TRIBHUVAN UNIVERSITY

INSTITUTE OF ENGINEERING

PULCHOWK CAMPUS

Mathematical modeling for design of Francis Turbine using MATLAB

Submitted By:
Balkrishna Poudel(077BME005)
Biswash Khatiwada(077BME014)
Hari Prasad Gajurel(077BME053)
Ravin Purbey(077BME030)

A REPORT
SUBMITTED TO THE DEPARTMENT OF MECHANICAL AND AEROSPACE ENGINEERING IN PARTIAL FULFILLMENT OF THE REQUIREMENT FOR THE COURSE OF FLUID MACHINES

DEPARTMENT OF MECHANICAL AND AEROSPACE ENGINEERING
LALITPUR, NEPAL

August 2023

ABSTRACT

The Francis turbine is the most popular hydro turbine due to its ability to work well over a broad range and its high efficiency. Its important part, the runner, is connected to the common shaft of the electrical generator. Thus, the design of the runner plays a vital role in the operation of the machine.

We aim to model the blade profile and runner of the Francis turbine. Among the many design approaches (Direct method, indirect method, Bovet method, and curve fitting), the Bovet method has been used to obtain the blade profile as it is found that this method provides the blade profile in the most efficient way.

Based on the given input parameters: net head, discharge rate, and speed of the runner, we calculated the inlet and outlet parameters. The implementation of the Bovet Method is facilitated through computational tools, particularly MATLAB. We then plot the Meridional View, Perpendicular view, Axial view, and 3D design of the runner blade. The finding of this study helps to contribute to the design of runners by a systematic approach to the calculation of its parameters.

Keywords: *Francis turbine,Bovet method,MATLAB*

ACKNOWLEDGEMENT

We would like to express our deepest gratitude to all those who have contributed to the completion of this project on mathematical modeling for the design of the Francis turbine using MATLAB.

First and foremost, we would like to thank our assistant professor, Neeraj Adhikari, for his invaluable guidance and support throughout this research endeavor. His expertise and insights have been instrumental in shaping the direction of this paper and ensuring its scientific rigor.

We extend our appreciation to the academic community and researchers in the field of design of the Francis turbine they have shared. Their groundbreaking contributions have laid the foundation for our understanding and exploration of the mathematical modeling of the Francis turbine.

We are also grateful to the institutions and organizations that provided the necessary resources, and facilities to conduct this project. Their continued commitment to fostering scientific research has made this project possible.

Furthermore, we would like to express our heartfelt appreciation to our seniors, Archana Lamsal and Siropa Shrestha, for their unwavering support and understanding throughout this research journey. Their encouragement and belief in our abilities have been a constant source of motivation.

Last but not least, we would like to acknowledge the contributions of our colleagues and peers who provided valuable feedback and constructive discussions during the development of this paper. Special thanks to Darshan Koirala as he has significantly enhanced the quality and clarity of our work.

Although it is not possible to individually mention all those who have contributed, we sincerely thank each and every person who has played a role, no matter how small, in the completion of this research paper.

TABLE OF CONTENTS

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Background

Hydropower has been recognized as a sustainable source of energy with almost zero input cost. The first hydroelectric plant was constructed in Wisconsin, USA, in 1882. This plant made use of a fast-flowing river as its source. Dams were built later to create artificial water storage areas. The water flow rate to the power plant turbines was regulated by these dams. The first hydroelectric plant in Nepal was built in 1911 at Pharping (500 KW). Nepal possesses the largest power system in South Asia, generating 2,577 MW of electricity while it has the potential to generate about 40,000MW of electricity. The total population with access to grid-connected electricity has reached 92.51% and all power stations generated the highest recorded annual energy of 3,259 GWh in FY 2021/22. [1] Besides, traditional energy like fuel wood, agriculture residues, and animal dung and commercial energy like petroleum, hydropower, and solar energy constitute other sources of energy in Nepal.

Hydraulic turbines convert hydraulic energy (energy possessed by water) into mechanical energy (which is further converted into electrical energy). These turbines are classified according to the type of energy available at the inlet of the turbine, the direction of flow through the vanes, the head at the inlet of the turbine, and the specific speed of the turbines.

Impulse turbines are driven by one or two high-velocity jets. Each jet is accelerated in a nozzle external to the turbine wheel known as the turbine rotor. If friction and gravity are neglected the fluid pressure and relative velocity do not change as it passes over the blades/buckets. In reaction turbines, part of the fluid pressure change takes place externally and part takes place within the runnet. External acceleration occurs in the guide vanes and flow is turned to enter the runner in the proper direction. Additional acceleration of fluid relative to the rotor occurs within the moving blades. So both relative velocity and pressure change over the runner. Reaction turbines run full of fluid.

Francis turbine is a reaction turbine. Water enters circumferentially through the turbine casing. It enters from the outer periphery of guide vanes and flows into a runner. It flows down the rotor radially and leaves axially. Water leaving the runner flows through a diffuser known as a draft tube before entering the tail race.

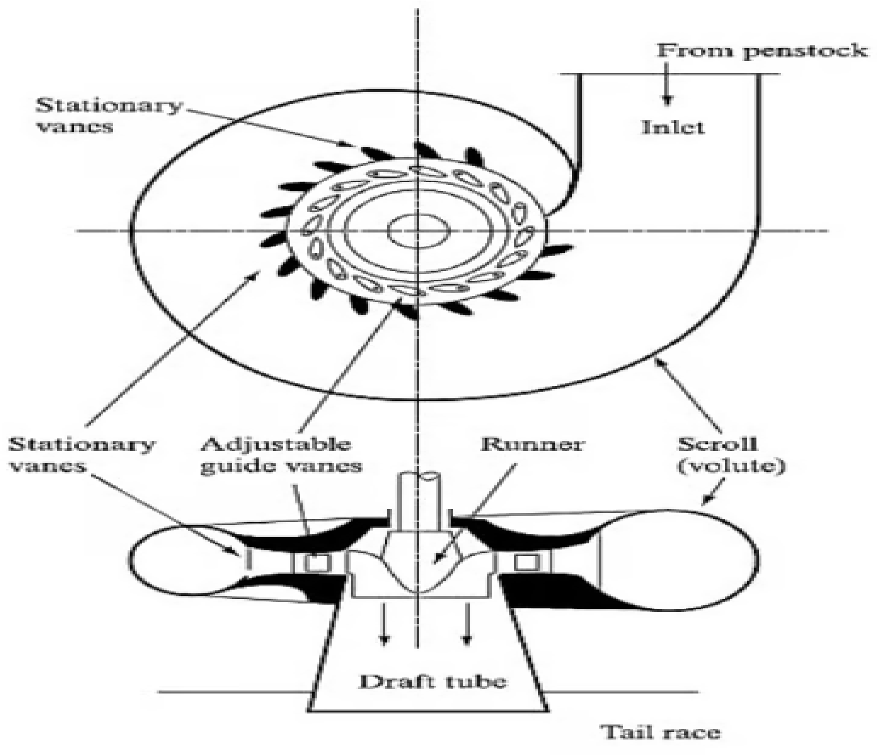


Figure 1.1: Different Components of Francis turbine

1.2 Problem Statement

Since Nepal has a range of altitudes, the large potential of energy remains unused (untapped) in our flowing systems. This energy available near the ruler's consumption can meet the significant energy demands of the place and the country. Most hydropower systems with low heads are installed close to areas that need energy. So large electrical transmission lines are not necessary. The method for the calculation of the Low head Francis turbine is a lot of tedious and the approach for the calculation is not easily found. The design complication slows manufacturing and installation processes and also results in the untimely wear of turbines. Determination of limiting curves for the profile of the low-head Francis turbine runner blade itself is a very difficult task. Previous attempts made to develop software for the design of the Francis turbine were only applicable for a certain range due to the many assumptions and references taken for the runner blade limiting curves. So, in this project, we aim to develop a GUI-based MATLAB that calculates the parameters program based on the Bovet method and hence gives a reference design of the runner.

1.3 Objectives

Main objective:

- To design and model various profiles of Bovet-based Francis runner.

Minor Objective:

- To study the design parameters of the runner blade.
- To develop a MATLAB code to design Francis turbine runner blade.
- To study the results obtained for different low-head hydropower stations using the software developed

1.4 Scope of the Work

This project is applicable to generating various profiles for the reliable and expeditious design of Francis turbine runners. However, only software-based design will be included in the entire project.

1.5 Limitations

The limitations of this project are enlisted as:

- This GUI only works for the range of the low-head turbine.
- This project does not involve the fabrication process. So, The material properties and the manufacturing complexities of the optimized runner blades are not considered.
- There requires a change in the code in order to change the blade thickness. There is no experimental validation of the results obtained.

1.6 System Requirements

MATLAB is used for the entire programming of the project for generating the profile of the blade.

Chapter 2

Literature Review

2.1 History of Francis Turbine

In 1826, French engineer Benoit Fourneyron created an outward-flow turbine that had nearly 80% efficiency. Water was fed through a stationary inner core in the Fourneyron turbine before being sent forth through curved horizontal guide vanes. In 1844, Uriah A. Boyden, an American engineer from Massachusetts, created an outward flow turbine that significantly outperformed the Fourneyron turbine and had a peak efficiency of about 80%. Boyden added a conical approach path for the incoming water to reduce losses brought on by the Fourneyron design's abrupt area shifts. Boyden also recognized the lost potential in the discharged water and added a submerged diffuser at the discharge of the wheel and a diverging exit passage. James B. Francis, Chief Engineer for Locks and Canals Company in Lowell, put Boyden's inward-flow turbine through a series of systematic testing. These experiments resulted in the standardized technique for evaluating the performance of hydro turbines. Francis also created a method for designing scientific turbines based on Boyden's earlier work, which involved evaluating the movement of water particles at various relative velocities under a particular head. In 1847, Francis designed and built his first model scale inward-flow turbine after purchasing patent rights from Howd. The Howd and Poncelet inward-flow wheel designs were modified to create the Francis model, which had a peak efficiency of 71%. With the addition of a progressive vertical area shift to provide acceleration, the guiding vanes were nearly identical to the straightforward straight blade designs in the Howd patent. The turbine blades had no curve and were also quite short. The straight blades reduced performance but made it easier to calculate the various flow angles needed for Francis' turbine performance theory. The first full-scale Francis turbine was installed at the Boott Cotton Mills in 1849. Francis made a number of changes to his original design, including adding a curved discharge after the water flowing through the turbine wheel and bringing water from the penstock offset to one side of the turbine to add pre-swirl. Francis also added curvature to the turbine blades and fixed guiding vanes. The first Francis turbine's performance prediction accuracy was its most striking feature. Francis calculated a peak efficiency of 79.31% for his design, and the installed turbine's testing revealed a 79.37% efficiency. In 1855 Francis published his work in Lowell Hydraulic Experiments, which is considered one of the most outstanding American contributions to hydraulic engineering. [2]

