DEVELOPMENT OF SOFTWARE FOR ROTODYNAMIC ANALYSIS OF FLEXIBLE ROTOR

A Project Report Submitted for Partial Fulfilment of the Requirements for the Degree of

B. Tech.
(Mechanical Engineering)

By
NAME: Siddhartha Meher
Roll-110ME0242

Under the supervision of
Prof. Suraj Kumar Behera,
Department of Mechanical Engineering, NIT, Rourkela

Department of Mechanical Engineering
NATIONAL INSTITUTE OF TECHNOLOGY ROURKELA

NATIONAL INSTITUTE OF TECHNOLOGY
ROURKELA

CERTIFICATE

This is to certify that the project entitled "**Development of a Software for Rotodynamic Analysis Of Flexible Rotor**", being submitted by Mr Siddhartha Meher (Roll No.-110ME0242), is a record of bona fide research carried out by him at our institute, National Institute of Technology, Rourkela, under my guidance and supervision. The work incorporated in this project has not been, to the best of my knowledge, submitted to any other University or Institute for the award of any degree or diploma.

Date:

Place:

Prof. Suraj Kumar Behera
Department of mechanical engineering
NIT ROURKELA
Odisha -769008

ACKNOWLEDGEMENT

It provides me gigantic joy to express my profound feeling of appreciation to my supervisor**Prof. S.K. Behera**, for his significant direction, inspiration, consistent spark or more just for his ever co-working demeanour that empowered me in raising this project in the present structure. It was the shot of a life time to interact with an identity that is a perfect mix of head and heart, dynamism and delicacy, wit and gravity with judiciousness and curiosity. I record my unspeakable appreciation for him.

I am also grateful to **Prof. K. P. Maity**, Head, Department of Mechanical Engineering and **Prof. S. K. Sahoo and Prof. S. C. Mohanty,** Course Coordinator for their help and advice during the course of this work.

I am greatly thankful to my well-wishers, class mates and friends for their inspiration, help and support. Finally to my parents for without their constant blessing nothing would have been possible.

Date:

Place:

Siddhartha Meher
Roll-110ME0242
B.Tech (Mechanical Engineering)
NIT ROURKELA
Odisha -769008

ABSTRACT

Rotating shafts are employed in industrial machines such as steam and gas turbines, cryogenic turbo expanders, turbo generators, internal combustion engines, centrifugal compressors for power transmission. On account of the ever increasing demand for power and high speed transportation, the rotors of these machines are made extremely flexible to allow bending. The shafts of these machines are subjected to torsional and bending vibration and in some cases unstable condition of operation. These machines can develop excessive stress in torsion because of low torsional natural frequencies of the system involving flexible couplings. Determination of natural frequencies and mode shapes is thus important from design point of view. Lateral bending and subsequently whirling of a rotor may arise due to residual unbalance (present due to corrosion, material inhomogeneity etc.) which would cause the simple circular shaft on rigid supports carrying concentrated masses experience synchronous whirl and when the shaft runs at a speed equal to its natural frequency in lateral bending, the whirling becomes predominant. This phenomenon is different from conventional resonance since large amplitude vibration cannot be controlled by additional damping. As the resonance occurs the shaft develops stresses violently, causing the shaft to fail suddenly or decreasing its life. Considering the importance of critical shape and unbalance responses, an effort is made to develop interactive and user-friendly software capable of evaluating the much needed data like critical speed, the associated mode shapes and unbalance-response of a given rotor.

CONTENTS

NOMENCLATURE

C: Damping coefficient

D: Shaft diameter(mm)

E: Modulus of elasticity (N/m^2)

e: Journal eccentricity (mm)

I: Area moment of inertia (N/mm^2)

I_P**:** Polar mass moment of inertia (N/mm^2)

I_T**:** Transverse mass moment of inertia (N/mm^2)

K_b**:** Bearing stiffness (N/mm)

L: Length of rotor station (mm)

m: Mass of rotor element (gm)

***M*, M'**: Bending moment (N-mm)

***r, r'*:** Displacement (mm)

***V,V'*:** Shear forces (N/mm^2)

x,y,z: Coordinates

ω: Rotor speed (rad/s)

[*X*]: A square matrix, e.g. the overall transfer matrix [*To*]

n: Rotor station for transfer matrix analysis

LIST OF FIGURES

CHAPTER - 1

Introduction

In industrial machines such as internal combustion engines, steam and gas turbines, cryogenic turbo expanders, centrifugal compressors rotating shafts are used for power transmission. The occurrence of bending vibration and critical speeds of rotating shafts is perhaps the most common annoyingregular problem in design and maintenance of the machinery. As the demand for power and high speed transportation is growing every day, the rotors of these machines are made extremely flexible to allow bending. Some of the rotors are as high as 100 tons for big steam turbines and obviously they warrant utmost attention in this regard. The shafts of these machines are subjected to torsional and bending vibration and in some cases to unstable condition of operation. These machines sometimes develop excessive stress in torsion due to low torsional natural frequencies of flexible couplings. The rotors have a little amount of residual unbalance however well they are balanced, and this cause resonance if they start rotating at speeds equal to bending natural frequency. These speeds are called as critical speeds and as far as possible they should be avoided. Even while taking the rotor through a critical speed to an operational speed, special precaution should be taken. The problem to find out the bending natural frequency of a simple shaft becomes complex due to different moments of area of shaft, stiffness and damping properties of bearing, coupling between two rotors etc.

Because of the residual unbalance excessive vibration in rotating shafts occur. The unbalance in the rotor due to material inhomogeneity, manufacturing processes, slots etc. can be removed by proper balancing procedure. However, during the operation the rotor deteriorated in its balance condition, due to wear, thermal bending, process dirt collection etc., and gradually develops more vibratory response due to this unbalance. Rotor unbalance considered does not only cause vibration, it also transmits rotational forces to rotor bearing and to the supporting structure. The forces thus transmitted may change the machine and shorten its working life. Rotor unbalance considered does not only cause vibration, it also transmits rotational forces to rotor bearing and to the supporting structure. The forces thus transmitted may change the machine and shorten its working life.

To calculate the critical speed, mode shape and unbalance response theoretically, over the year many researches have been carried out. However the influence coefficients technique proves to be correct with reasonable accuracy. It was presented by Goodman, since then it has been improved and tested by several authors such as Lund and Tonneson, Tessarzik and Badgley, Everett. The method used is Transfer Matrix Method (TMM). In this paper to calculate the above mentioned data, the TMM is employed.

Considering the importance of the critical speed, mode shape and unbalance response of an imbalanced rotor, we have decided to develop intuitive, user friendly software to find the above data for any flexible rotor. The proposed software is developed using Matlab.

Aim of the Work

Development of user friendly, intuitive software to find critical speeds, associated mode shapes and unbalance response for flexible rotors with multiple disks.

