)=yls(i,j)-yrake(i);
        zlr(i,j,1)=zls(i,j,1);
        for k = 2:length(theta)-1
            xur(i,j,k)= xur(i,j,1)*cosd(theta(k))- ...
                        zlr(i,j,1)*sind(theta(k));
            xlr(i,j,k)= xlr(i,j,1)*cosd(theta(k))- ...
                        zlr(i,j,1)*sind(theta(k));
            yur(i,j,k) = yur(i,j,1);
            ylr(i,j,k) = ylr(i,j,1);
            zlr(i,j,k) = xur(i,j,1)*sind(theta(k))+ ...
                         zlr(i,j,1)*cosd(theta(k));
        end
    end
end

% ---------------------- Create Figure for 2D Propeller Blade Image
Fig2_S = figure('units','normalized','position',[0.31 .06 .4 .3], ...
    'name','Blade Image','numbertitle','off');
style=['r' 'g' 'b' 'm' 'k'];
str_prefix = {'r/R = '};
flag=1;

for i = 1:ceil(MT/5):MT
    plot(xur(i,:,1),yur(i,:,1),style(flag));
    str_legend(flag)=strcat(str_prefix,num2str(RC(i)));
    hold on;
    flag = flag+1;
end

flag=1;

for i = 1:ceil(MT/5):MT
    plot(xlr(i,:,1),ylr(i,:,1),style(flag));
    hold on;
    flag = flag+1;
end

legend(str_legend,'location','northwest');
axis equal;     grid on;
xlabel('X (m)');    ylabel('Y (m)');    hold off;

% ------------------------------------ Create 3D Propeller Image
Fig3_S = figure('units','normalized','position',[.61 .06 .4 .3],...
       'name','Propeller Image','numbertitle','off');

for k = 1:NBLADE
    surf(xur(:,:,k),yur(:,:,k),zlr(:,:,k));         hold on;
```

```
        surf(xlr(:,:,k),ylr(:,:,k),zlr(:,:,k));          hold on;
    end

    tick = 0:15:90;
    [xh0,yh0,zh0] = cylinder(Rhub*sind(tick),50);
    surf(yh0,zh0.*.3+min(ylp(1,:,1))-.3,xh0);
    [xh1,yh1,zh1] = cylinder(Rhub,50);
    surf(yh1, zh1+min(ylp(1,:,1)), xh1);
    hold off;       colormap gray;          grid on;        axis equal;
    xlabel('X');    ylabel('Y');            zlabel('Z');
    set(New_Single,'visible','on','enable','on');
    figure(Fig_Main);
    toc
end     % ================================================ Do not delete

% ================================================== Main Function Code
function [CT,CP,KT,KQ,WAKE,EFFY,RC,G,VAC,VTC,UASTAR,UTSTAR,TANBC,TANBIC, ...
    CDC,CD,KTRY]= Main(MT,ITER,IHUB,RHV,NX,NBLADE,ADVCO,CTDES,HR, ...
    HT,CRP,XR,XCHD,XCD,XVA,XVT)

    global Parametric_Flag Single_Flag

    % ---------------------------------------- FORTRAN Function VOLWK
    YW = XR.*XVA;
    YDX = trapz(XR,YW);
    WAKE = 2*YDX/(1-XR(1)^2);
    % ------------------------------------------------- End of VOLWK

    XRC = 1-sqrt(1-XR);
    DEL = pi/(2*MT);            % Compute cosine spaced vortex radii
    HRR = 0.5*(XR(NX)-XR(1));
    for i=1:MT+1
        RV(i) = XR(1)+HRR*(1-cos(2*(i-1)*DEL));
    end
    VAV = pchip(XR,XVA,RV);
    VTV = pchip(XR,XVT,RV);
    TANBV = VAV./((pi.*RV./ADVCO)+VTV);
    VBAV = VTV.*TANBV./VAV;

   % Cosine spaced control point radii:Evaluate c/D,Va,Vt,tanB,Cd,Vt*,tanB/Va
    for i=1:MT
        RC(i) = XR(1)+HRR*(1-cos((2*i-1)*DEL));
        RCWG(i) = 1-sqrt(1-RC(i));
    end

    CDC = pchip(XRC,XCHD,RCWG);
    CD = pchip(XR,XCD,RC);
    VAC = pchip(XR,XVA,RC);
    VTC = pchip(XR,XVT,RC);
    TANBC = VAC./(pi.*RC./ADVCO+VTC);
    VBAC = VTC.*TANBC./VAC;

