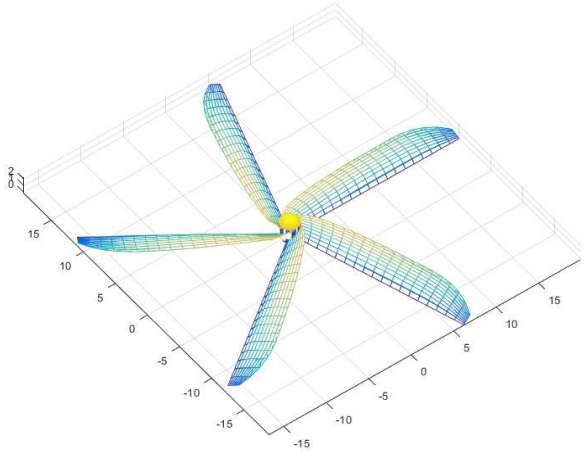
Modelling 2 – (MENG 21712)
Propeller Parametric Design

Shiraz Wasim and Jules Boissier

December 2nd, 2016

Abstract

In this project, we set out to write a program using Matlab that would generate a propeller. The initial parameters for the propeller were user defined in a graphical user interface. An STL file was also generated to be able to 3D print the final design. The code built the propeller in stages from the hub to the lower and upper surfaces of the blade to the nose. The blade is formed with aerofoil cross-sections which were designed using NACA profiles. This resulted in some major challenges that are highlighted in this report. The report also includes pseudocode as well as an explanation of how to run the code. This project enhanced our ability to write code and helped us gain an appreciation of the importance of parametric design.



Contents

1 Introduction

Parametric design is the generation of geometry using initial parameters and the relations they keep with each other. Using variables and algorithms, a hierarchy of mathematical and geometric relations can be created that allow the production of a certain design. The variability of these parameters allow testing of a variety of solutions. [1] Propellers, thus, become an ideal object to parameterise. The variable geometry of propellers results in several different applications. Furthermore, minor disparities in the geometry can affect the performance. Therefore, several variations are tested to determine the most efficient geometry as well as the ideal dimensions for specific applications.

MATLAB was used to enable the modelling of the propeller designs, which are made up of a number of blades, a hub and a nose. Using blades with aerofoil cross-sections, a propeller creates pressure differences between the front and trailing edge to transform rotational motion into forwards thrust. To create the cross-section, a NACA profile was used; these designs cause little drag due to their low angle of attack. The flexibility of the code allows the user to create a fan for the cold section of a plane turbine; this can be done by increasing the number of blades. A Graphical User Interface (GUI), made using 'Appdesigner,' allows the user to input parameters for the design. There are also options to create a random or default propeller.

Propellers are important engineering objects as they have a variety of applications. As a result, an attempt to create a program to generate propeller has been attempted before. One program called 'Openprop' is a "Propeller Vortex Lattice Lifting Line Program" which can model anything ranging from a propeller to tidal turbines. Their program considers creating propellers using lattice representation where each blade is represented by a set of concentrated radial and helical vortex lines. Another method is to represent the blade using a continuous vortex sheet representation which consists of both bound and free vortices. [2] This, however, was considered complex so we decided to come up with our own method.

The blades of the propeller modelled by the MATLAB code are aerofoils with a NACA profile. To define this profile, a few key variables had to be specified. These are shown in Figure 1 and are the maximum thickness, maximum camber, location of maximum camber and chord length. From these parameters we generated different variations of the aerofoil shape.

An STL file is generated. This means that the propeller surface is sectioned into a series of triangles linked together to recreate the meshed plot. This will convert the propeller design into a format which can be 3D printed. It's useful to note that converting into an STL file isn't enough to 3D print our object, we would first have to put that file through a slicer program which splits the program into horizontal layers, which are deposited one atop the other to print the file.