CHAPTER - 2

Literature survey

For every rotor-bearing system there exist an infinite number of discrete natural frequencies of lateral vibration. Associated with each natural frequency is a mode shape, which can be conceived as a snapshot of the rotor deflection curve at the instant of maximum strain during the vibration. When there exists an externally impressed periodic force with a frequency equal to one of the natural frequencies, the vibration amplitude increases significantly. In a rotating shaft, such periodic forces can arise due to a variety of causes, one among them being the shaft imbalance. The corresponding speed of rotation of the shaft is called a critical speed. At the critical speed, the rotor does not vibrate, but rather is bowed into the mode shape associated with the particular natural frequency, and whirls about its bearing centreline. Although theoretically there exists infinite number of natural frequencies, in practice, only the first three or four are excited, which forms the region of interest. The mode shapes are determined by the distribution of mass and stiffness along the rotor, together with the bearing stiffness. The first two modes are the rigid body (cylindrical or conical) modes and are heavily influenced by the bearing stiffness. From third mode (first bender) upwards the influence of bearings wane and the rotor geometry and mass distribution come into picture. Most of the turbo-machines to date run at speeds much lower than the first bender, because at speeds higher than this limit, the rotor becomes flexible, the clearances change and the balancing gets disturbed. This is especially true for high-speed cryogenic turbo-expander rotors and ours is no exception. It is not possible to design gas bearings of adequate stiffness to raise the frequencies of the first two critical speeds beyond the operating speed of the rotor. Therefore, they must be crossed during start-ups and shut downs; and adequate damping must be provided in the bearings to limit the vibration amplitude during this period. The third critical speed or the first bender, however, must be studied analytically and the shaft must be designed to maintain this critical speed above the operating speed of the rotor.

The amplitude of vibration due to inherent imbalance in the rotor increases with speed, reaching a maximum when the operating speed equals a system natural frequency. To prevent damage from uncontrolled vibration, the system damping can be increased, or better still, the rotor can be precisely balanced. By precise balancing, the residual unbalance should be lower than the limit set for the kind of use the rotor is put to.

There are several theoretical method available for the analysis. Dunkerley[1] proposed a very simple method do the calculation. Here it reduces the actual system into a number of simple subsystems, each of the systems critical speed is calculated and then their square's inverse are added to get the square of inverse of overall critical speed. It's a lower bound approximation. Rayleigh's method is also quite popular and simple for such analysis. According to this method maximum kinetic energy must be equal to maximum potential energy for a conservative system under free vibration condition. This method gives upper bound approximation. Now of all the available methods for computation of critical speeds (natural frequencies) and imbalance response, the transfer matrix method is one of the most popular, easy to use and one requiring lower computer run time (than comparable methods) due to the presence of a maximum of 17x17 matrix[2]. We have chosen this method for computing the critical speeds as well as unbalance response for our prototype cryogenic turbo expander rotor.

CHAPTER - 3

3.1 Software development

3.1.1 Life cycle model

A software life cycle model (also called process model) is a descriptive and diagrammatic representation of the software life cycle. A life cycle model represents all the activities required to make a software product transit through its life cycle phases. It also captures the order in which these activities are to be undertaken. In other words, a life cycle model maps the different activities performed on a software product from its inception to retirement. Different life cycle models may map the basic development activities to phases in different ways. Thus, no matter which life cycle model is followed, the basic activities are included in all life cycle models though the activities may be carried out in different orders in different life cycle models. During any life cycle phase, more than one activity may also be carried out. For example, the design phase might consist of the structured analysis activity followed by the structured design activity.

3.1.2 Different software life cycle models

1. Many life cycle models have been proposed so far. Each of them has some advantages as well as some disadvantages. A few important and commonly used life cycle models are as follows:
2. Classical Waterfall Model
3. Iterative Waterfall Model
4. Prototyping Model
5. Evolutionary Model
6. Spiral Model

3.1.3 Iterative Waterfall Model

An iterative Waterfall life cycle model does not attempt to start with a full specification of requirements. Instead, development begins by specifying and implementing just part of the software, which can then be reviewed in order to identify further requirements. This process is then repeated, producing a new version of the software for each cycle of the model. Figure 2.1 shows work flow of iterative waterfall model.

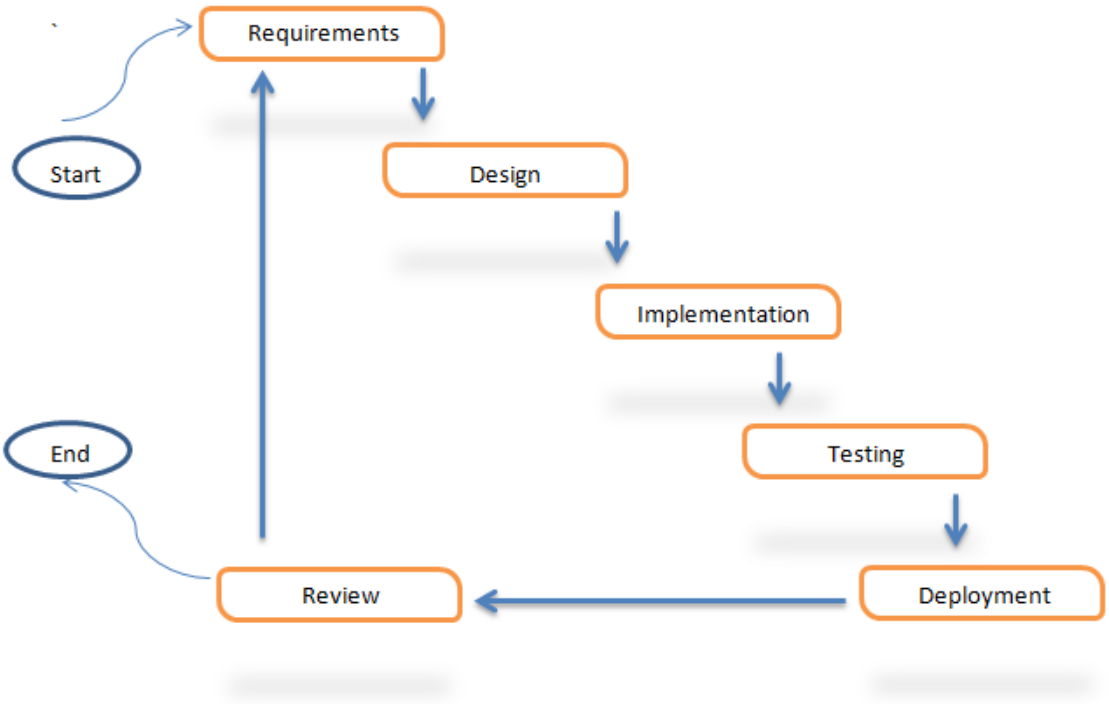


Fig 3.1: Iterative waterfall model- work flow

3.1.3.1 Advantages:

1. In iterative waterfall model one can only create a high-level design of the application before developers actually begin to build the product and define the design solution for the entire product. Later on developers can design and built a skeleton version of that, and then evolved the design based on what had been built.
2. In iterative model developers are building and improving the product step by step. Hence they can track the defects at early stages. This avoids the downward flow of the defects.
3. In iterative model developers can get the reliable user feedback. When presenting sketches and blueprints of the product to users for their feedback, they are effectively asking them to imagine how the product will work.
4. In iterative model less time is spent on documenting and more time is given for designing.

3.1.3.2 Disadvantages:

1. Each phase of iteration is rigid with no overlaps.
2. Complex system architecture or design issues may arise because not all requirements are gathered up front for the entire lifecycle.

3.1.3.3 When to use iterative model:

- Requirements of the complete system are clearly defined and understood.
- When the project is moderate or big.
- Major requirements must be defined; however, some details can evolve with time.