    % First estimation of tanBi based on 90% of actuator disk efficiency
    EDISK = 1.8/(1+sqrt(1+CTDES/WAKE^2));
    TANBXV = TANBV.*sqrt(WAKE./(VAV-VBAV))/EDISK;
    TANBXC = TANBC.*sqrt(WAKE./(VAC-VBAC))/EDISK;
```

```
% Unload hub and tip as specified by HR and HT
RM = (XR(1)+XR(NX))/2;
for i=1:MT+1
    if RV(i)<RM
        HRF=HR;
    else
        HRF=HT;
    end
    DTANB = HRF*(TANBXV(i)-TANBV(i))*((RV(i)-RM)/(XR(1)-RM))^2;
    TANBXV(i) = TANBXV(i)-DTANB;
end

for i=1:MT
    if RC(i)<RM
        HRF=HR;
    else
        HRF=HT;
    end
    DTANB = HRF*(TANBXC(i)-TANBC(i))*((RC(i)-RM)/(XR(1)-RM))^2;
    TANBXC(i) = TANBXC(i)-DTANB;
end

% Iterations to scale tanBi to get desired value of thrust coefficient
for KTRY=1:ITER
    if KTRY==1
        T(KTRY) = 1;
    elseif KTRY==2
        T(KTRY) = 1+(CTDES-CT(1))/(5*CTDES);
    elseif KTRY>2
        if CT(KTRY-1)-CT(KTRY-2)==0
            break
        else
            T(KTRY)=T(KTRY-1)+(T(KTRY-1)-T(KTRY-2))*(CTDES-CT(KTRY-1))/...
                (CT(KTRY-1)-CT(KTRY-2));
        end
    end
    TANBIV = T(KTRY).*TANBXV;
    TANBIC = T(KTRY).*TANBXC;

    % Compute axial and tangential horseshoe influence coefficients
    for i=1:MT
        RCW = RC(i);
        for j = 1:MT+1
            RVW = RV(j);
            % induction of trailing vortices shed at RV(N)
            TANBIW = TANBIV(j);
            [UAIF,UTIF]=WRENCH(NBLADE,TANBIW,RCW,RVW);
            if (isnan(UAIF)==1)||(isnan(UTIF)==1)
                if Single_Flag==1
                   set(Err_Single,'visible','on','enable','on', ...
                       'string','Error in MPVL.');
                end
                return;
            end
            UAW(j) = -UAIF/(2*(RC(i)-RV(j)));
            UTIF = UTIF*CRP;
            UTW(j) = UTIF/(2*(RC(i)-RV(j)));
```

```
            % Induction of corresponding hub-image trailing vorticies
            if IHUB==1
                RVW = XR(1)^2/RV(j);
                TANBIW = TANBIV(1)*RV(1)/RVW;
                [UAIF,UTIF]=WRENCH(NBLADE,TANBIW,RCW,RVW);
                if (isnan(UAIF)==1)||(isnan(UTIF)==1)
                    if Single_Flag==1
                       set(Err_Single,'visible','on','enable','on',...
                           'string','Error in MPVL.');
                    end
                    return;
                end
                UAW(j) = UAW(j)+UAIF/(2*(RC(i)-RVW));
                UTIF = UTIF*CRP;
                UTW(j) = UTW(j)-UTIF/(2*(RC(i)-RVW));
            end
        end
        % Final step in building influence functions
        for k=1:MT
            UAHIF(i,k) = UAW(k+1)-UAW(k);
            UTHIF(i,k) = UTW(k+1)-UTW(k);
        end
    end

    % Solve simutaneous equations for circulation strengths G(i)
    for m=1:MT
        B(m) = VAC(m)*((TANBIC(m)/TANBC(m))-1);
        for n=1:MT
            A(m,n) = UAHIF(m,n)-UTHIF(m,n)*TANBIC(m);
        end
    end

    % ================================== FORTRAN Subroutine SIMEQN
    NEQ = length(B);
    IERR = 1;
% Find |maximum| element in each row and exit if a zero row is detected
    for i=1:NEQ
        IPIVOT(i) = i;
        ROWMAX = 0;
        for j=1:NEQ
            ROWMAX = max(ROWMAX, abs(A(i,j)));
        end
        if ROWMAX==0
            fprintf('Matrix is Singular-1.\n')
            G = NaN;     % Must return something to avoid error warning
            if Single_Flag==1
                set(Err_Single,'visible','on','enable','on', ...
                    'string','Error in MPVL.');
            end
            return;
        end
        D(i) = ROWMAX;
    end
    NM1=NEQ-1;
```