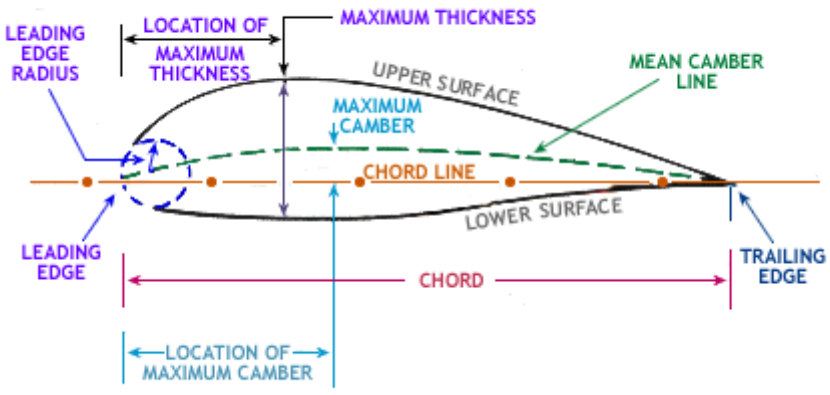


Figure 1 : Aerofoil cross-section showing important design parameters

2 Pseudocode

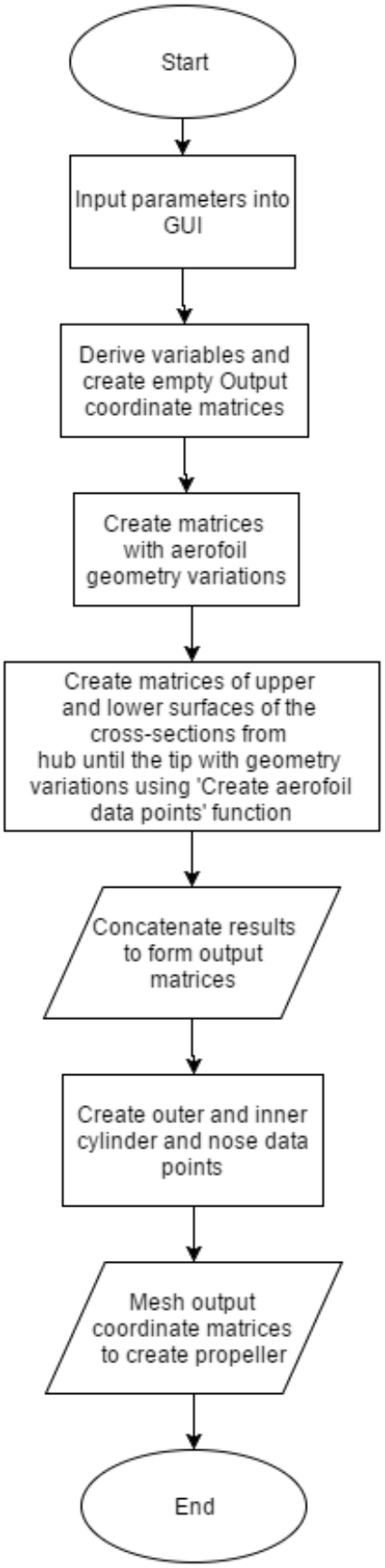


Figure 3 : Pseudocode for the main script of the code

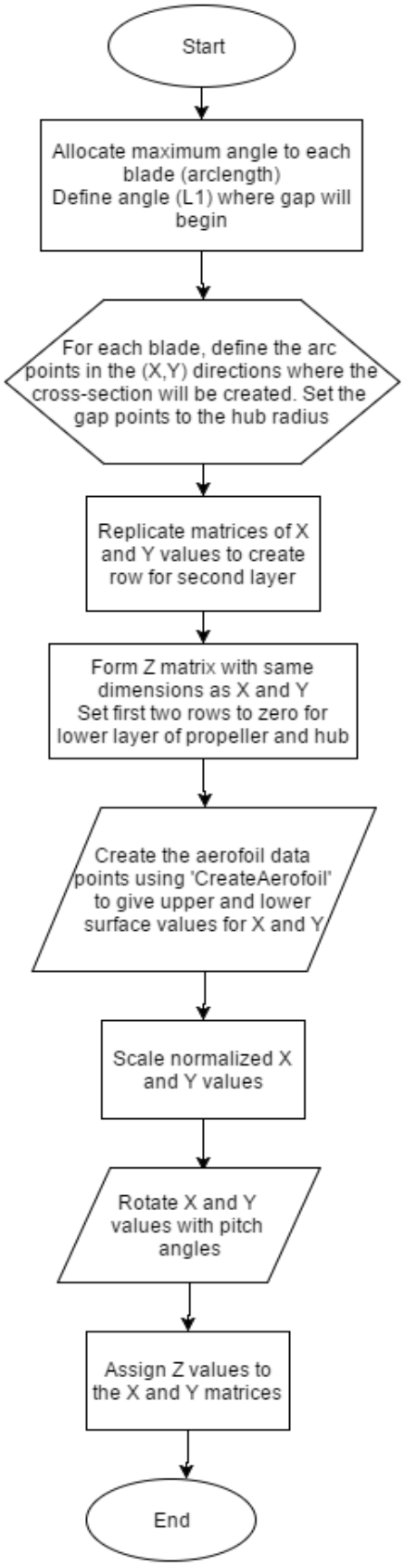


Figure 2 : Pseudocode showing the creation of the aerofoil data points

3 Major Challenges

When writing code, there are often several challenges. In the following section, discussion is made on aspects of the program that were difficult to code.

3.1 Developing aerofoil cross-section

One of the first obstacles when designing a propeller is coding the blades' aerofoil cross-section. The aerofoil is defined by key parameters. These are chord length, maximum camber, position of the maximum camber and maximum thickness. In order to create the data points to define the shape, these variables need to be input into equations. Consequently, a NACA aerofoil specification is used which can be found online. [3] NACA refers to a four or five digit number where the digits designate the camber, position of the maximum camber and thickness. These digits can then be input into equations (also provided) which determine the camber line, gradient and then thickness distribution. Initially, within the program, this process was completed in a for loop which repeated for a set number of points. However, for optimisation, this was removed and replaced by matrix operations. The outputs of these equations, in the form of matrices, are then used in equations that define the upper and lower surface of the aerofoil. These matrices contain x and y coordinates for the upper and lower surfaces of the blade (i.e. xU,xL). As these are normalised, these can then be scaled. Without the provision of these equations and method of defining the input parameters, this section of code would become more difficult to create. The pseudocode to display this process is shown in the figure below.