3.1.4 Why this model is chosen?

As the objective of the paper is very clear, the requirement of the software and its functionality is clearly defined and well understood. As it was a moderate size product, a step by step evolving design based on what was previously built was proposed. As it was improved step by step the defects were caught at early stages. At the end of the each iteration, from the then finished product feedback can be taken and in the semi-finished software, the user can interact with it and understand how it would work. Some minor features can be embedded at the later stage of development also.

3.1.5 Steps:

The requirements were broken into small, manageable, feature oriented parts. In the 1st iteration only the critical speed and associated mode shape calculations were considered and an appropriate user interface was also developed. The backend functions were also written using the designed algorithm. In the 2nd iteration an evolved user interface was designed but it was discarded later because of its poor intuitive interactivity. At the same time backend code for the unbalance response was written using the appropriate algorithm. In the 3rd iteration another user interface was made which included more features like an interactive data table where the user can change the rotor data on spot, a rotor preview so that the user can get the sense of what kind of rotor he is interacting. The code written in the previous iterations were successfully integrated with it. In the 4th iteration even more functionalities were added, like the plot controller (grid on/off, legend on/off with formatting options, title, xlabel, ylabel can also be added, manual scaling etc.). The user was disabled from direct interaction with the rotor data, giving the user indirect and effective ways. A generate function was developed which can build the 3d model of the rotor automatically with the click of a button.

3.2 Requirements:

3.2.1 1st iteration

To begin the project with, 1st as far as the user interface is concerned, it had to be a very simple one containing only the very basic functionalities like the a rotor data table, a calculate button to trigger the calculation which would be executed in the backend, some text fields to display all the critical speeds found, a plot which would show the mode shapes those were associated with the critical speed. The program to calculate the critical speed and associated mode shapes were also written.

3.2.2 2nd iteration

In this iteration, a user interface was developed which would have the 3 sections.

1. Rotor definition
2. Critical speed and mode shape calculation
3. Unbalance response

The above mentioned 3 sections were built on 3 different forms which the user can toggle between.

3.2.2.1 Rotor definition

In this section the rotor profile was defined. The rotor definition section contained 2 sub parts.

i) A field where user can enter an 'xls' file having the station data of rotor.
ii) Define the rotor in the form itself.

3.2.2.2 Critical speed and mode shape calculation

In this section rotor a pop up menu was introduced where the user can specify the number of critical speed he can calculate. Some text fields where the critical speed would be displayed. A save button to save the mode shapes. Some radio buttons to select which mode shape to be displayed.

3.2.2.3 Unbalance response

In this section, unbalance response could be calculated. A calculate button which would trigger a function execution which would calculate the unbalance at all the stations. A popup menu drop box from which the user can select a station whose unbalance response he wants to find and a plot button which would create the plot.

3.2.3 3rd Iteration

As the product obtained at the end of 2nd iteration was not so intuitive, it was decided to converge all the sections into one form, where user can have access to all the area at once and every data would be at the user's glance. Some other features were also added for better user interactivity.

1. A unified user interface was made.
2. In the Rotor definition section save option was introduced whose function was to save the rotor data into an 'xls' file.
3. In the Critical speed and mode shape calculation section a compare radio button was added so that the user can compare any two mode shapes corresponding to their critical speeds.
4. Similarly in the Unbalance response section another compare radio button was added so that the user can pick any two stations to compare their unbalance response.
5. A plot control section was also added which would enable the user to control various aspects of the plot like title, x label, y label, grid on/off, scaling, etc.

3.2.4 4th iteration

The product obtained at the end of the 3rt iteration was working fine but there were some input related issues in the rotor definition section and also the plot window was small for preview purpose. There was no legend control option. As the interface was big the a dedicated hint section was also needed.

1. The rotor definition section was made more constrained so as to prevent the user from entering inane values.
2. The plot controller was improved and legend controls were also introduced.

3. As the user interface became larger with many controls, to provide the user directionality a dedicated hint section was also introduced.

3.3 MATLAB Environment

3.3.1 Introduction

MATLAB (matrix laboratory) is a multi-paradigm numerical computing environment and fourth-generation programming language. Developed by MathWorks, MATLAB allows matrix manipulations, plotting of functions and data, implementation of algorithms, creation of user interfaces, and interfacing with programs written in other languages, including C, C++ and Java. MATLAB is a high-performance language for technical computing. It integrates computation, visualization, and programming environment. Furthermore, MATLAB is a modern programming language environment: it has sophisticated data structures, contains built-in editing and debugging tools, and supports object-oriented programming. These factors make MATLAB an excellent tool for research, engineering application development and teaching.

CHAPTER - 4

Numerical Analysis

4.1 The transfer matrix method

The transfer matrix method represents a general flexible rotor by a number of lumped inertias (discs) connected by mass less elastic shaft sections. The number of discs ranges from 8 to 80. Point and field matrices, for the inertia and the shaft elements respectively, are multiplied together to give expressions for the boundary conditions at the right end of the rotor in terms of the boundary conditions at the left end. The multiplication of the coefficients of all the matrices finally gives a resultant matrix, known as the ‘overall transfer matrix’. The elements of this matrix are determined, in part, by the natural frequencies or eigen values of the system.