```
if NM1>0        % Otherwise special case of one equation
    for k=1:NM1
        j = k;
        KP1 = k+1;
        IP = IPIVOT(k);
        COLMAX = abs(A(IP,k))/D(IP);
        for i=KP1:NEQ
            IP = IPIVOT(i);
            AWIKOV = abs(A(IP,k))/D(IP);
            if AWIKOV>COLMAX
                COLMAX = AWIKOV;
                j=i;
            end
        end

        if COLMAX==0
            fprintf('Matrix is Singular-2.\n')
            G = NaN;    % Must return something to avoid error warning
            if Single_Flag==1
                set(Err_Single,'visible','on','enable','on', ...
                    'string','Error in MPVL.');
            end
            return;
        end
        IPK = IPIVOT(j);
        IPIVOT(j) = IPIVOT(k);
        IPIVOT(k) = IPK;
        for i=KP1:NEQ
            IP = IPIVOT(i);
            A(IP,k) = A(IP,k)/A(IPK,k);
            RATIO = -A(IP,k);
            for j=KP1:NEQ
                A(IP,j) = RATIO*A(IPK,j)+A(IP,j);
            end
        end
    end

    if A(IP,NEQ)==0
        fprintf('Matrix is Singular-3.\n')
        G = NaN;  % Must return something to avoid error warning
        if Single_Flag==1
            set(Err_Single,'visible','on','enable','on', ...
                'string','Error in MPVL.');
        end
        return;
    end
end
IERR = 0;       % Matrix survived singular test

if NEQ==1       % Back substitute to obtain solution (G)
    G(1) = B(1)/A(1,1);
else
    IP = IPIVOT(1);
    G(1) = B(IP);
    for k=2:NEQ
        IP = IPIVOT(k);
        KM1 = k-1;
```

```
        SUMM = 0;
        for j=1:KM1
            SUMM = A(IP,j)*G(j)+SUMM;
        end
        G(k) = B(IP)-SUMM;
    end
    G(NEQ) = G(NEQ)/A(IP,NEQ);
    k = NEQ;
    for NP1MK=2:NEQ
        KP1 = k;
        k = k-1;
        IP = IPIVOT(k);
        SUMM = 0;
        for j=KP1:NEQ
            SUMM = A(IP,j)*G(j)+SUMM;
        end
        G(k) = (G(k)-SUMM)/A(IP,k);
    end
end
% ================================================== End of SIMEQN

if IERR==1  % Matrix is singular
    if Single_Flag==1
        set(Err_Single,'visible','on','enable','on','string', ...
            'Error in MPVL.');
    end
    return;
end

% Evaluate the induced velocities from the circulation G(i)
for p=1:MT
    UASTAR(p) = 0;
    UTSTAR(p) = 0;
    for q=1:MT
       UASTAR(p) = UASTAR(p)+G(q)*UAHIF(p,q);
       UTSTAR(p) = UTSTAR(p)+G(q)*UTHIF(p,q);
    end
end

% ================================== FORTRAN Subroutine FORCES
LD = 0;     % Default: Input is Cd, not L/D
if CD>1
    LD = 1;  % CD> 1 means the input is L/D
end
CT(KTRY) = 0;
CQ(KTRY) = 0;

for m=1:MT
    DR = RV(m+1)-RV(m);
    VTSTAR = VAC(m)/TANBC(m)+UTSTAR(m);
    VASTAR = VAC(m)+UASTAR(m);
    VSTAR = sqrt(VTSTAR^2+VASTAR^2);
    if LD==0
        DVISC = (VSTAR^2*CDC(m)*CD(m))/(2*pi);
    else
        FKJ = VSTAR*G(m);
        DVISC = FKJ/CD(m);
```