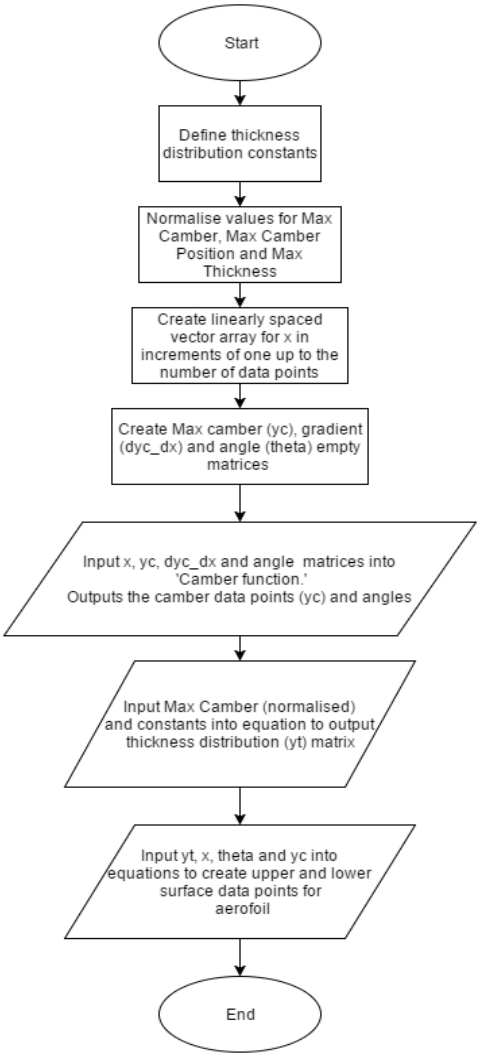


Figure 4: Pseudocode describing the creation of aerofoil cross-section

3.2 Creation method of blades/Mesh function

How the propeller is constructed is a major consideration. Originally, the code was formed in a similar manner to the provided gear example. The propeller face was first modelled in 2D. Each blade was designated an equal arc length around a circle. In angle increments, using trigonometric functions and the radius, x and y values were plotted. Using a combination of for loops and if statements, the x and y coordinates for the blades and hub were differentiated. With the 2D surface plotted, functions such as 'repmat', 'vertcat' and 'cylinder' were used. These created x,y,z matrices of coordinates that when meshed, formed a basic propeller in 3D. This is shown in in the appendix. The next step was to set the blade cross-sections to aerofoils using the data for the normalised aerofoil. The process to create these is explained in detail in the following section. To complete this next step, research into the mesh function was made. It was found that for the mesh function to work effectively, a logical sequence has to be followed while connecting data points. This meant that the concatenation to create the output matrices had to be done in a specific way. The arrangement first builds the bottom surface from the inner to the outer cylinder; it then creates the data points for the bottom surface attachment of the blade to the hub; the bottom surface of the blade is created in an outward direction as data points on each cross-section are joined together.

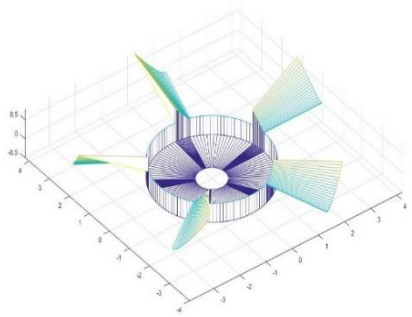


Figure 5 Propeller build sequence 1

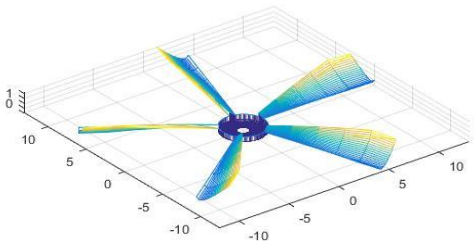


Figure 6 Propeller build sequence 2

The mesh function then creates the upper surface by connecting data points inwards. In the code, this was done by flipping the matrix that defined the upper surface values using the 'flipud' in-built MATLAB function. Finally, it reattaches at the hub and builds upwards to form the cone. The process is shown in figure 4 to 7**.** Using the 'mesh' function in this way to create the propeller required a lot of thought and code manipulation. A vital requirement for this construction to work was the ability to create multiple cross-sections along the blade.

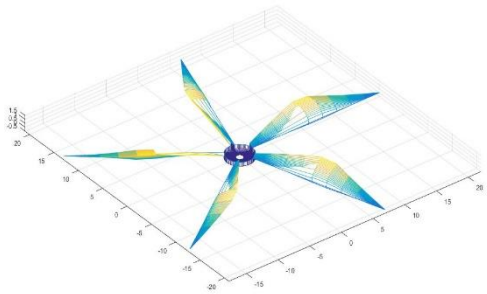


Figure 8 Propeller build sequence 3

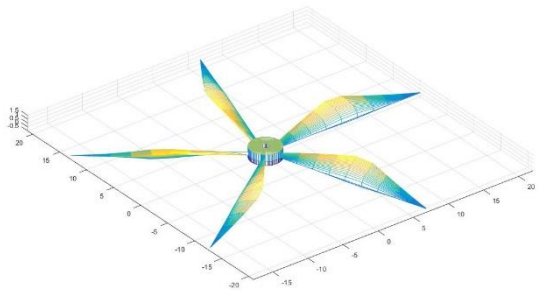


Figure 7 Propeller build sequence 4

3.3 Creating multiple aerofoil cross-sections

The creation of multiple cross-sections is imperative for a variety of reasons. It allows the blade to be modelled as accurately as possible to resemble actual blades; with several cross-sections, parameters can be changed along the blade to the user's requirements; a smoother blade can be created with varied cross-sections. To create multiple cross-sections, a for loop is created with iterations from 1 to the number of desired cross-sections. These are each inputted into the 'CreateAerofoilDataPoints' function (Figure 2**)** which outputs x,y and z matrices of the coordinates for all of the cross-sections at the end of the loop. These matrices are separated into upper and lower surface values; they are then concatenated in a logical sequence so that the propeller can be built incrementally as described in the last section. The output provides the data points for each cross-section. This enables the manipulation of aerofoil dimensions of each cross-section along the blade.