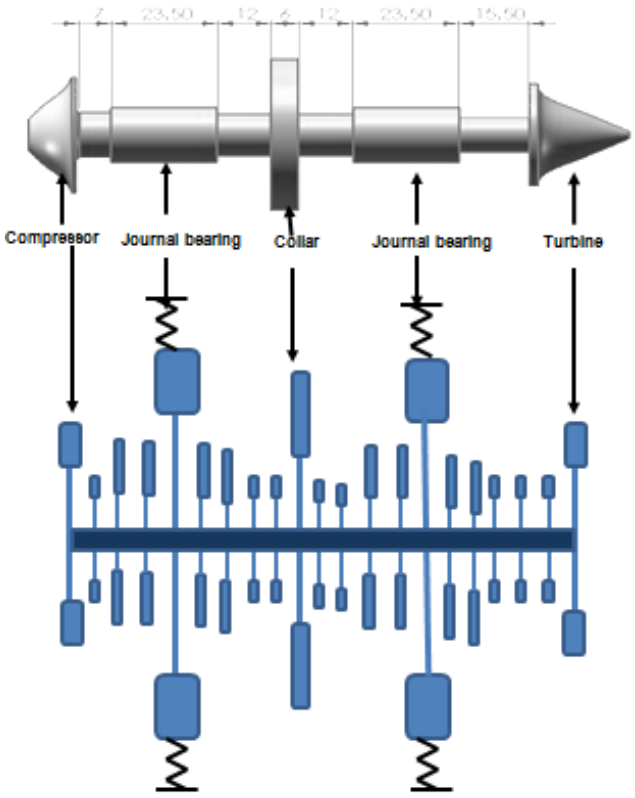


Fig 4.1: A rotor model with bearing, collar, turbine and compressor

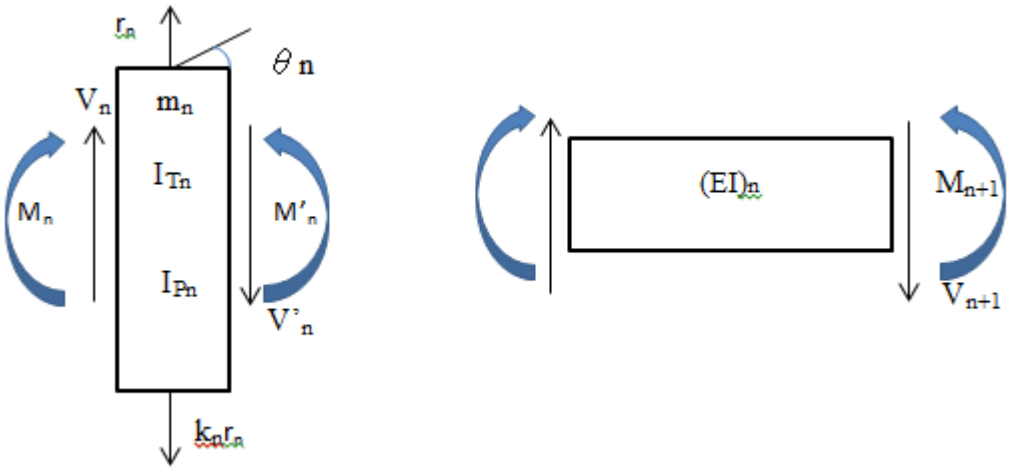


Fig 4.2 : Free-body diagram of the elements in the nth rotor station

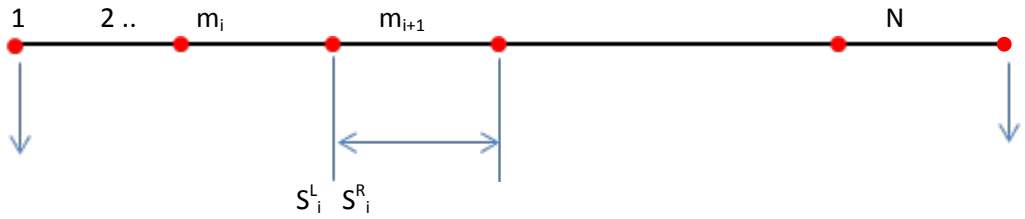


Fig 4.3: Station Distribution and Nomenclature

4.2 Critical speed analysis

The process of computation of critical speeds (natural frequencies) can be explained easily by considering an undamped, symmetric rotor-bearing system with a pair of bearings of similar stiffness Kb . The rotor system is modelled into N numbers of stations as shown in figure 4.1.where N denotes the last inertia station at the right end of the rotor. Figure 4.2 shows the free body diagrams of the inertia (disc) and the shaft (elastic) elements constituting the nth station. With X-Y symmetry, the disc displacements can be expressed well in the r and θ coordinate system.

Consider an n mass system, each of the mass representing either a gear, a disk, or a flywheel etc. All these masses are taken as lumped with their gyroscopic inertia neglected. The ith shaft element of length l_i and mass m_i are separately shown, and {S} represents the state vector containing the deflection w, slope 'θ', bending moment M_y and shear force Vz, for bending in the x-z plane. The state vector can be defined to the left or right in the positive z and y direction. V_z and M_y represent the internal forces and their positive signs are along z ad

y direction for a positive face. A positive face is defined as one which has its outward normal in the x-x direction. On a negative face V_x and M_y are positive in the negative direction of z and y axes respectively. For the shaft element and inertia element, Field matrix and point matrix are defined respectively.

4.2.1 Field matrix

From the equilibrium relations for the i th field

$$\left.\begin{aligned} v_{zi}^{L} &= v_{z,i-1}^{R} \\ M_{yi}^{L} &= M_{y,i-1}^{R} + v_{zi}^{L} l_i \end{aligned}\right\} \tag{4.1}$$

To derive the transfer relations for deflection w and slope θ, for the element shown in above figure, the relations for a cantilever beam is used,

$$\left.\begin{aligned} w &= -\frac{Ml^2}{2EI} + \frac{Vl^3}{3EI} \\ \theta &= \frac{Ml}{EI} - \frac{Vl^2}{2EI} \end{aligned}\right\} \tag{4.2}$$

Using the above, the following relations for deflection w and slope θ to the left of station I, in terms of the corresponding quantities to the right of station i-1 is obtained.

$$\left.\begin{aligned} w_i^{L} &= w_{i-1}^{R} - \theta_{i-1}^{R} l_i - M_{yi}^{L} \frac{l_i^2}{2EI_i} + v_{zi}^{L} \frac{l_i^3}{3EI_i} \\ \theta_i^{L} &= \theta_{i-1}^{R} + M_{yi}^{L} \frac{l_i}{EI_i} - v_{zi}^{L} \frac{l_i^2}{2EI_i} \end{aligned}\right\} \tag{4.3}$$

On solving the above equation and simplifying,

$$\left.\begin{aligned} -w_i^{L} &= -w_{i-1}^{R} + \theta_{i-1}^{R} l_i + M_{y,i-1}^{R} \frac{l_i^2}{2EI_i} + v_{z,i-1}^{R} \frac{l_i^3}{6EI_i} \\ \theta_i^{L} &= \theta_{i-1}^{R} + M_{y,i-1}^{R} \frac{l_i}{EI_i} + v_{z,i-1}^{R} \frac{l_i^2}{2EI_i} \end{aligned}\right\} \tag{4.4}$$

All the above equation can be combined to form the following transfer matrix relation.

$$\begin{Bmatrix} w \\ \theta \\ V_x \\ M_y \end{Bmatrix}_i^L = \begin{bmatrix} 1 & l_n & \frac{l_n}{6E_nI_n} & \frac{l_n^2}{2E_nI_n} \\ 0 & 1 & \frac{l_n^2}{2E_nI_n} & \frac{l_n}{2E_nI_n} \\ 0 & 0 & 1 & 0 \\ 0 & 0 & l_n & 1 \end{bmatrix}_i \begin{Bmatrix} w \\ \theta \\ V_x \\ M_y \end{Bmatrix}_{i-1}^R \tag{4.5}$$

Symbolically the above equation can be written as

$$\{S\}_i^L = [F]_i \{S\}_{i-1}^R \tag{4.6}$$

where {S} is defined as state vector and $[F]_i$ is the transfer matrix for the i th field.

4.2.2 Point Matrix

The figure shows the equilibrium relations for the mass at station i, so the inertia matrix can directly be written.

$$\begin{Bmatrix} w \\ \theta \\ V_x \\ M_y \end{Bmatrix}_i^R = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ m_n w^2 - k_n & 0 & 1 & 0 \\ 0 & (I_{Tn} - I_{Pn})\,\mathrm{w}^2 & 0 & 1 \end{bmatrix}_i \begin{Bmatrix} w \\ \theta \\ V_x \\ M_y \end{Bmatrix}_i^L \tag{4.7}$$

Symbolically the above equation can be written as

$$\{S\}_i^R = [P]_i \{S\}_i^L \tag{4.8}$$

The above [P] matrix is the point matrix for the ith mass rotating with speed w.