2.2 Overview of the Francis Turbine

The primary characteristic of the Francis turbine is the direction of the water as it flows through the turbine. Although the flow enters the turbine in a radial direction, it exits in the axial direction. Because of this, the Francis turbine is referred to as a mixed-flow turbine. A Francis turbine's blades are carefully designed to maximize the energy captured from the water flowing through it. The turbine spins due to the force of the water on the blades, and a generator uses the rotation to produce energy. The height of the available water head and the flow volume define the blade form. A Francis turbine can effectively capture 90–95 percent of the energy present in the water.[3] The majority of hydroelectric power facilities in nations with mountainous terrain have machinery installed for operation under high heads, which is within the range of 200 to 1200m. Although a feasible upper limit for the Francis turbine is roughly 700m, both the high-head Francis and Pelton turbines may be used in this head range. Francis turbines with relatively low heads ranging from 3 to 600m were used in low-head projects for hydroelectric power development, especially in non-mountainous nations. Low-head turbines may have quite different mechanical design concepts and general operating requirements than high-head turbines. The best performance of the Francis turbine is between heads ranging from 100 to 300m.[4]

2.3 Low-head Francis Turbine

Low-head Francis turbines encompass a range of parameters for its specification. The turbine is designed to operate effectively within a head range of 15 to over 100 meters and accommodate flow rates varying from 0.3 to 100 cubic meters per second. The speed number falls within the range of 0.2 to 2.15. The low-head Francis turbines are applicable for flat landscapes and can be used in often running and polluted water. A dam or weir constructed is required in this low-speed turbine (approx. 100 to 500 rpm). But for the lower heads and smaller runner diameter, the design of the intake and the draft tube is crucial and the power loss can happen due to algae and grass. So, Low-head Francis turbines need optimized flush control. [5]

2.4 Bovet method and Conformal mapping method

Bovet methods were suggested in ‘International Conference on Small Hydropower - Hydro Sri Lanka, 22-24 October 2007’ by A. Nourbakhsh, O. Seyed Razavi., H. Khodabakhsh, A. Mehrabadi in their research article ‘New Approach for Hydraulic Design of Francis Runner Based on Empirical Correlations’.[6] Bovet method uses empirical equations to obtain dimensional parameters of Francis runner. The dimensionless specific speed (n0) is the main parameter to find out the whole dimension of the meridional plane. This method can be used for sites with dimensionless specific speeds between 0.1 and 0.8. Most of the medium-head and low-head Francis turbines fall in this range. [7]. Considering this aspect, the Bovet method is used to design the low-head Francis turbine. This method becomes applicable in designing the meridional plane of a runner. Conformal mapping is applied for a perpendicular plane. The combination of both methods eases the design process and reduces time.

2.5 Related works

The goal of Kristine Gjsaeter’s paper, ”Hydraulic Design of Francis Turbine Exposed to Sediment Erosion,” is to create a Francis turbine that is extremely resistant to sediment erosion. It features a user-friendly MATLAB tool for creating a high-head Francis turbine’s hydraulic design. [8] The research ‘Optimized Design of Francis Turbine Runner for Sand Laden Water’ by Krishna Prasad Shrestha, Bhola Thapa, Ole G. Dahlhaug, Hari Prasad Neupane, Nikhel Gurung and Atmaram Kayastha’ was carried out with an objective to develop a new Francis turbine runner that can handle sediment load with specific objectives: 1. To determine effective area and its impact on erosion on Francis turbine runner blade. 2. To determine possible solutions for the minimizing effect of sand erosion on the turbine runner. 3. To innovate optimization technology on Francis turbine runner resigns. The MATLAB program ”KHOJ” was used to produce the basic design data, and it also yielded the 3D coordinates of the component parts. The runner 25 blade was created when these coordinates were converted into a suitable file format and imported into Pro/Engineer software. After that, the Pro/Engineer was used to create the cut model for the FSI analysis, and ANSYS CFX was used to view and optimize the outcome. [9]

Chapter 3

Methodology

The section describes the Francis turbine runner design approach using the Bovet method which is based on fundamental parameters of the turbine. It includes the calculation of the parameters of the turbine using the given data and the implementation of the Bovet and Conformal.

3.1 Main Dimensions

The fundamental parameters of the turbine are net head (H), flow rate (Q), (determined by topographical and hydrological features of the plant), and rotational speed (nr).

Net Head$(H) = 15m$
Flow Rate$(Q) = 0.3m/s$
Rotational Speed $(N) = 1500rpm$
Acceleration due to gravity $(g) = 9.91m/s^2$

3.2 Meridional Profile

The dimensionless specific speed number is the key parameter used to generate the meridional profile of a runner channel. Dimensionless Specific Speed is given as

$$n_o = \frac{2\pi N}{60 \times 2gH} \times \sqrt{\frac{Q}{\pi}} \tag{3.1}$$

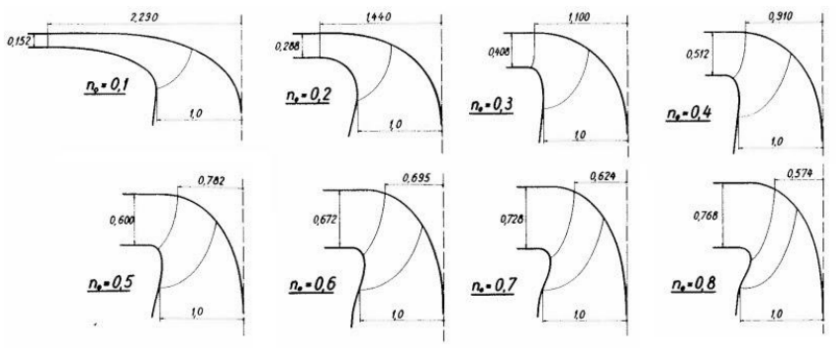


Figure 3.1: Variation of Meridional profile with specific speed [7]

3.2.1 Hub, Shroud, Leading, and Trailing Edge

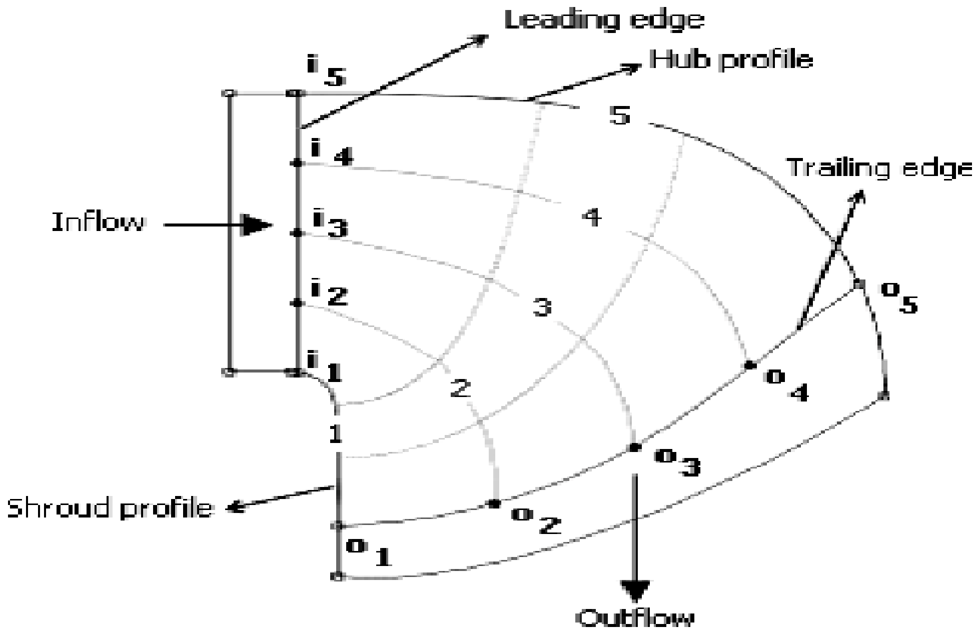


Figure 3.2: Meridoinal sections of the runner blade

Indication	Hub	Shroud	Leading Edge	Trailing Edge
Representation	“i”	“e”	“1”	“2”
Nature	Equation	Equation	Parabolic Arcs	Parabolic Arcs

Table 3.1: Symbol reprentation

The leading-edge curve has a center at shroud point (1e) and passes through (1i) point at the hub. Similarly, the trailing edge curve has a center at shroud point (2e) and passes through the point (2i) at the shroud.

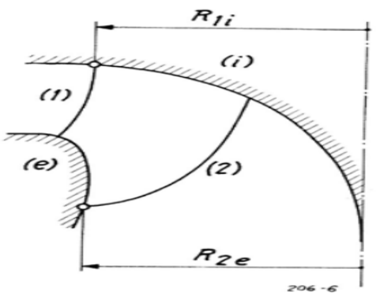


Figure 3.3: Limiting Curve of the blade
[7]

The BOVET well-known relations give the geometry of hub and shroud curves in the runner blade zone.

$$\left(\frac{y}{y_m}\right)_{i,e} = \left[3.08 \times \left(1-\frac{x}{l}\right)\sqrt{\frac{x}{l}\left(1-\frac{x}{l}\right)}\,\right]_{i,e} \tag{3.2}$$

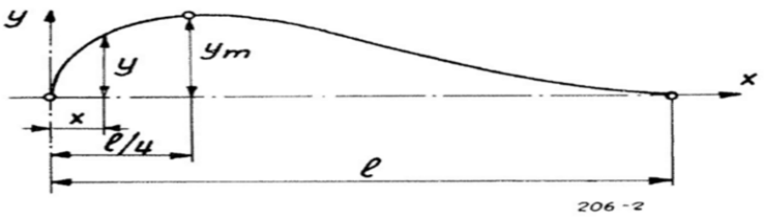


Figure 3.4: Characteristics shape of hub and shroud curve of the meridional channel
[7]

Using the dimensionless specific number, the normalized characteristic dimensions for the flow passage were calculated using an empirical relation

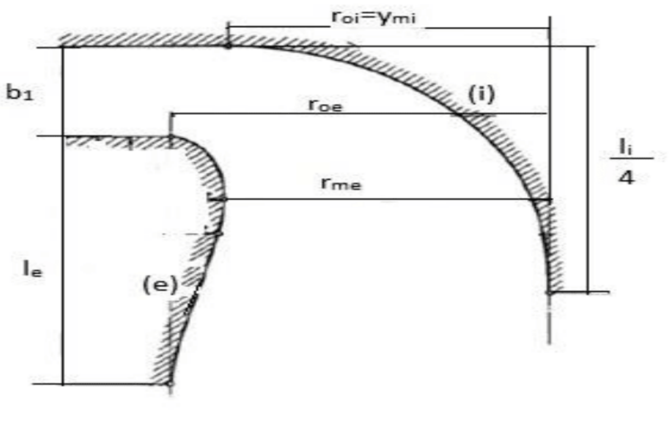


Figure 3.5: Parameters of the Meridional View

The calculations of r_{oe} depends on the value of the dimensionless specific speed (n_o)