3.4 Creating geometry variations in aerofoil

The variations in the cross-sections are created by giving the aerofoil design parameters (i.e. Max Thickness) an input of a matrix of values. The function takes these input matrices given for Max Camber, Max Camber Position, Max Thickness, Chord Lengths and Pitch Angles and interpolates with a spline. This allows the values of each of these variables to be calculated at the radii of the blade cross-section being created. The two figures below demonstrate an example of how the cross-section can change from the hub to the outer edge.

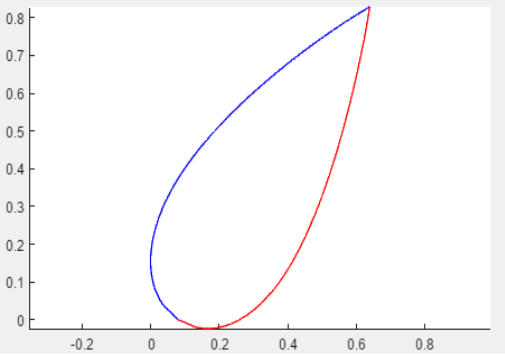


Figure 9 : Blade cross-section at hub

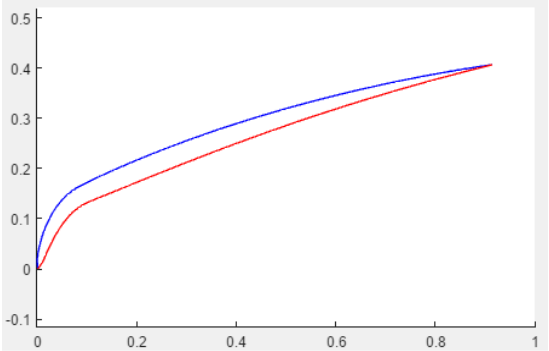


Figure 10 : Blade cross-section near the tip

The default inputs used to generate the model were created using ideal graphs of Max thickness, Max Camber, Max Camber Position, Chord Length and Pitch Angle against Blade Radius. These were found in a report detailing the geometry of marine propellers. [4] The figure below shows the plots of the variation of each of these parameters against the blade radius in my program. This was done so that blade was as accurately designed as possible in the default case.

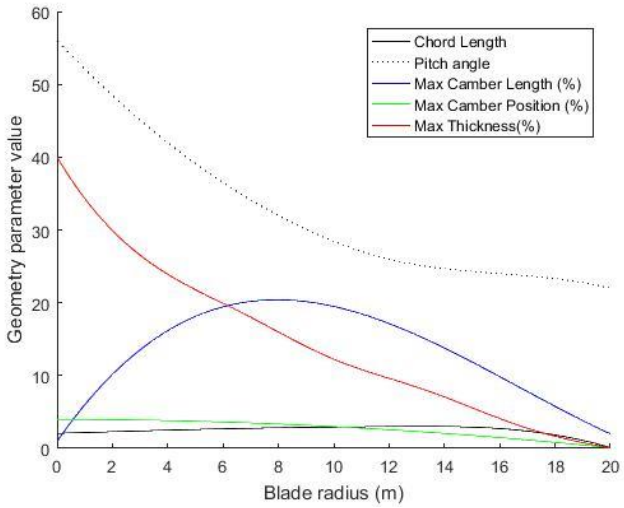


Figure 11 : Graph showing the variation of aerofoil dimensions against blade radius

3.5 Minor challenges

There were some minor challenges when creating the code. The majority of the program involves variables in the form of large matrices that are continually inputted through equations and functions. This made it difficult to ensure that all the matrices dimensions remained constant. The dimensions remaining constant, particularly the number of columns, was essential as these matrices were concatenated to form the output matrices. Additionally, another challenge was the creation of a pitch angle along the length of the blade. Equations for the rotation of data points were found online. [5] These equations were used to rotate specific coordinate points on cross-sections through user inputed angles.