4.2.3 Overall transfer matrix for and frequency equation

Starting from station 1st element of the shaft, the following equation can be written.

$$\begin{aligned} &\{S\}_1^L = [F]_1 \{S\}_0 \\ &\{S\}_1^R = [P]_1 \{S\}_1^L = [P]_1 [F]_1 \{S\}_0 \\ &\{S\}_2^L = [F]_2 \{S\}_1^R = [F]_2 [P]_1 [F]_1 \{S\}_0 \\ &\vdots \\ &\{S\}_n = [F]_n [P]_{n-1} [F]_{n-1} [P]_{n-2} \cdots [F]_1 \{S\}_0 \end{aligned} \tag{4.9}$$

After multiplying all the transfer matrix and point matrix in the given order, the matrix found is defined as overall transfer matrix [U], so

$$\{S\}_{n+1}=[T_o]\{S\}_0 \qquad (4.10)$$

Substitution of the left end boundary conditions $V_1 = 0$ and $M_1 = 0$, gives the right end boundary conditions as:

$$\begin{Bmatrix} V'_N \\ M'_N \end{Bmatrix} = \begin{Bmatrix} 0 \\ 0 \end{Bmatrix} = \begin{bmatrix} d_{11} & d_{12} \\ d_{21} & d_{22} \end{bmatrix} \begin{Bmatrix} r_1 \\ \theta_1 \end{Bmatrix} = [D] \begin{Bmatrix} r_1 \\ \theta_1 \end{Bmatrix} \qquad (4.11)$$

The elements d_{ij} of the square matrix [D] in equation depend on ω. For nonzero solutions to r1 and θ , the determinant of the matrix [D] is zero. A critical speed can thus be obtained by starting from an initial guess value of ω and increasing it in steps till the value of the determinant D vanishes within a specified tolerance. The next higher critical speed can be determined by repeating the process, starting with a higher initial value. The eigenvectors (mode shapes) are determined by multiplication of the transfer matrices to compute the displacements at each station.

4.3.4 Programming for bending critical speed

In this part how the program code was written is documented. After going through numerous literatures of the topic a generalised procedure was found [2]. It was then developed into an algorithm suitable for computer programming. According to the algorithm a flow chart was drawn (Figure 4.5), so that it would be easy to understand for the new users that how the program works. Its other purpose was that in the program, the variables scope, life, function and over all behaviour of each object can fully be understood. This saved time and unnecessary coding.

4.3.5 Algorithm according to the procedure

1. Input the numerical values of the inertial parameters at all stations.

2. Input a starting (guess) value for the natural frequency, ω.

3. Calculate the elements of transfer matrices at all the stations.

4. Start from the left end matrix, substitute r_1= 1, θ_1 = 0, V_1= 0 and M_1=0 to get r'$_1$ θ_1, V_1 and M_1'. Use this in the next matrix to obtain r_2, θ 2, V_2 and M_2, and so on, till we get V_N' and M_N'. Since we had taken r_1=1 and θ_1 =0, our computed V_N' and M_N' give d_{11} and d_{21} respectively.

5. Using the same procedure as above, but setting, r_1=0 and θ_1 =1, compute V_N' and M_n' to obtain d_{12} and d_{22}.

6. Now all the elements of the matrix [D] are computed with the given input frequency, ω. Calculate the value of the determinant, and check if it is within the acceptable tolerance limits, print $\omega_{cr} = \omega$, ω_{cr} being the critical speed. If it is not within acceptable limits, increment ω to a new value and return to step 3.

7. With ω_{cr} known, θ_1 is computed for r_1=1. Successive matrix multiplications give the displacement at all the rotor stations, r_2, r_3… r_n. Thus the mode shape is determined for the computed critical speed normalised with respect to r_1=1.

8. The next higher critical speed is determined in the same manner, with a new starting value of ω higher than the last computed critical speed, ω_{cr}.

4.3.7 Code:

The code was written in MATLAB programming language. The algorithm was used and the variables scope and life were according to the flow chart. After writing the code it was tested using various rotor data. All gave expected result. Hence we can use this code to calculate any rotor's critical speed and mode shape.

4.4 Analysis of unbalance response:

4.4.1 Overview

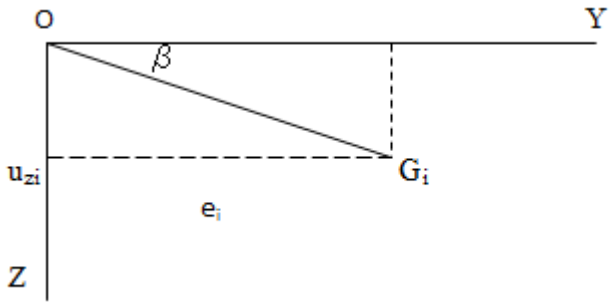


Fig 4.4: The i^{th} unbalance mass

The process of computation of unbalance response can be seen as an extension of critical speed analysis. The amplitude of vibration, especially in the proximity of a critical speed, is highly dependent on system damping. In the absence of damping, vibration amplitudes can increase indefinitely, leading to catastrophic failure of bearings and/or rotors. In the model shown in Fig 4.1, we add damping at the bearings along with an unbalance mass m_u at an eccentricity e and phase angle β. The unbalance, if previously measured, can be specified at all the rotor stations (i.e. an unbalance distribution) or at particular rotor stations at the discretion of the analyst.

From the free body diagram of Fig 4.2, after inclusion of the unbalance mass and the damping parameters, with *X*-*Y* asymmetry, the equations for the shear forces [2] can be written as

$$V'_{ycn}=V_{ycn} + (m_n w^2-k_n)y_{cn} - cwy_{sn} + u_y \quad (4.12)$$

$$V'_{ycn}=V_{ycn} + (m_n w^2-k_n)y_{cn} - cwy_{cn} + u_x \quad (4.13)$$

$$V'_{xcn}=V_{xcn} + (m_n w^2-k_n)x_{cn} - cwy_{sn} + u_x \quad (4.14)$$

$$V'_{xsn}=V_{xsn} + (m_n w^2-k_n)x_{sn} - cwy_{cn} + u_y \quad (4.15)$$

Where V'_{ycn},V'_{ycn},$V'_{xcn}$and V'_{xsn} are the cosine and sine component of shear forces at left of the disc, and V_{ycn},V_{ycn},V_{xcn} and V_{xsn} are those at the right. C is the damping coefficient. The unbalance forces are