If $(n_o) < 0.275$

$$r_{li} = r_{oe} = 0.493\frac{0.493}{(n_o)^{2/3}} \tag{3.3}$$

If $(n_o) > 0.275$

$$r_{li} = r_{oe} = 1.225 - 0.3 \times n_o \tag{3.4}$$

$b_o = 0.8 \times (2 - n_o) \times n_o$	$r_{oi} = y_{mi} = 0.7 + \dfrac{0.16}{n_o + 0.08}$
$l_i = 3.2 + 3.2 \times (2 - n_o) \times n_o$	$l_e = 2.4 - 1.9 \times (2 - n_o) \times n_o$

Table 3.2: Equations of meridional plane parameters

All the runner's dimensions are rated with respect to the runner hydraulic outlet radius, $r_{2e} = 1$. [Reference]
The actual dimension of the meridional channel can be found by multiplying the normalized dimensions by R_{2e} which is given by:

$$R_{2e} = \left(\frac{\frac{Q}{\pi}}{\phi_{2e} \times n_r} \right)^{1/3} \tag{3.5}$$

where, $\phi\epsilon(0.26, 0.28)$ = specific flow = 0.27

Hence, therefore the actual dimensions of the meridional channel are calculated as:

$R_{oi} = r_{oi} * R_{2e}$ $R_{li} = r_l i * R_{2e}$ $Y_{2e} = y_{2e} * R_{2e}$	$B_o = b_o * R_{2e}$ $Y_{me} = y_{me} * R_{2e}e$ $R_{oe} = r_{oe} * R_{2e}$	$L_e = l_e * R_{2e}$ $R_{me} = r_{me} * R_{2e}$	$L_o = l_o * R_{2e}$ $X_{2e} = x_{2e} * R_{2e}$

Table 3.3: Actual Parameters of Meridional Channel

Hence, R_{1e},R_{2e},$R_{1i}$and R_{2i} are calculated, and hence parabolic arc method was used for the leading and the trailing edge while determining the leading and trailing edge curves.
The form of limit curves is defined by a parabolic equation. The equation of a parabola is used to determine the leading and trailing edge curves. It is given as:

$$y = a(x-h)^2 + k \tag{3.6}$$

where, (h,k) is the center of the parabola and (x,y) is a point on the parabola.

For the leading edge, point 1e is used as the center, and point 1i is used as a passing point. Similarly, for the trailing edge, point 2e is used as the center and point 2i is used as a passing point. The parabola was matched by defining point R1i at the leading edge and the R2e at the trailing edge.

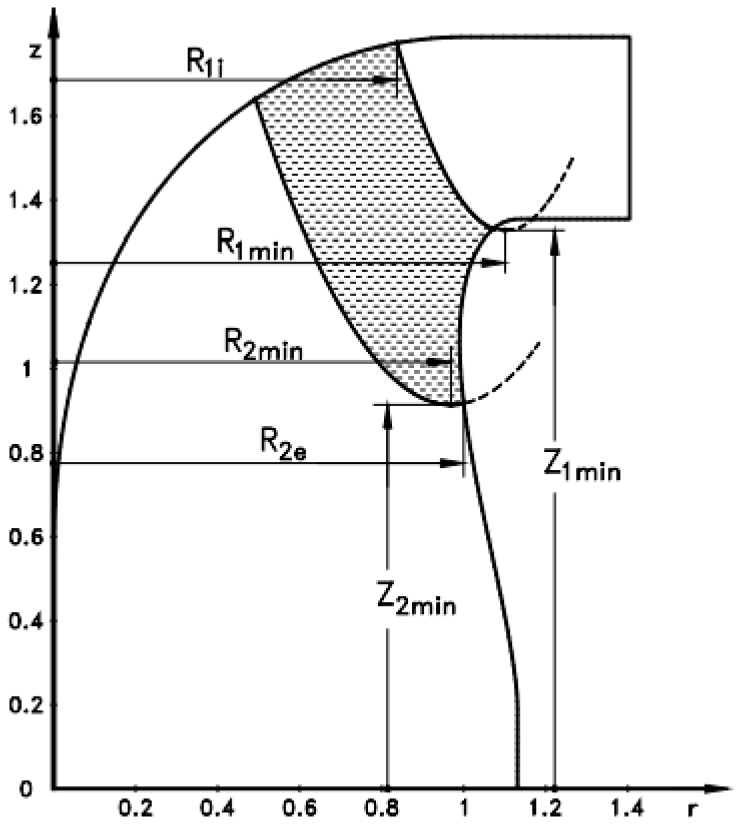


Figure 3.6: Leading and Trailing Edge given by parabola

3.2.2 Outlet and Inlet Parameters

$$\begin{aligned}
\text{Overflow Rate}(Q_n) &= 1.2Q \\
\text{Power}(P) &= \eta_h \times H \times Q \times g(KW) \\
\text{Specific Speed}(n_r) &= n_r.\sqrt{\frac{p * 10^{-3}}{H^{5/4}}} \\
\text{Angular Rotational Speed}(\omega) &= (2 \times \pi * n_r)/60
\end{aligned}$$

The real value of the outlet nominal radius R_2 is calculated with the formula given by BOVET:

$$\text{Outlet Radius}(R_2) = \left(\frac{Q}{\pi * \phi * \omega}\right)^{1/3} \tag{3.7}$$

where, $\phi = 0.24$
Outlet Diameter $(D_2) = 2 * R_2$

The empirical relations are given by the experiment done by F.de Siervo and F.de Leva which gives the following relation between specific speed and outlet diameter with the main different parameters of the runner and they are as follows:

$$\text{Inlet Diameter(D1)} = (0.4 + \frac{94.5}{n_s}) * D2 \tag{3.8}$$

$$H_1 = D_2 \times (0.094 + 0.00025 n_s) \tag{3.9}$$

For the calculation of H2:
For $n_s \epsilon (50, 110)$ ::

$$H_2 = D_2 * (-0.05 + \frac{42}{n_s}) \tag{3.10}$$

And
For $n_s \epsilon (110, 350)$ ::

$$H_2 = D_2 * \left(\frac{1}{3.16 - 0.0013 * n_s}\right) \tag{3.11}$$

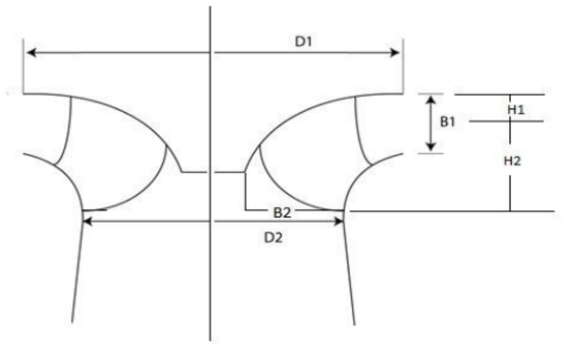


Figure 3.7: Inlet and Outlet Dimensions

3.3 Calculations of Velocity Triangles

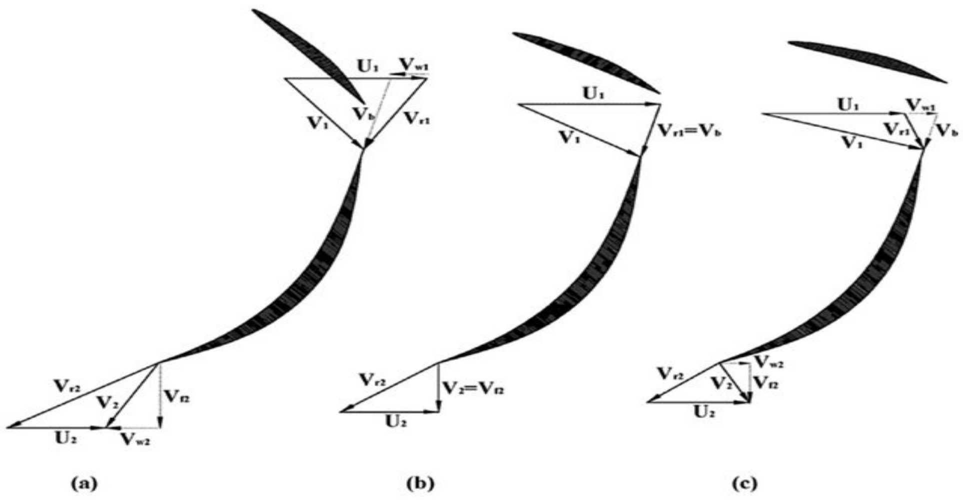


Figure 3.8: Velocity Triangles for different Beta Angle

3.3.1 Outlet Velocity Triangle

We assume the Best Efficiency Point (BEP) i.e., no swirl conditions at the outlet for the calculations. Therefore, the tangential component of the outlet velocity V_{w2} equals zero. Hence, the velocity triangles reduce as:

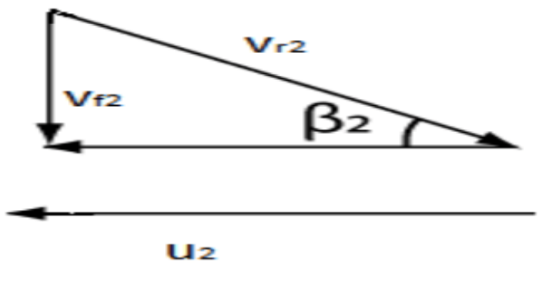


Figure 3.9: Outlet Velocity Triangle

The further calculations are described below:

$$U_2 = (\pi * n_r * D_2)/60$$
$$V_{f2} = Q/(\pi * D_2 * B_2)$$
$$\tan(\beta_2) = V_{f2}/U_2$$

3.3.2 Inlet Velocity Triangle

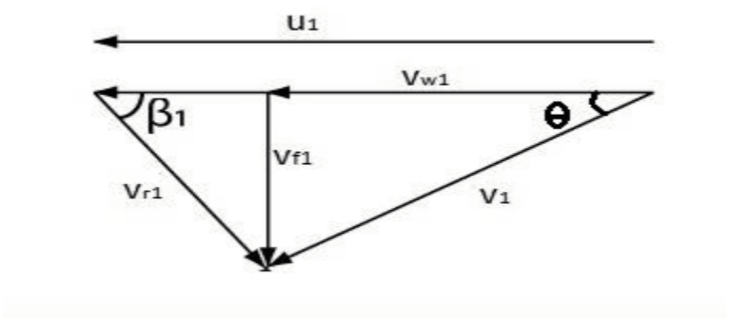


Figure 3.10: Inlet Velocity Triangle

$$U_1 = (\pi * n_r * D_1)/60$$
$$V_{f1} = Q/(\pi * D_1 * B_1)$$
$$V_{w1} = \frac{\eta h \times g \times H}{\pi \times U_1}$$
$$\tan(\beta_1) = \frac{V_{f1}}{V_{w1}U_1}$$

3.4 Axial View

From the Bovet method, the meridional view was obtained. Two curves from the meridional view: Hub and Shroud are used and with the help of the equation of these curves the initial axial view was plotted.