4 Running the Code

The code should be run on MATLAB 2016b and on a windows operating system. This will reduce the likelihood of any errors or inaccurate results. Furthermore, the GUI used is built on appdesigner which is only available on the 2016 versions. First of all, download the zip file labelled Final-Project-2.zip, decompress it by right clicking on unzip, or if using a newer version of windows, this should be done automatically and can be opened directly. In the folder, there should be five files. These are labelled PropellerProject.mlapp, AerofoilViewFunction.m, PropellerFunction.m (Main Script) and surf2stl.m. There is also an extra stl sample file created for the default inputs. To run the program, open the file PropellerProject.mlapp and the GUI, shown in Figure 12 below, will appear. To view the default or a random propeller design, click the required button on the right followed by 'Execute.' A figure in a separate window will open displaying the propeller. To view the code, the remaining '.m' files can be opened.

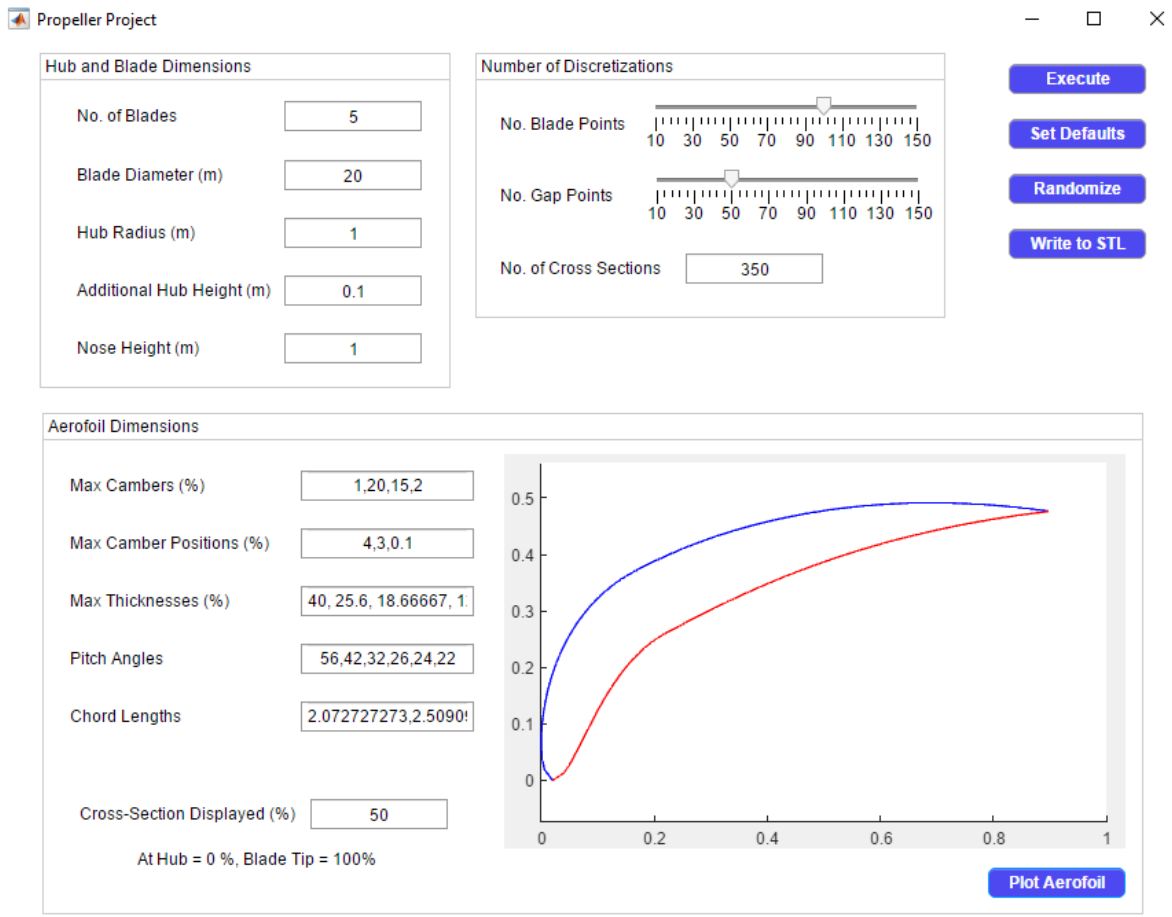


Figure 12: GUI run with default inputs

The GUI is split into three separate panels. The 'Hub and Blade' panel describes basic inputs for the overall dimensions of the propeller. For further clarification, 'Additional Hub Height' refers to the addition of hub height on either side of the blade attachment. The 'Number of Discretizations' panel allows the user to change the number of data points that define the blades and the gaps in between. Furthermore, the number of cross-sections that make up the blade can also be inputted. This is restricted to two, if one is inputted, then three cross-sections will be used instead. The most important panel is 'Aerofoil Dimensions.' These parameters determine the shape of several aerofoils along the length of the blade. The values are input as strings; therefore, the