```
            end
            CT(KTRY) = CT(KTRY)+(VTSTAR*G(m)-DVISC*VASTAR/VSTAR)*DR;
            CQ(KTRY)=CQ(KTRY)+(VASTAR*G(m)+DVISC*VTSTAR/VSTAR)*RC(m)*DR;
        end
        if IHUB~=0
            CTH = .5*(log(1/RHV)+3)*(NBLADE*G(1))^2;
        else
            CTH = 0;
        end
        CT(KTRY) = CT(KTRY)*4*NBLADE-CTH;
        CQ(KTRY) = CQ(KTRY)*2*NBLADE;
        CP(KTRY) = CQ(KTRY)*2*pi/ADVCO;
        KT(KTRY) = CT(KTRY)*ADVCO^2*pi/8;
        KQ(KTRY) = CQ(KTRY)*ADVCO^2*pi/8;
        EFFY(KTRY) = CT(KTRY)*WAKE/CP(KTRY);

        for i=1:length(RC)
            if (isreal(TANBIC(i))==0)||(isreal(EFFY(KTRY))==0) ...
                ||(EFFY(KTRY)<=0)
                TANBIC(i) = NaN;
                EFFY(KTRY) = NaN;
            end
        end
        % ============================================== End of FORCES

        if abs(CT(KTRY)-CTDES)<(5e-6)
            break
        end
    end             % For KTRY=1:ITER.  Do not delete

    if IERR==1      % Stop run if matrix is singular
       fprintf('Matrix is Singular. Run Terminated.\n');
       return;
    else
        if Single_Flag==1
            TANBC = atand(TANBC);
            TANBIC = atand(TANBIC);
            fid = fopen('MPVL_Output.txt','w');
            fprintf(fid,'\t\t\t\t\t MPVL_Output.txt\n');
            fprintf(fid,'\t\t\t\t\tMPVL Output Table\n');
            fprintf(fid,'Ct= % 5.4f\n' ,CT(KTRY));
            fprintf(fid,'Cp= % 5.4f\n' ,CP(KTRY));
            fprintf(fid,'Kt= % 5.4f\n' ,KT(KTRY));
            fprintf(fid,'Kq= % 5.4f\n' ,KQ(KTRY));
            fprintf(fid,'Va/Vs= % 5.4f\n' ,WAKE);
            fprintf(fid,'Efficiency= %5.4f\n' ,EFFY(KTRY));
            fprintf(fid,' r/R\t  \tG\t\t  Va\t  Vt\t  Ua\t  \tUt\t...
                \tBeta\tBetaI\t c/D\t  Cd\t\n');
            for i = 1:length(RC)
            fprintf(fid,'%5.5f  %5.6f  %5.5f  %5.4f  %5.5f  %5.5f  %5.3f...
                %5.3f  %5.5f  %5.5f\n',RC(i),G(i),VAC(i),VTC(i), ...
                UASTAR(i),UTSTAR(i), TANBC(i),TANBIC(i),CDC(i),CD(i));
            end
            fclose(fid);
        end
    end
% ============================================== End of Main Function
```

```
% ========================================== FORTRAN Subroutine WRENCH
function [UAIF,UTIF]=WRENCH(NBLADE,TANBIW,RCW,RVW)

    if NBLADE>20    % Return infinite blade result if NBLADE>20
        if RCW>RVW
            UAIF = 0;
            UTIF = NBLADE*(RCW-RVW)/RCW;
        else
            UAIF = -NBLADE*(RCW-RVW)/(RVW*TANBIW);
            UTIF = 0;
        end
        return;
    end

    XG = 1/TANBIW;    % End of infinite blade patch
    ETA = RVW/RCW;
    H = XG/ETA;
    XS = 1+H^2;
    TW = sqrt(XS);
    V = 1+XG^2;
    W = sqrt(V);
    AE = TW-W;
    U = exp(AE);
    R = (((TW-1)/H*(XG/(W-1)))*U)^NBLADE;
    XX = (1/(2*NBLADE*XG))*((V/XS)^0.25);
    Y = ((9*XG^2)+2)/(V^1.5)+((3*H^2-2)/(XS^1.5));
    Z = 1/(24*NBLADE)*Y;

    if H>=XG
        AF = 1+1/(R-1);
        if AF==0
            UAIF = NaN;
            UTIF = NaN;
            return;
        else
            AA = XX*(1/(R-1)-Z*log(AF));
            UAIF = 2*NBLADE^2*XG*H*(1-ETA)*AA;
            UTIF = NBLADE*(1-ETA)*(1+2*NBLADE*XG*AA);
        end
    else
        if R>1e-12
            RATIO = 1/(1/R-1);
        else
            RATIO = 0;
        end
        AG = 1+RATIO;
        if AG==0
            UAIF = NaN;
            UTIF = NaN;
            return;
        else
            AB = -XX*(RATIO+Z*log(AG));
            UAIF = NBLADE*XG*(1-1/ETA)*(1-2*NBLADE*XG*AB);
            UTIF = 2*NBLADE^2*XG*(1-ETA)*AB;
        end
    end
```