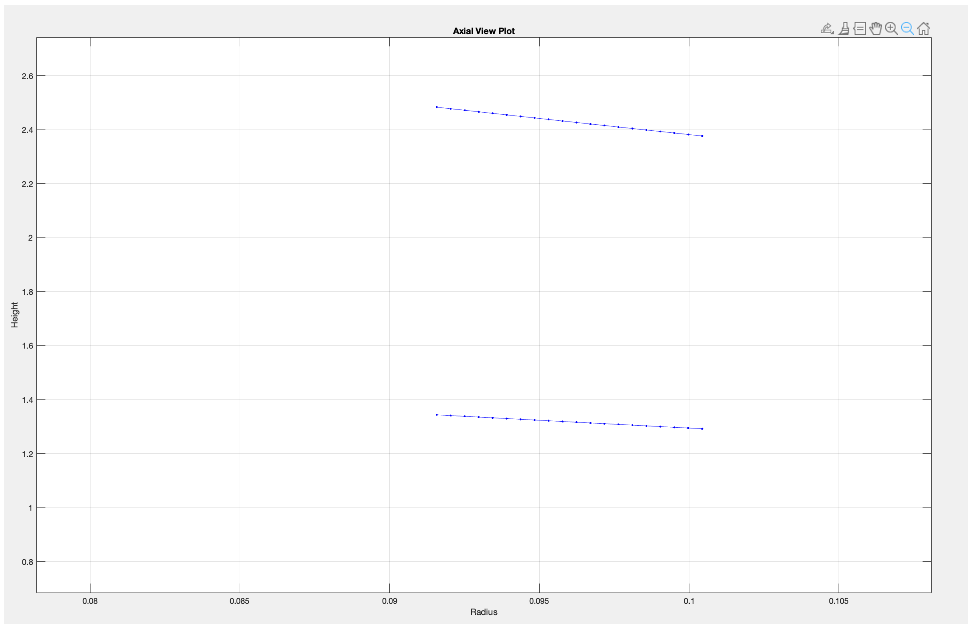


Figure 3.11: Axial View before

Then the streamlines were drawn in between the hub and shroud with the help of interpolation. The figure below represents a rough plot of the axial view for the equations.

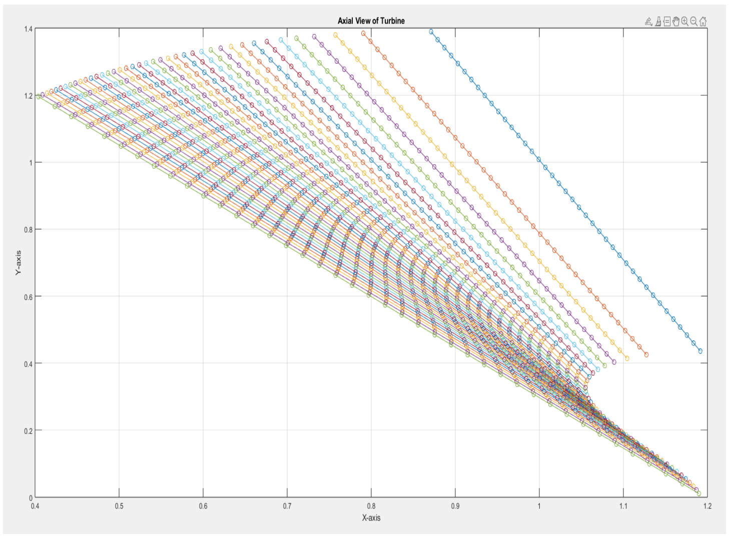


Figure 3.12: Axial View after

3.5 Perpendicular View

When the flow streamlines between the entrance and exit edges are determined, other turbine specifications such as entrance and exit flow angles can be calculated. Then we can use the conformal mapping method to draw the form of the blade in the perpendicular plan. This method is based on trial and error and gives us the ability to map each of the streamlines from the meridional view to the perpendicular plan. A curved triangle can be drawn by known streamlines and flow angles. This triangle is a transformation link between two plants.

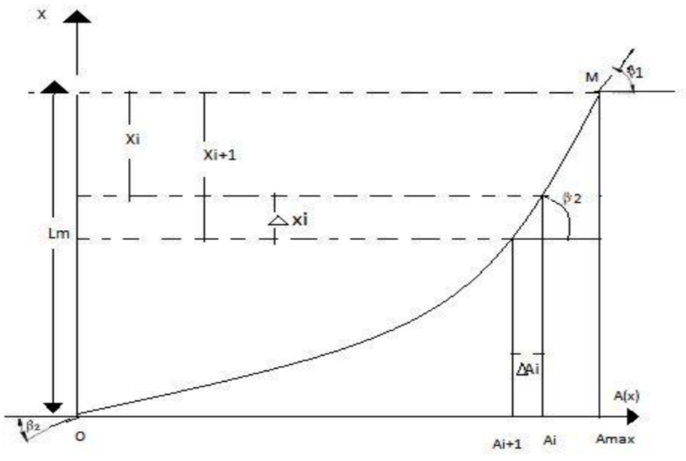


Figure 3.13: Conformal Mapping Triangle
[6]

Regarding decreasing the design period in this part, we can estimate the curved edge of the triangle as a second-order polynomial equation and each of the three unknowns of this function can be calculated:
Let us assume,

$$f(x) = y = ax^2 + bx + c \tag{3.12}$$

At, f(0) = 0 , c = 0
Now, at $x = 0$

$$\frac{dy}{dx} = \tan\beta_2 = 2ax + b \tag{3.13}$$

So, b = $\tan\beta_2$
At x=Amax,

$$\frac{dy}{dx} = \tan\beta_1 = 2aA + b \tag{3.14}$$

$$a = \frac{(\tan\beta_1 - \tan\beta_2)}{2A} \tag{3.15}$$

From the above equations

$$f(x) = y = \left(\frac{\tan\beta_1 - \tan\beta_2}{2A}\right)x^2 + (\tan\beta_2)x \tag{3.16}$$

Here, at x=Amax, f(x)= Lm
So,

$$L_m = \left(\frac{\tan\beta_1 - \tan\beta_2}{2} + \tan\beta_2\right) \times A \tag{3.17}$$

Therefore,

$$A = \frac{2L_m}{\tan\beta_1 + \tan\beta_2} \tag{3.18}$$

Lastly, after determining Lm and A, we get a second-degree polynomial equation. For every Lm, there will be a different polynomial equation and all these equations are plotted to get a perpendicular view.

3.6 3D view of Blade and Runner

After getting the meridional view, we obtain the x and y coordinates of the hub, shroud, and all the streamlines. In the perpendicular view, we have the equations of all the above-mentioned lines which are in the YZ plane. Hence using the y coordinate from the meridional view and substituting it in the equations in the yz plane of the perpendicular view, we obtain the z coordinate of each point. Hence we have all x.y and z coordinates of each point lying in all the streamlines. Thus by merging these two views, we obtain the 3D view.[6] Thus the blade was arranged in a circular pattern and hence a 3D view of the turbine runner was obtained.

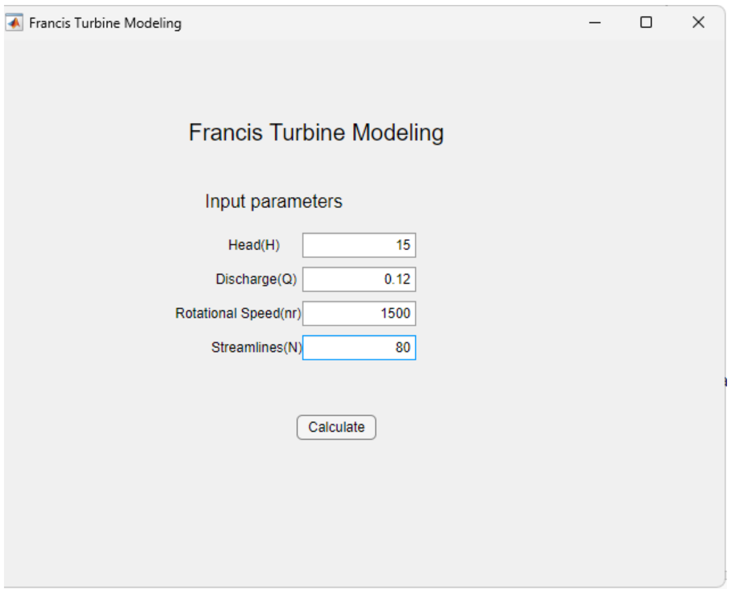
Chapter 4

Results and Discussions

The implemented GUI application for Francis turbine modeling successfully generates insightful visualizations and calculations pertaining to the geometry and blade profiles of the turbine.

4.1 Program Review

In order to run the program, the user has to run “Francis Turbine Modeling.mlapp”, after which the main interface opens.


Francis Turbine Modeling

Francis Turbine Modeling

Input parameters

Head(H) 15
Discharge(Q) 0.12
Rotational Speed(nr) 1500
Streamlines(N) 80

Figure 4.1: Home User Interface

The user has to input values of the fundamental parameters(H), (Q), (nr), and (N). On clicking the push button “Calculate”: the interface displays the following profiles:

4.2 Meridional Profile

The program calculates the turbine's key geometric properties. These include the inlet and outlet blade thickness, blade heights, and radii. Based on the calculations, it displays the meridional view bounded by the hub and shroud curves of the turbine and streamlines characterizing the lines of fluid flow in the turbine.

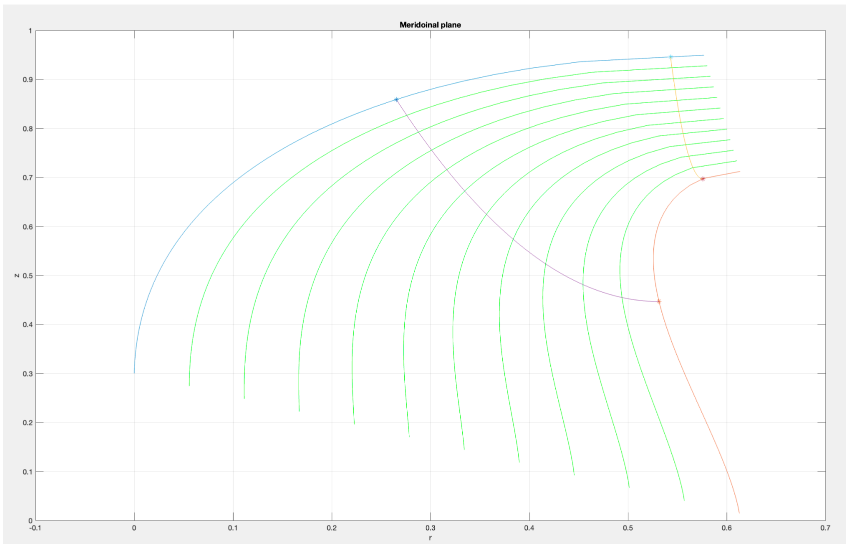


Figure 4.2: Meridional profile

4.3 Velocity Triangle

This application further offers a visualization of the inlet and outlet velocity triangle.