user can determine how many inputs they want to give up to the total amount of cross sections. However, no less than two inputs can be given otherwise an error message will be displayed in the command prompt. This is because at least two cross sections along the blade need to be defined to produce the propeller. Consequently, these two cross-sections will be at the hub and blade tip. As these inputs were modelled using the NACA aerofoil series, ideal shapes can be created using tested aerofoil designs with inputs on their website. [3] Inputs for NACA aerofoil's are, however, restricted

to certain ranges. These are shown in the table below. The equations used to create the aerofoil cross-sections may not work outside these ranges or in some cases may create distorted shapes. This is especially the case when 0 is given for some values.

Dimension	Range
Max Camber (%)	0 to 9.5
Max Camber Position (%)	0 to 90
Thickness (%)	1 to 40

Table 1: Table showing the restricted range of aerofoil dimensions

Lastly, the 'Cross-Section Displayed' input displays the shape of the aerofoil at a designated position along the blade. This is shown on the figure in the GUI. For example, the cross-section for the default case shown above is half way through the blade. By adjusting the value, cross-sections at any part of the blade can be viewed. This reinforces visually the variation along the blade. After the values are chosen, click 'Execute' to create a figure with the propeller design or 'plot aerofoil' to view the cross-section at the chosen percentage. Once the user is satisfied with their inputs, an STL file can be generated with these values by clicking on 'Write to STL.' A window will open; Choose a file location, name it and save the STL file. This can then be viewed using modelling software or '3D builder' on Windows 10.

The following are figures displaying different user inputs.