$$U_x=m_u\, e\, \omega^2 \cos\beta \quad (4.16)$$

$$U_y=m_u\, e\, \omega^2 \sin\beta \quad (4.17)$$

4.4.2 Programming for Unbalance responses

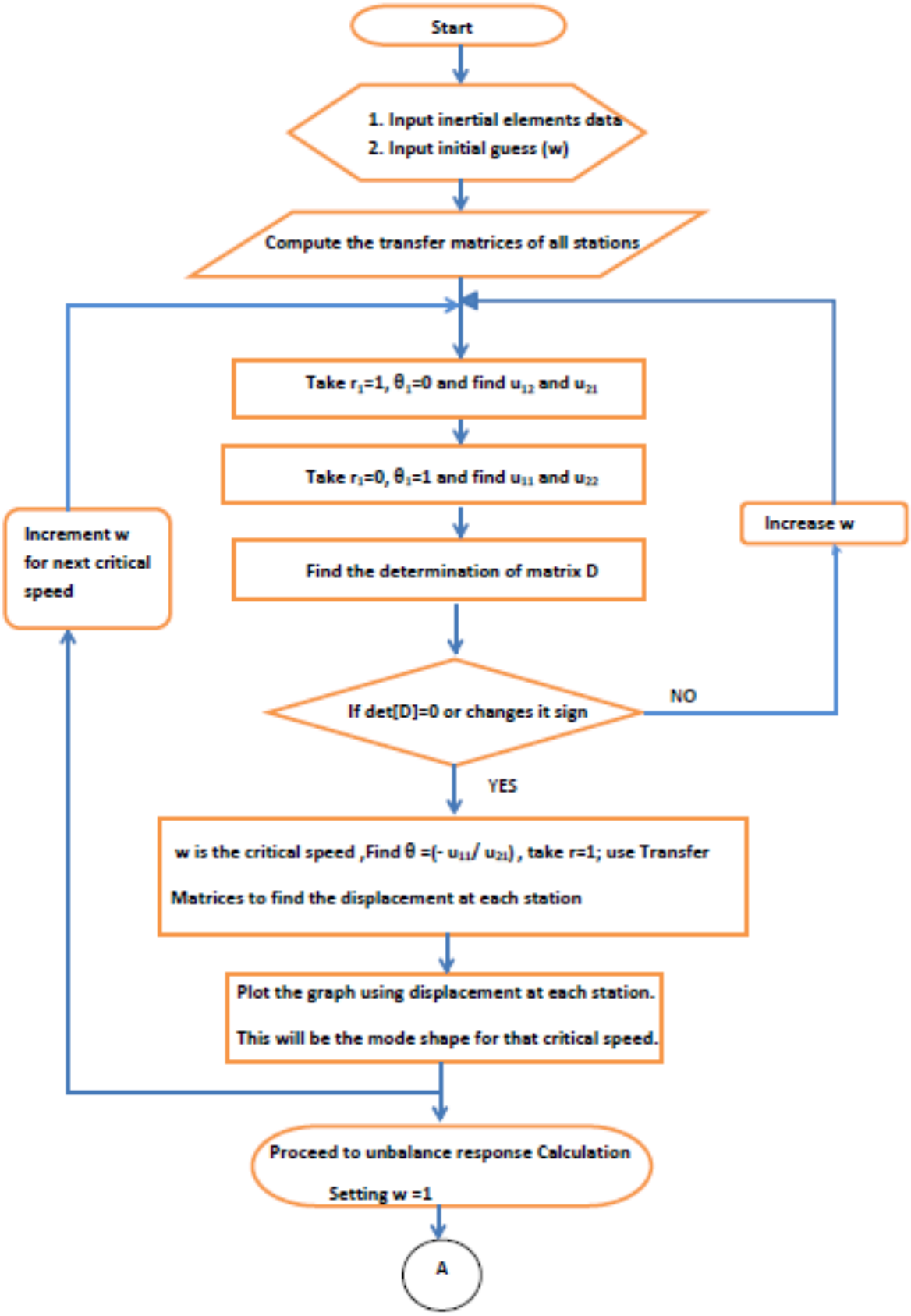
Algorithm:

1. Input the numerical values of the inertial parameters at all stations.
2. Input the value of rotor speed, ω at which the unbalance response has to be calculated.
3. Calculate the elements of the transfer matrices at all the stations.
4. Start from the left end matrix and go on multiplying the matrices to get the overall transfer matrix $[T_o]$ (17x17 matrix).
5. As the shear force and bending moments are zero at the left end boundary, these columns are deleted to get a reduced overall transfer matrix $[T_o']$ of size 9x9 from the original transfer matrix $[T_o]$.
6. Derive the overall transfer matrix in the final form, $[T_o'']$. An 8x8 matrix, from its parent matrix $[T_o']$. The unbalance column vector is obtained by extracting the last column vector of $[T_o']$. The last row of $[T_o']$ is deleted as it's a unity row.
7. State vector at station 1 is obtained by $[S_1] = U^{-1} [T_o'']$. The inverse can be found out by using any suitable method.
8. Use successive multiplication of the point and field matrices from the left to obtain the state vectors at all the rotor stations.
9. Compute the response at each rotor station by vector summation of the displacements.
10. Repeat the process 2-9 with different input data for rotational speed ω to obtain unbalance response over a range of rotational speeds. The vibration peaks corresponding to rotor critical speeds are determined from the plot of vibration amplitude versus frequency.

4.4.3 Code:

The code was written in the MATLAB environment. The algorithm was used and the variables scope and life were according to the flow chart (Fig 4.5). After writing the code it was tested using various rotor data and results were accurate with reasonable error. Hence the code was later adopted as back-end function in the Graphical User Interface to calculate any rotor's Unbalance responses.