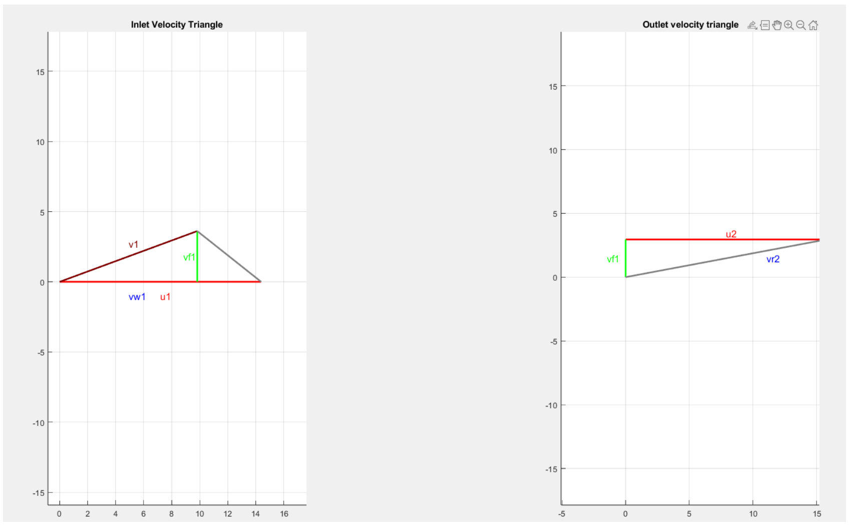


Figure 4.3: Inlet and Outlet Velocity Triangles

4.4 Axial View

This program calculates the axial view of the blades and the streamlines show the path of fluid flow during the operation of the turbine.

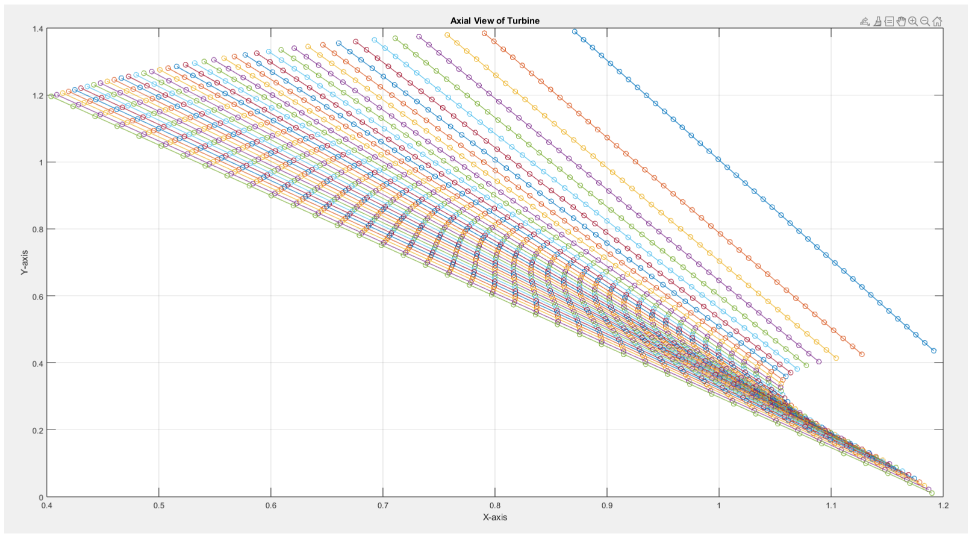


Figure 4.4: Axial View

4.5 Perpendicular View

Based on the Conformal method, this program calculates the data and displays the perpendicular view of the blade of a runner.

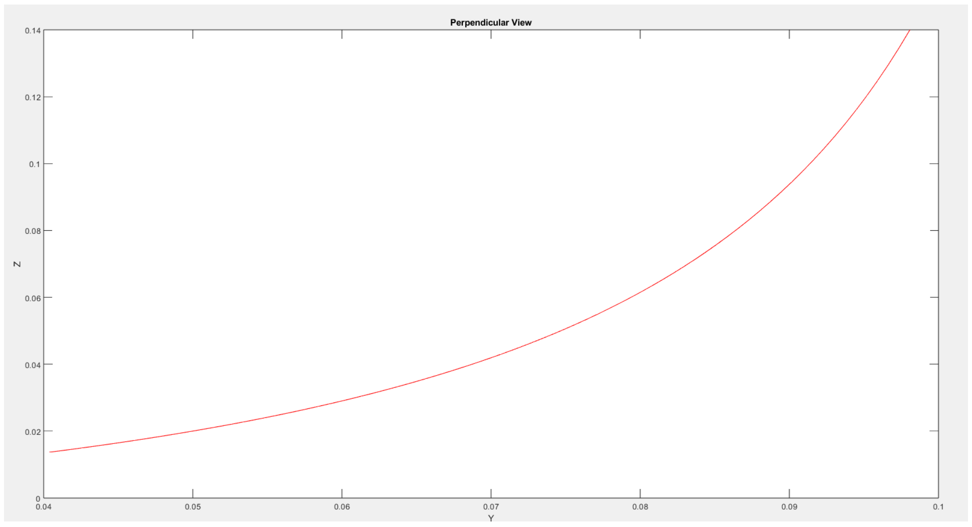


Figure 4.5: Perpendicular View

4.6 3D View of Blade and Runner

This application displays the 3D view of blades and also arranges the blade in circular arrays in order to give insight into what a runner looks like.

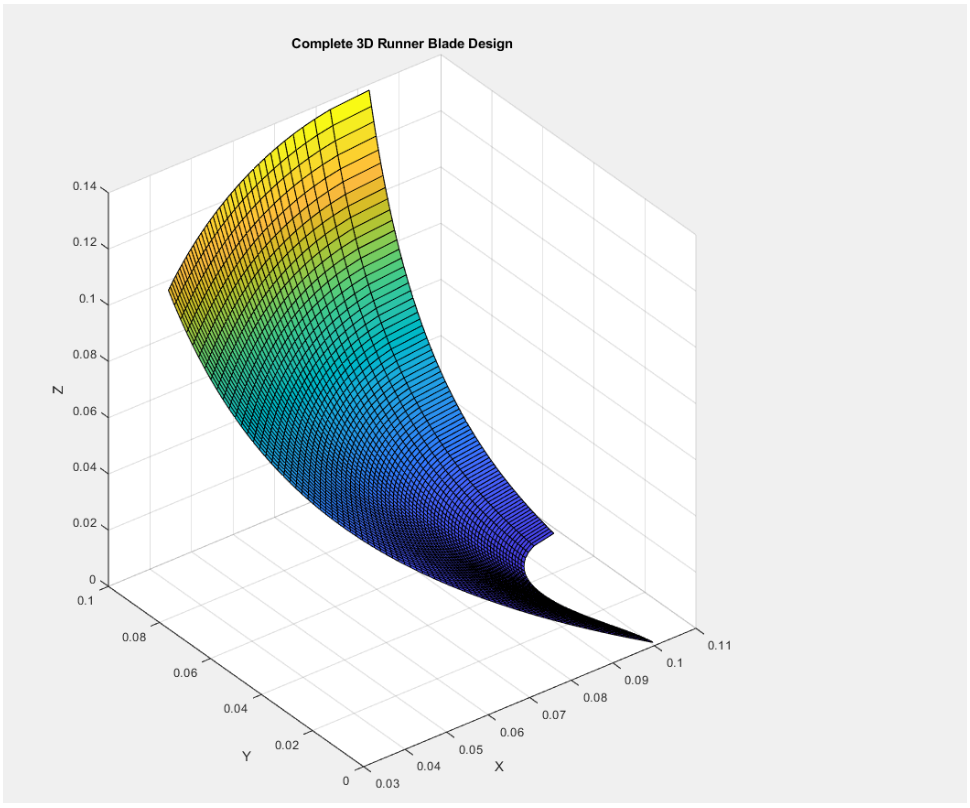


Figure 4.6: Blade 3D View

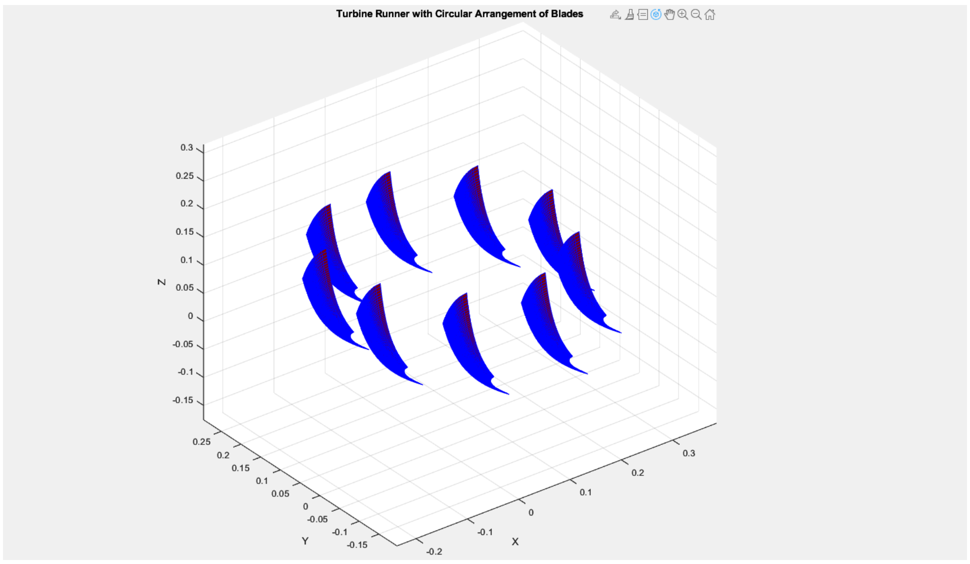
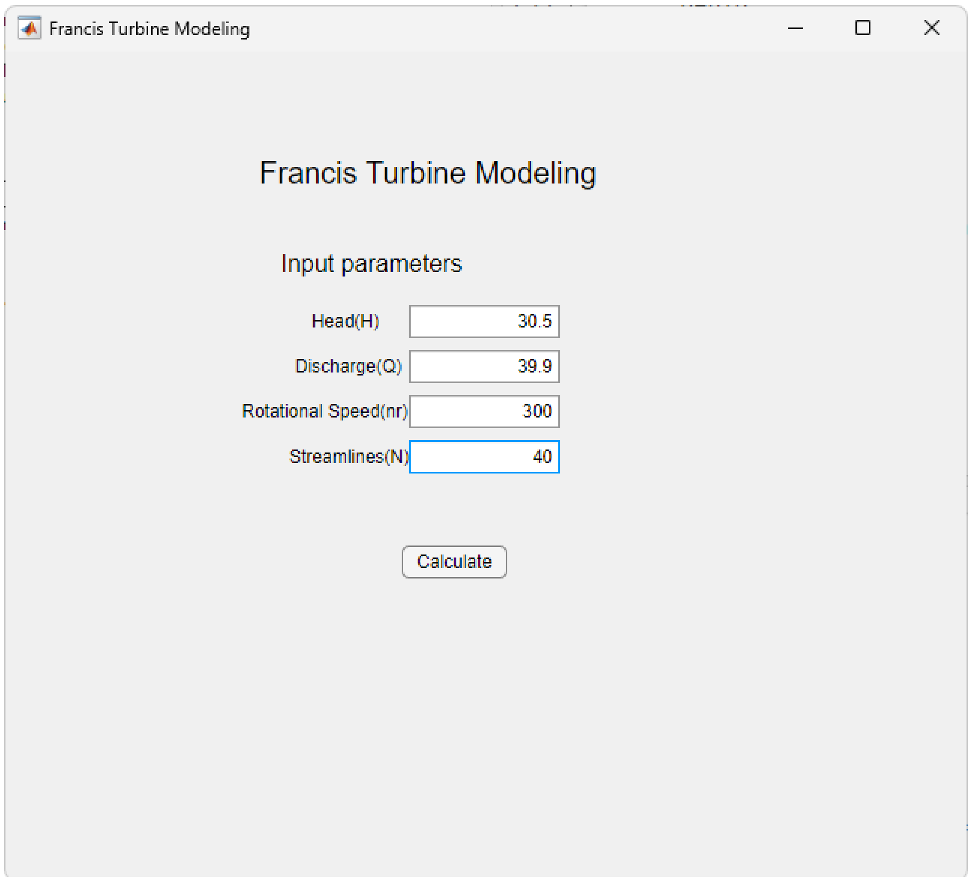