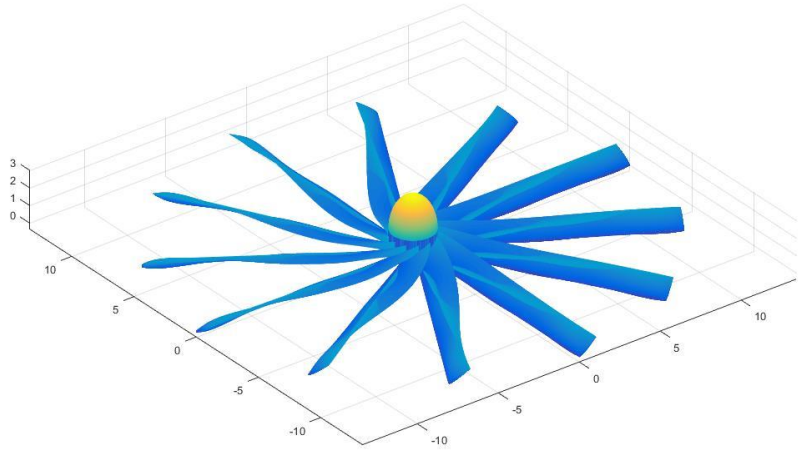


Figure 13: Figure displaying randomised inputs

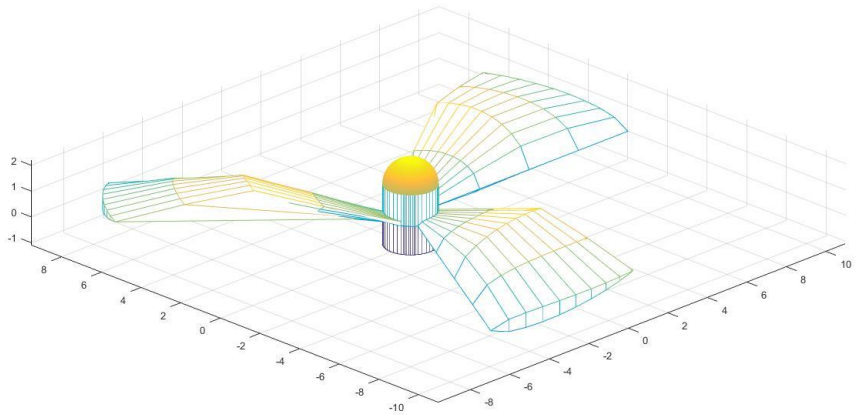


Figure 14 : Displays low resolution propeller with three blades

5 Conclusion

In conclusion, MATLAB can be used to parametrically design and generate models of engineering objects such as propellers to high levels of precision. The code, using initially defined parameters, constructs an aerodynamically accurate propeller. This is due to the inclusion of mathematical models and equations, such as the use of NACA profiles for the cross-sections of the blades. Using MATLAB to create propellers is advantageous as the ability to vary the input values results in the creation of several different output designs. Furthermore, the generation of an STL file allows the designs to be used for additive manufacturing or 3D printing. In terms of personal development, the project provided insight into the mechanism of a propeller and the physics behind its function. Furthermore, the project enabled us to develop our ability to write computer code as well as understand the importance of parametric design in engineering.

6 Bibliography

[1] "ParametricCamp," [Online]. Available: http://www.parametriccamp.com/en/what-is-parametric-design/.

[2] [Online]. Available: https://www.researchgate.net/publication/224305726_An_Open_Source_Parametric_Propeller_Design_Tool.

[3] "Airfoil Tools," [Online]. Available: http://airfoiltools.com/airfoil/naca4digit?MNaca4DigitForm%5Bcamber%5D=9&MNaca4DigitForm%5Bposition%5D=45&MNaca4DigitForm%5Bthick%5D=12&MNaca4DigitForm%5BnumPoints%5D=81&MNaca4DigitForm%5BcosSpace%5D=0&MNaca4DigitForm%5BcosSpace%5D=1&MNaca4DigitForm%5Bclo.

[4] J. E. S. Donald R. Smith, "The Geometry of Marine Propellers," 1988.

[5] G. S. Owen, 1999. [Online]. Available: https://www.siggraph.org/education/materials/HyperGraph/modeling/mod_tran/2drota.htm.

[6] "ALL3DP," 17 November 2016. [Online]. Available: https://all3dp.com/what-is-stl-file-format-extension-3d-printing/.

7 Appendix

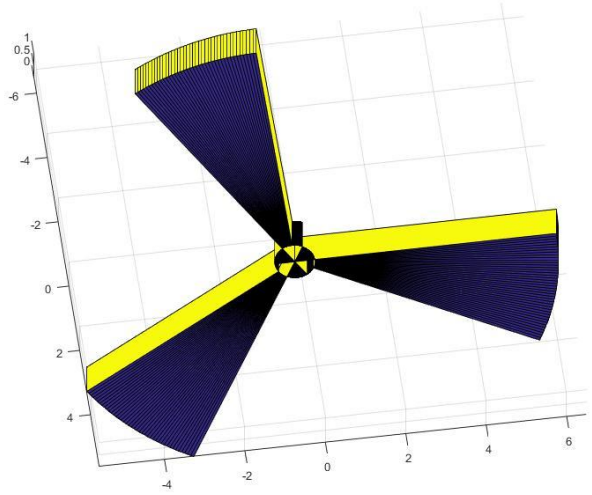


Figure 15: Intial design of propeller