4.3.6 Flow Chart for critical speed, mode shapes and unbalanced response



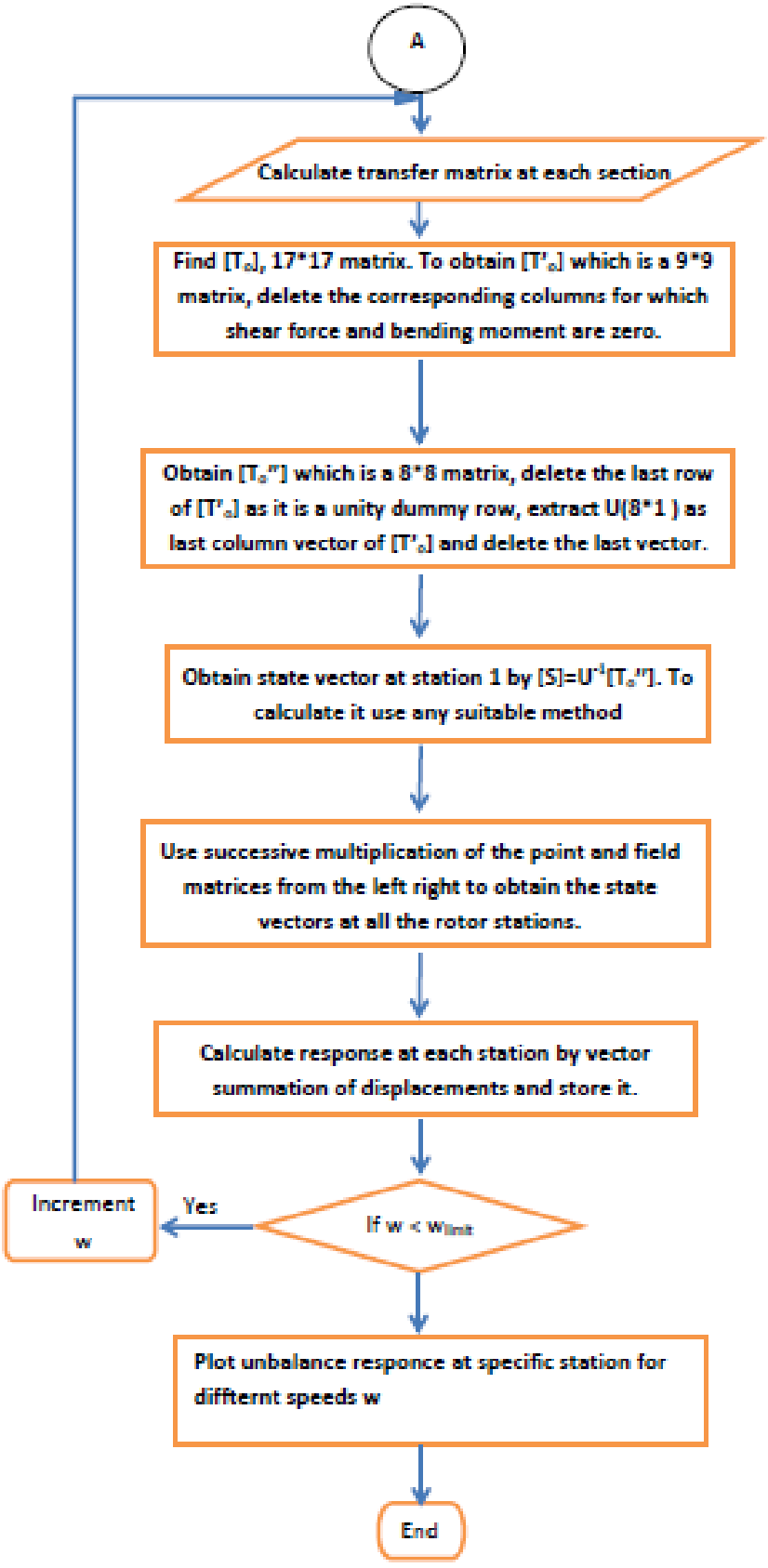


Fig 4.5: Flow Chart for critical speed, mode shapes and unbalanced response

CHAPTER –5

UI design and Integration

5.1 GUIDE – GUI Development Environment

GUIDE, MATLAB's Graphical User Interface development environment, provides a set of tools for laying out your GUI. The Layout Editor is the controlpanel for GUIDE. To start the Layout Editor, use the guide command.

5.1.1 GUIDE Toolset

The following links provide more information on the full set of GUIDEdevelopment tools.

•Layout Editor – adds and arranges objects in the figure window.

•Alignment Tool – aligns objects with respect to each other.

•Property Inspector – inspects and sets property values.

•Object Browser – observes a hierarchical list of the Handle Graphics objects in the current MATLAB session.

•Menu Editor – creates menus for the window menu bar and context menus for any component in the layout.

5.1.2 Steps followed to create a GUI:

•Design the GUI – often it is better to design the GUI on paper before beginning the implementation process.

•Laying out the GUI figure – the GUI figure is the window that contains the user interface controls, such as push buttons and menus and can also contain axes for displaying graphs and images. To create a component, from the palette the component was selected and draged them into the layout area. While it is selected the component can be resized from any corner handle. To align the components, those were selected (ctrl + click for multiple selection) and then clicked on 'Align' button. To modify property of each component double clicked on it to

open property inspector. Of all properties String, Tag, Value and enable are notable ones. String property controls what would be written on the component. The component's Tag property is used by GUIDE to name its 'callback' function. Value is important for popup-menu, radio-button and slider.

•Program the GUI – the M-file generated by GUIDE displays and controls the GUI figure you created with GUIDE. It is in this M-file that you program the callback functions for each user interface control.GUIDE generates this M-file with empty subfunctions for each component that has a callback associated with it.

5.1.3 Handles Structure

•Stores the handles of all controls, menus, and axes used in the GUI.

To access them use handles.edit1 or handles.pushbutton1, where edit1 or pushbutton1 are the Tag property of some component.

•Stores global data used in the program.

As the scope of a variable in a subfunction is limited to the scope of the subfunction and many a times it becomes necessary to pass data across function there the handles structure becomes handy. To create a new field in handles structure, do the following.

```
Handles.newValue=1:10;
guidata(h,handles);
```

The 1st line creates the variable and stores the assigned value, the 2nd line guidata then saves the new version of handles to the figure's application data. To retrieve the data

```
X=handles.newValue;
```

5.1.4 Push button Callbacks

When a push button is clicked in run time, it executes the callback with the name <tag of push button>_callback(). The callback takes 3 arguments, hObject state of currently selected object, eventdata which is to be defined by the future versions of matlab and handles which stores all the properties of all components.

```
functionpushbuttonOk_Callback(hObject, eventdata, handles)
```

To access a property of a component from within another components callbackget() and set()functions are used.

```
Name=get(handles.editname,'String');
```

It will get the string property of a component whose tag is 'editname' and assign it to Name variable.

```
set(handles.editname,'String','Hello World!');
```

It will assign 'Hello World!' to edit property of the component whose tag is editname.

5.2 UI design and Integration

As the Iterative waterfall model was followed, the software was developed gradually from very basic one to a sophisticated one. Here the user interfaces made in the progress were shown.

5.2.1 Type 1(Iteration 1)

5.2.1 User Interface version 1

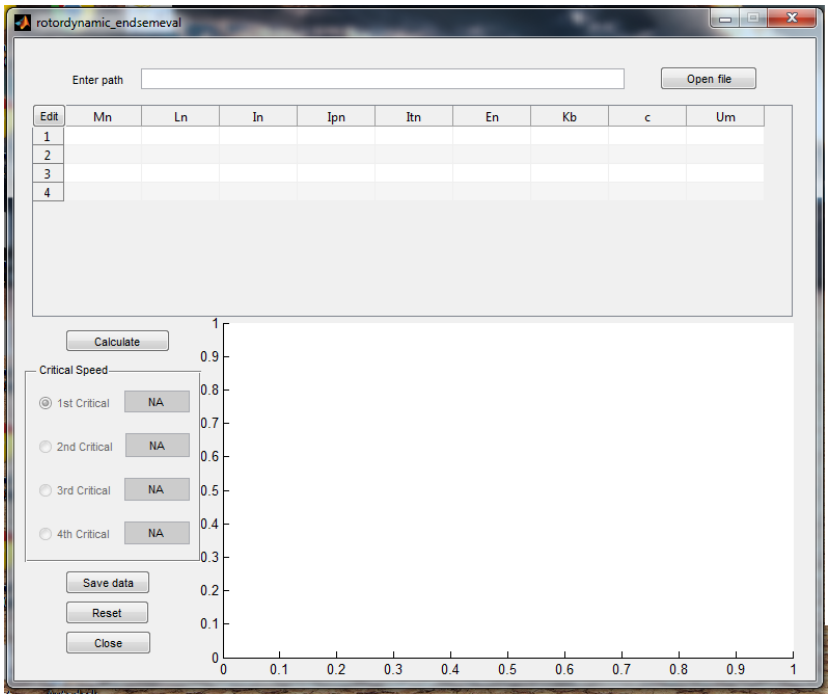


Fig5.1: User interface (type 1)

5.2.2Requirements fulfilled:

1. A button to locate an '.xls' file having the rotor data.
2. A table which would read data from the above data-file and display it.
3. Another button to calculate the critical speed and associated mode-shapes.
4. 4 radio buttons to represent 4 critical speed and associated mode-shapes, on selecting their critical speed would be shown and the mode-shape will be plotted.
5. A plot window that would show all the mode-shapes hose are to be shown.