Figure 4.7: Runner 3D View

4.7 Case Of Sunkoshi Hydropower Plant

1. Head(H) = 30 m
2. Discharge rate (Q) = 39.9 m^3/s
3. Speed (nr) = 300 rpm
4. No of blades = 13



Francis Turbine Modeling

Input parameters

Head(H)	30.5
Discharge(Q)	39.9
Rotational Speed(nr)	300
Streamlines(N)	40

Figure 4.8: Home User Interface for Sunkoshi HPP

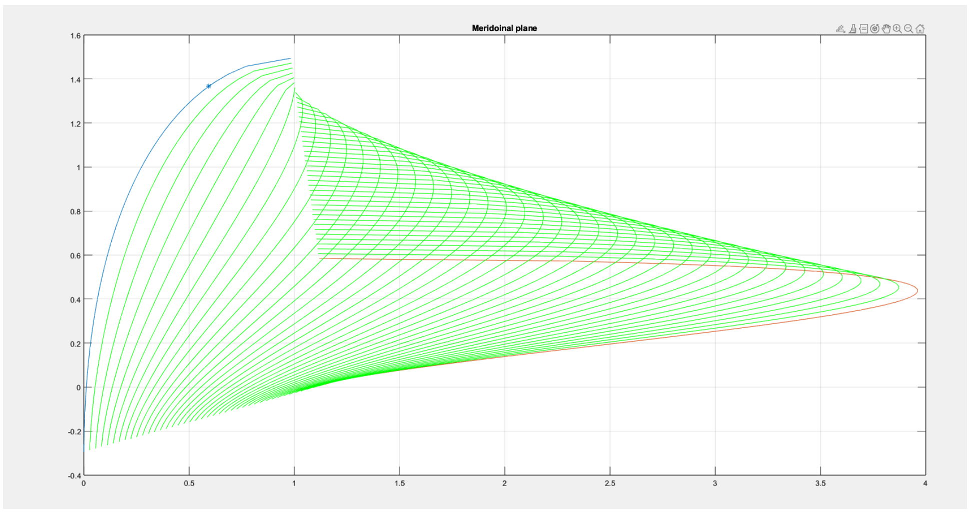


Figure 4.9: Meridional Profile for Sunkoshi HPP

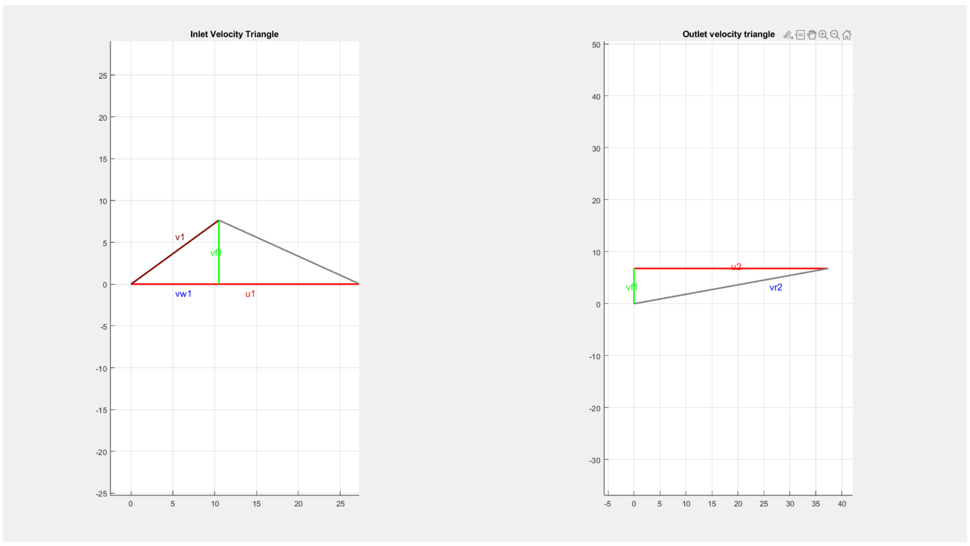


Figure 4.10: Velocity Triangles for Sunkoshi HPP

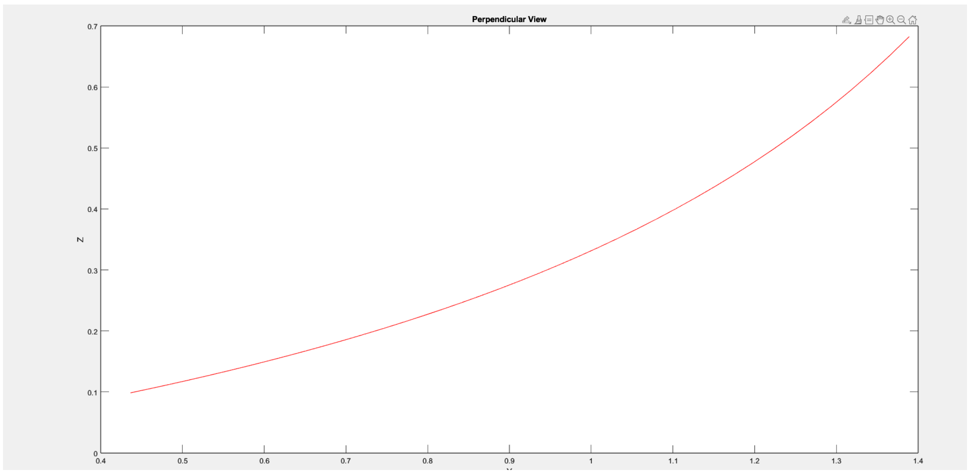


Figure 4.11: Perpendicular Profile For Sunkoshi HPP

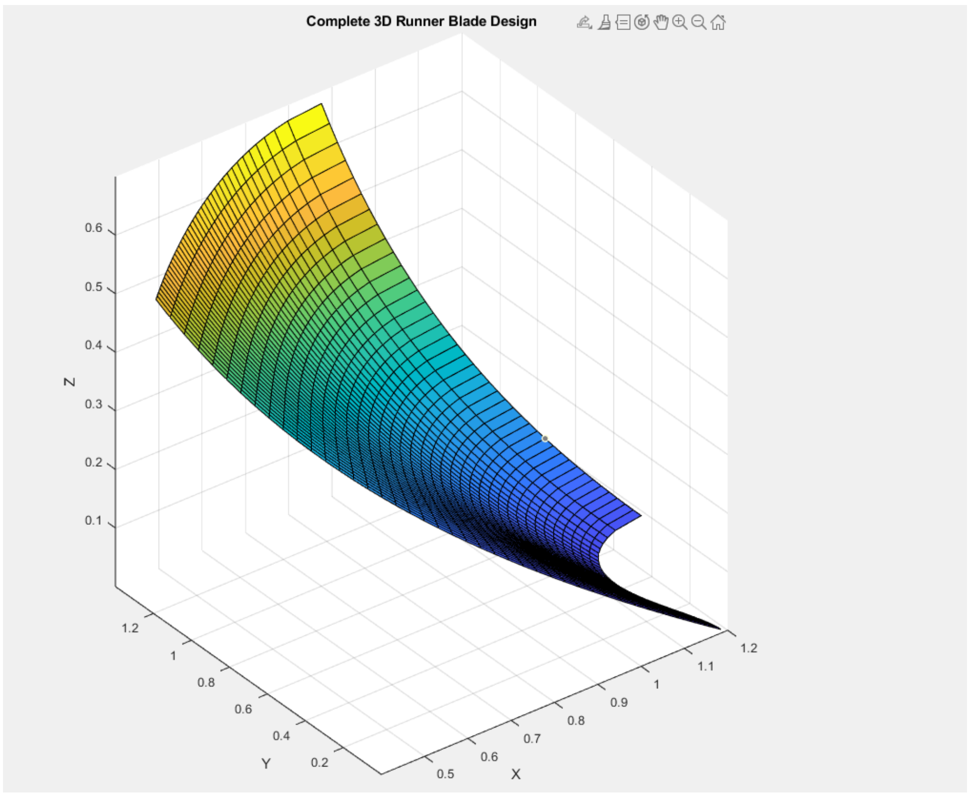


Figure 4.12: Blade 3d Views for Sunkoshi HPP

Chapter 5

Conclusion

This application implements the various methods and aspects of Francis Turbine's modeling process. A user-friendly GUI using MATLAB was developed which allows the user to input values for the design Head, flow rate, rotational speed, and number of streamlines and gives the design of the blade and runner. The code's modular structure and utilization of MATLAB's APP Designer framework allow for easy parameterization and visualization of turbine configuration. Further enhancement could include additional features such as efficiency calculation, performance analysis, integration with optimization algorithms for turbine design, and validation of design using other methods.

Bibliography

[1] Nepal Electricity Authority. “Annual report 2021-22,” Nepal Electricity Authority. (2022), [Online]. Available: `https://www.nea.org.np/admin/assets/uploads/annual_publications/Annual_Report_2021-22.pdf` (visited on 08/14/2023).

[2] B. Lewis, J. Cimbala, and A. Wouden, “Major historical developments in the design of water wheels and francis hydroturbines,” in *IOP Conference Series: Earth and Environmental Science*, IOP Publishing, vol. 22, 2014, p. 012 020.

[3] P. Breeze, “Power generation technologies. elsevier,” 2005.

[4] Z.-y. Mei, “Mechanical design and manufacturing of hydraulic machinery,” *(No Title)*, 1991.

[5] A. tech Hydro, “Low head hydro turbines,” in *Joule Centre Annual Conference: Small Hydro Power Schemes in the North West of England: Overcoming the Barriers*, 2008.

[6] A Nourbakhsh, O. S. Razavi, H Khodabakhsh, and A Mehrabadi, “New approach for hydraulic design of francis runner based on empirical correlations,” in *International Conference on Small Hydropower-Hydro Sri Lanka*, vol. 22, 2007, p. 24.

[7] B. Th, “Contribution to the study of francis–turbine runner design,” 1964.

[8] K. Gjøsæter, “Hydraulic design of francis turbine exposed to sediment erosion,” M.S. thesis, Institutt for energi-og prosessteknikk, 2011.

[9] K. Prasad Shrestha, B. Thapa, O. G. Dahlhaug, H. Prasad Neupane, N. Gurung, and A. Kayastha, “Optimized design of francis turbine runner for sand laden water.,” *Hydro Nepal: Journal of Water, Energy & Environment*, no. 13, 2013.

Appendix

```
%%main.m%%

clc
clear

Q=0.12; % in m^3/s
H=15; % in m
nrpm=1500; %in rpm
nr=2*pi*nrpm/60; %in rad per second
N=10;        %no of streamlines
n=50;                    % n is set to 50, which will determine the
   number of points in the curve.


%%%%%%% meridoinaldim.m%%%%%%
%constant
        g=9.81;
        v2eo=0.27;
        x2e=0.5;

        %dimensionless dimension of meridoinal plane
no=(nr*(Q/pi)^(0.5))/((2*g*H)^(3/4));       %dimensionless specific
    speed
bo=0.8*(2-no)*no;                            %inlet blade thickness
   (width of the blade)

%i is for the hub whereas e is for the shrub
roi=0.7+0.16/(no+0.08);
ymi=roi;
rli=0.493/(no^(2/3));
if no<0.275
    roe=0.493/(no^(2/3));
else
    roe=1.255-0.3*no;
end
li=3.2+3.2*(2-no)*no;
le=2.4-1.9*(2-no)*no;
```

```
x2e=0.5; %suppose constant
y2e=roe-1;
y2ebyyme=curve(x2e,le);
yme=y2e/y2ebyyme;
rme=roe-yme;
% Normalizing dimensions
R2e=((Q/pi)/(v2eo*nr))^(1/3);
Bo=bo*R2e;
Roi=roi*R2e;
Ymi=ymi*R2e;
Rli=rli*R2e;
Roe=roe*R2e;
Li=li*R2e;
Le=le*R2e;
X2e=x2e*R2e;
Y2e=y2e*R2e;
Yme=yme*R2e;
Rme=rme*R2e;

%%%%%meri_lines.m%%%%%

%for hub curve
x1=nan(1,n);            % NaN = create a memories for arrays which
   is not a number but a floating point values.
x2=nan(1,n);
y1=nan(1,n);
y2=nan(1,n);
p=Li/4;                  %     figure 2 ; pg no 3
q=p/n;                  %       interval length for fittting
b=0;                    % intial assumption for sum

for i=1:n
    x1(1,i)=b;
    y1=Ymi.*curve(x1,Li);          %ymi = leading edge staring
       point at hub =
    b=b+q;
end

%transforming hub curve
 x1=-x1;
 y1=-y1;
 x_trans=Le+Bo;
 y_trans=Roi;
for i=1:n
    x1(1,i)=x1(1,i)+x_trans;
    y1(1,i)=y1(1,i)+y_trans;
end
```

```
figure(1);
plot(y1,x1);
hold on;
grid on;

%%% shroud curve
p=Le;
q=p/n;
b=0;
for i=1:n
    x2(1,i)=b;
    y2=Yme.*curve(x2,Le);
    b=b+q;
end

%transforming shroud curve
x2=-x2;
y2=-y2;
x_tran=Le;
y_tran=Roe;
for i=1:n
    x2(1,i)=x2(1,i)+x_tran;
    y2(1,i)=y2(1,i)+y_tran;
end
plot(y2,x2);

% meridional view with streamline
x = nan(N,n);
y = nan(N,n);
for i=1:N
    x(i,:) = x1 + (x2-x1).*i/(N+1);
    y(i,:) = y1 + (y2-y1).*i/(N+1);
end
for i=1:N
    plot(y(i,:),x(i,:),'g')
end
title('Meridoinal plane')
%%%% Leading and trailing edge point
%%intersection of leading and trailing edge in hub and shroud curve
    respectively
x_le1=0:0.001:(Le+Bo);
y_le1=x_le1*0+Rli;
LE1=InterX([y_le1;x_le1],[y1;x1]);
plot(LE1(1,1),LE1(2,1),'*')
x_le2=0:0.001:Le+Bo;
y_le2=0*x_le2+R2e*r1e(no);
Le2=InterX([y_le2;x_le2],[y2;x2]);
```

```
[pnt,idx]=max(Le2(2,:));
LE2=Le2(:,idx)
plot(LE2(1,1),LE2(2,1),'*')
x_te1=0:0.001:Le+Bo;
y_te1=0*x_te1+R2e*r2i(no);
TE1=InterX([y_te1;x_te1],[y1;x1]);

plot(TE1(1,1),TE1(2,1),'*')
x_te2=0:0.001:Le+Bo;
y_te2=0*x_te2+R2e;
Te2=InterX([y_te2;x_te2],[y2;x2]);
[pnt,idx]=min(Te2(2,:));
TE2=Te2(:,idx);
plot(TE2(1,1),TE2(2,1),'*');

%%%Leading and trailing edge curve
xl=nan(1,n);
yl=nan(1,n);
xt=nan(1,n);
yt=nan(1,n);

%Leading edge
a=(LE1(2,1)-LE2(2,1))/((LE1(1,1)-LE2(1,1))^2);
p=LE2(1,1)-LE1(1,1);
q=(p)/(n-1);
r=LE1(1,1);
for i=1:n
    xl(1,i)=r;
    yl(1,i)=a*(xl(1,i)-LE2(1,1))^2+LE2(2,1);
    r=r+q;
end
plot(xl,yl);

%trailing edge
a=(TE1(2,1)-TE2(2,1))/((TE1(1,1)-TE2(1,1))^2);
p=abs(TE2(1,1)-TE1(1,1));
q=p/(n-1);
r=TE1(1,1);
for i=1:n
    xt(1,i)=r;
    yt(1,i)=a*(xt(1,i)-TE2(1,1))^2+TE2(2,1);
    r=r+q;
end
plot(xt,yt)
xlabel('r');
ylabel('z');
hold off;
```

```
%%%%% curve.m%%%%
function y= curve(x,l)
x=x./l;
y=3.08.*(1-x).^(3/2).*(x).^(0.5);
end

%%%%%%r1e.m%%%%%
function r1e= r1e(no)
r1e=-125.6*no^5 + 299.4*no^4 - 278.0*no^3 + 126.3*no^2 - 28.52*no +
     3.674;
end

%%%%%% r2i%%%%%%
function r2i = r2i(no)
r2i= -25.824*no^5 + 61.275*no^4 - 56.035 *no^3 + 25.096*no^2 -
    5.8833*no + 1.0977;
end


function P = InterX(L1,varargin)
%INTERX Intersection of curves
%   P = INTERX(L1,L2) returns the intersection points of two curves
     L1
%   and L2. The curves L1,L2 can be either closed or open and are
    described
%   by two-row-matrices, where each row contains its x- and y-
    coordinates.
%   The intersection of groups of curves (e.g. contour lines,
    multiply
%   connected regions etc) can also be computed by separating them
    with a
%   column of NaNs as for example
%
%         L  = [x11 x12 x13 ... NaN x21 x22 x23 ...;
%               y11 y12 y13 ... NaN y21 y22 y23 ...]
%
%   P has the same structure as L1 and L2, and its rows correspond
    to the
%   x- and y- coordinates of the intersection points of L1 and L2.
    If no
```

```
%   intersections are found, the returned P is empty.
%
%   P = INTERX(L1) returns the self-intersection points of L1. To
   keep
%   the code simple, the points at which the curve is tangent to
   itself are
%   not included. P = INTERX(L1,L1) returns all the points of the
   curve
%   together with any self-intersection points.
%
%   Example:
%       t = linspace(0,2*pi);
%       r1 = sin(4*t)+2;  x1 = r1.*cos(t); y1 = r1.*sin(t);
%       r2 = sin(8*t)+2;  x2 = r2.*cos(t); y2 = r2.*sin(t);
%       P = InterX([x1;y1],[x2;y2]);
%       plot(x1,y1,x2,y2,P(1,:),P(2,:),'ro')

%   Author : NS
%   Version: 3.0, 21 Sept. 2010

%   Two words about the algorithm: Most of the code is self-
   explanatory.
%   The only trick lies in the calculation of C1 and C2. To be
   brief, this
%   is essentially the two-dimensional analog of the condition that
    needs
%   to be satisfied by a function F(x) that has a zero in the
   interval
%   [a,b], namely
%           F(a)*F(b) <= 0
%   C1 and C2 exactly do this for each segment of curves 1 and 2
%   respectively. If this condition is satisfied simultaneously for
    two
%   segments then we know that they will cross at some point.
%   Each factor of the 'C' arrays is essentially a matrix
   containing
%   the numerators of the signed distances between points of one
   curve
%   and line segments of the other.

    %...Argument checks and assignment of L2
    error(nargchk(1,2,nargin));
    if nargin == 1,
        L2 = L1;    hF = @lt;   %...Avoid the inclusion of common
            points
    else
        L2 = varargin{1}; hF = @le;
    end

    %...Preliminary stuff
    x1  = L1(1,:)';  x2 = L2(1,:);
    y1  = L1(2,:)';  y2 = L2(2,:);
    dx1 = diff(x1); dy1 = diff(y1);
    dx2 = diff(x2); dy2 = diff(y2);

    %...Determine 'signed distances'
    S1 = dx1.*y1(1:end-1) - dy1.*x1(1:end-1);
    S2 = dx2.*y2(1:end-1) - dy2.*x2(1:end-1);

    C1 = feval(hF,D(bsxfun(@times,dx1,y2)-bsxfun(@times,dy1,x2),S1)
        ,0);
    C2 = feval(hF,D((bsxfun(@times,y1,dx2)-bsxfun(@times,x1,dy2))',
        S2'),0)';

    %...Obtain the segments where an intersection is expected
    [i,j] = find(C1 & C2);
    if isempty(i),P = zeros(2,0);return; end;

    %...Transpose and prepare for output
    i=i'; dx2=dx2'; dy2=dy2'; S2 = S2';
    L = dy2(j).*dx1(i) - dy1(i).*dx2(j);
    i = i(L~=0); j=j(L~=0); L=L(L~=0);  %...Avoid divisions by 0

    %...Solve system of eqs to get the common points
    P = unique([dx2(j).*S1(i) - dx1(i).*S2(j), ...
                dy2(j).*S1(i) - dy1(i).*S2(j)]./[L L],'rows')';

    function u = D(x,y)
        u = bsxfun(@minus,x(:,1:end-1),y).*bsxfun(@minus,x(:,2:end)
            ,y);
    end
end


Q=0.12;
H=15;
N=10;   % streamlines
nr=1500;
n=40; % no pf points

%initial calculation
P=(0.9)*(0.96)*(9810).*Q.*H;
ns=(nr.*sqrt(P.*10.^-3))/(H.^(5/4));
w=2.*pi.*nr/60;
```

```
r2=(Q/(pi.*0.24.*w)).^(1/3);
d2=2.*r2;
d1=(0.4+94.5/ns).*d2;
r1=d1/2;
dm=d2/(0.96+0.00038.*ns);
rm=dm/2;
h1=d2.*(0.094+0.00025.*ns);

if ns<111
    h2=d2.*(-0.05+42/ns);
else
    h2=d2/(3.16-0.0013.*ns);
end

b1=2.*h1;
height=h1+h2;

%%% hub and shroud
le=h2-h1;
ymi=r1;
li=b1/0.24;
yme=r2/8.711;

x1=nan(1,n);
x2=nan(1,n);
y1=nan(1,n);
y2=nan(1,n);

%%% hub
p=li/4;
q=p/n;
b=0;
for i=1:n
    y1(1,i)=b;
    x1=ymi*3.08*(1-y1/li).^(3/2).*(y1/li).^(1/2);
    b=b+q;
end

%hub point
x1=-x1;
y1=-y1;
x_trans=r1;
y_trans=h1+h2;

for i=1:n
    xx(1,i)=x1(1,i)+x_trans;
    yy(1,i)=y1(1,i)+y_trans;
```

```
end

% plot (xx,yy);
rb=xx(1,1)/3

for i=1:n
    error=xx(1,i)-rb;
    if error<0.001
        break
    end
    point=i;
end

point;
yvalue=-y1(1,point)

%hub again
p=yvalue;
q=p/n;
b=0;

for i=1:n
    y1(1,i)=b;
    x1=ymi*3.08*(1-y1/li).^(3/2).*(y1/li).^(1/2);
    b=b+q;
end

%%% shroud
p=0+le;
q=p/n;
b=0;
for i=1:n
    y2(1,i)=b;
    x2=yme*3.08*(1-y2/le).^(3/2).*(y2/le).^(1/2);
    b=b+q;
end

%offset shroud
d_leading=r1-rm;
for i=1:n
    y2(1,i)=y2(1,i)+b1;
    x2(1,i)=x2(1,i)+d_leading;
end

x = nan(N-1,n);
y = nan(N-1,n);
```

```
for i=1:N-1
    x(i,:) = x1 + (x2-x1).*i/N;
    y(i,:) = y1 + (y2-y1).*i/N;
end

a = nan(N+1,n);
b = nan(N+1,n);
c=N+1;
d=n;

for i=1:d
    a(1,i)=-x1(1,i);
    b(1,i)=-y1(1,i);
    a(N+1,i)=-x2(1,i);
    b(N+1,i)=-y2(1,i);
end

for i=2:N
    for j=1:d
        a(i,j)= -x(i-1,j);
        b(i,j)=-y(i-1,j);
    end
end

%%%% transformation of coordinates
x_trans=r1;
y_trans=h1+h2;
xx=nan(c,d);
yy=nan(c,d);

for i=1:c
    for j=1:d
        xx(i,j)=a(i,j)+x_trans;
        yy(i,j)=b(i,j)+y_trans;
    end
end

%%transform again
xtrans=r2-xx(c,d);

for i=1:c
    for j=1:d
        xx(i,j)=xx(i,j)+xtrans;
    end
end

% inlet and outlet
```

```
nh=0.96;
g=9.81;
b2=xx(c,d)-xx(1,d);
r1_new=xx(1,1);
d1_new=2.*r1_new;
d1=d1_new;
u1=(pi.*nr.*d1)/60;
vf1=Q/(pi.*d1.*b1);
vw1=(nh.*g.*H)/u1;

if vw1>u1
    beta1=atan(vf1/(vw1-u1));
else
    beta1=atan(vf1/(u1-vw1));
end

u2=(pi.*nr.*d2)/60;
vf2=Q/(pi.*d2.*b2);
beta2=atan(vf2/u2);
alpha1=atan(vf1/vw1);
vr1=(u1-vw1)/cos(beta1);
v1=vf1/sin(alpha1);
vr2=vf2/sin(beta2);

% perpendicular view
lm=nan(1,c);

for i=1:c
    lm(i)=yy(i,1)-yy(i,n);
end

disp(['lm = ' num2str(lm)]);

%plot of perpendicular view
bb=tan(beta2);
m=tan(beta1);
n=tan(beta2);
A=2*lm/(m+n);
aa=nan(1,c);

for i=1:c
    aa(i)=(m-n)/(2*A(i));
end

zz=nan(c,d);

for i=1:c
```

```
    for j=1:d
        m=aa(i);
        v=yy(i,j);
        zz(i,j)=(m.*v.^2+bb.*v);
    end
end

%%%perpendicular view
    figure;
    plot(yy(:, 1), zz(:, 1), 'r-');
    xlabel('Y');
    ylabel('Z');
    title('Perpendicular View');

%3D
%%combine view
% Combine and visualize the profiles
    figure;
% Plot combined profiles
    subplot(1, 1, 1);
    plot(yy(1, :), zz(1, :), 'b-', yy(:, 1), zz(:, 1), 'r-');
    xlabel('Y');
    ylabel('Z');
    title('Combined Meridional and Perpendicular View Profiles');
    legend('Shroud Profile', 'Perpendicular View Profile');

%%3D
    subplot(1, 2, 1);
    surf(xx, yy, zz);
    xlabel('X');
    ylabel('Y');
    zlabel('Z');
    title('Complete 3D Runner Blade Design');

% Create a new figure
    figure;

% Inlet velocity triangle
    subplot(1, 2, 1);
    hold on;

% Draw the velocities
    line([0, vw1], [0, 0], 'Color', 'b', 'LineWidth', 2);
    line([0, u1], [0, 0], 'Color', 'r', 'LineWidth', 2);
    line([0, vw1], [0, vf1], 'Color', [0.5,0,0], 'LineWidth', 2);
    line([vw1, vw1], [0, vf1], 'Color', 'g', 'LineWidth', 2);
    line([u1,vw1], [0, vf1], 'Color', [0.5,0.5,0.5], 'LineWidth',
       2);

% Draw labels
    text(u1/2, -1, 'u1', 'Color', 'r', 'FontSize', 12);
    text(vw1/2, -1, 'vw1', 'Color', 'b', 'FontSize', 12);
    if u1>vw1
        text(vw1/2, 0.75.*vf1, 'v1', 'Color', [0.5,0,0], 'FontSize'
           , 12);
        text(0.9.*vw1, vf1/2, 'vf1', 'Color', 'g', 'FontSize', 12);
        text(0.9.*u1,0.75.*vf1 , 'vr1', 'Color', [0.5,0.5,0.5], '
           FontSize', 12);
    else
        text(u1/2, 0.75*vf1, 'v1', 'Color', [0.5,0,0], 'FontSize',
           12);
        text(0.9.*u1, vf1/2, 'vf1', 'Color', 'g', 'FontSize', 12);
        text(0.9.*vw1,0.75.*vf1 , 'vr1', 'Color', [0.5,0.5,0.5], '
           FontSize', 12);
    end

% Set plot properties
    xlim([-100, 100]);
    ylim([-100, 100]);
   title('Inlet velocity triangle');

    grid on;
    axis equal;
    hold off;
    title('Inlet Velocity Triangle');

% Outlet velocity triangle
    subplot(1, 2, 2);

    %figure;
    hold on;

    % Draw the velocities
    line([0, u2], [vf2, vf2], 'Color', 'r', 'LineWidth', 2);
    line([0, 0], [0, vf2], 'Color', 'g', 'LineWidth', 2);
    line([0, u2], [0, vf2], 'Color', [0.5,0.5,0.5], 'LineWidth', 2)
       ;

% Draw labels
    text(u2/2, vf2+0.5, 'u2', 'Color', 'r', 'FontSize', 12);
    text(0.7*u2, vf2/2, 'vr2', 'Color', 'b', 'FontSize', 12);
    text(-1.5, vf2/2, 'vf1', 'Color', 'g', 'FontSize', 12);
```

```
% Set plot properties
    xlim([-100, 100]);
    ylim([-100, 100]);
    title('Outlet velocity triangle');

    grid on;
    axis equal;
    hold off;




%Define parameters
    num_blades = 10;
    radius = 0.2;
% Calculate blade positions
theta = linspace(0, 2*pi, num_blades);
xposition = radius * cos(theta);
yposition = radius * sin(theta);
zposition = zeros(size(xposition)); % Assuming blades start from
   the origin

% Create a figure
figure;
hold on;
grid on;
axis equal;

blade_x=xx;
blade_y=yy;
blade_z=zz;

% Clone and rotate blades
for i = 1:num_blades
    blade_x_shifted = blade_x + xposition(i);
    blade_y_shifted = blade_y + yposition(i);
    blade_z_shifted = blade_z + zposition(i);
    % Plot the blade surface
    surf(blade_x_shifted, blade_y_shifted, blade_z_shifted, '
       FaceColor', [0.5,0,0], 'EdgeColor', [0,0,1]);
end

% Set labels
xlabel('X');
ylabel('Y');
zlabel('Z');
title('Turbine Runner with Circular Arrangement of Blades');
```

```
% Show the plot
view(3);
rotate3d on;
hold off;




% Generate axial view plot
    ptemp = 20;
    Zmat = zeros(ptemp,N);
    Rmat = Zmat;
    x1 = linspace(d2/2, d1/2, ptemp);
    z1 = -31*x1.^3 + 50*x1.^2 - 27*x1 + 5.7;
    x = linspace(d2/2, d1/2, ptemp);
    z = 4*x.^3 - 4.7*x.^2 + 1.2*x + 0.13;
    yy = linspace(d2/2, d1/2, ptemp);
    xx20 = interp1(x, z, yy);
    xx30 = interp1(x1, z1, yy);

    for m = 1:N
        mm(:, m) = xx20 + (xx30 - xx20) * (m / N);
    end
    Zmat = mm;

    for L = 1:N
        Rmat(:, L) = yy;
    end

    c_mMat = zeros(ptemp, N);
    c_mMat(1, :) = vr1;
    dcm = vr2 - vr1;

    for P = 2:ptemp
        c_mMat(P, :) = c_mMat(P - 1, :) + dcm / (ptemp - 1);
    end

% Plotting the axial view
    figure;
    plot(Rmat(1:end, :), Zmat(1:end, :), 'b.-');
    xlim([0.4 0.8]);
    xlabel('Radius');
    ylabel('Height');
    title('Axial View Plot');
    grid on;
```