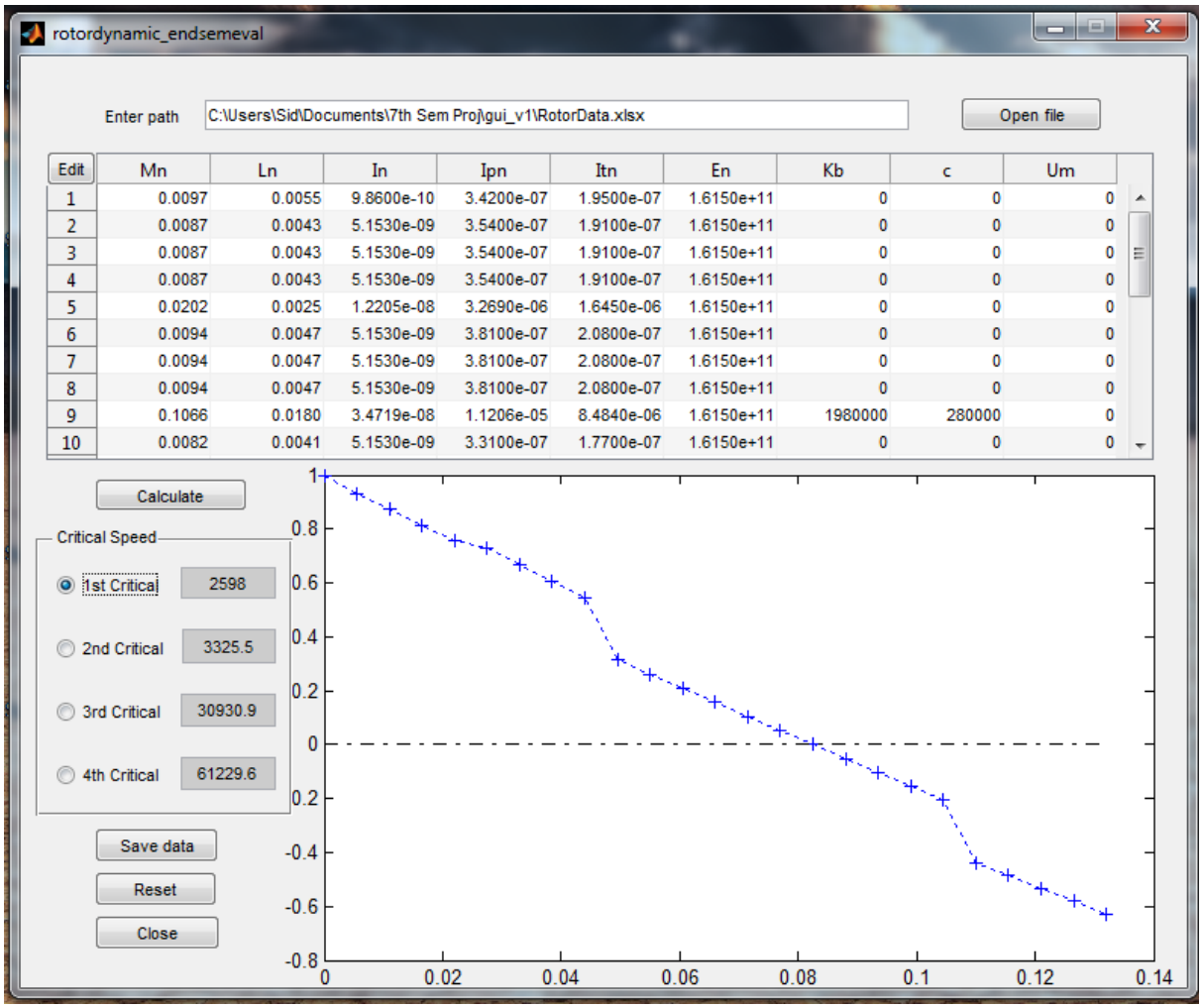


Fig5.2: User interface (type 1) while running

5.2.3 Functionalities

A file would be selected by an open file dialog window on clicking upon open file. Then the data presented in that file would be displayed on the table. Calculate button would calculate the critical speed by calling a function and passing the data present in the table. The function after calculation returns the 4 critical speed values and also the mode-shapes of them. Detail

of this function is discussed in the chapter 4 with algorithms and flowcharts. On selection any radio button the corresponding critical speed is shown and mode-shape is displayed. The save data button saves the .xls file. Reset button resets all fields to null. Close button closes the window.

5.2.4 Problems and new requirements

1st problem is that the user cannot change any data within the window. User has to do it in MS excel and reload the file again.

2nd problem is everything must be defined already outside, so a new user would be confused about how to make it. So an interactive table was proposed to overcome the problem.

New requirements were that their must also be the room for unbalance response. So a new section was to be made.

5.3Type 2(Iteration 2)

5.3.1 User Interface version 2

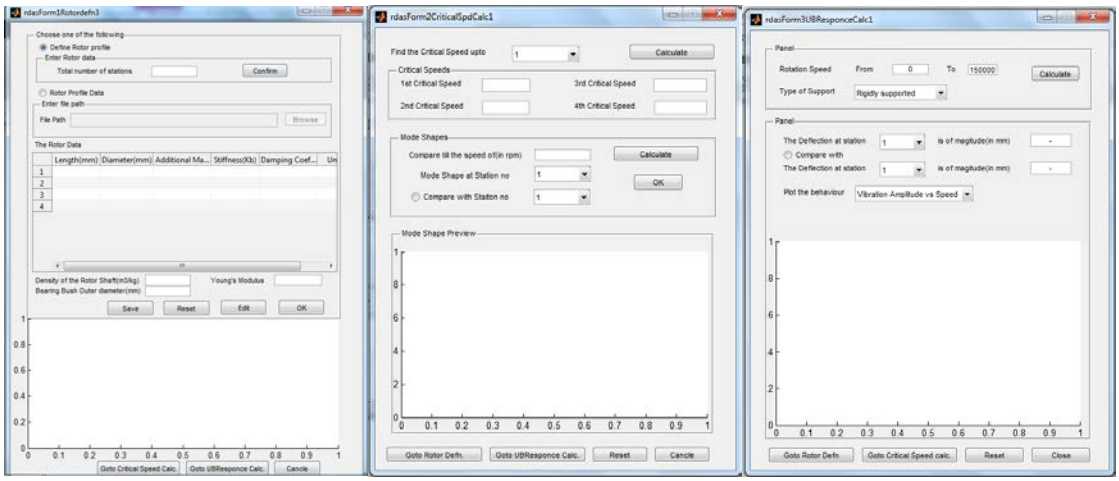


Fig5.3: User interface (type 2)

5.3.2Requirements fulfilled:

1. Now the user can change any data within the window. He need not to do it in the excel and reload the file again.
2. A new user would not be confused about how to make the excel file now. So an interactive table was made.
3. A new section was made for the Unbalance response.

5.3.3Functionalities

In the first form user can define the rotor profile by either importing an excel file or defining on the table given. Once done user can go for critical speed calculation or unbalance response calculation by clicking on suitable buttons. In the critical speed calculation window user can calculate the critical speeds or he can switch back to rotor definition window for any modification in the profile or skip it and go ahead and do the unbalance response calculation.

5.3.4 Problems and new requirements

1. All the information was not at a glance and switching in between was not user friendly.
2. There was no way to control the plot settings.
3. It was not possible to save any plot.

So the requirements were all the information and sections had to be in one window, there must be a plot controller and an option to save the plots.

5.4Type 3(Iteration 3)

5.4.1 User Interface version 3

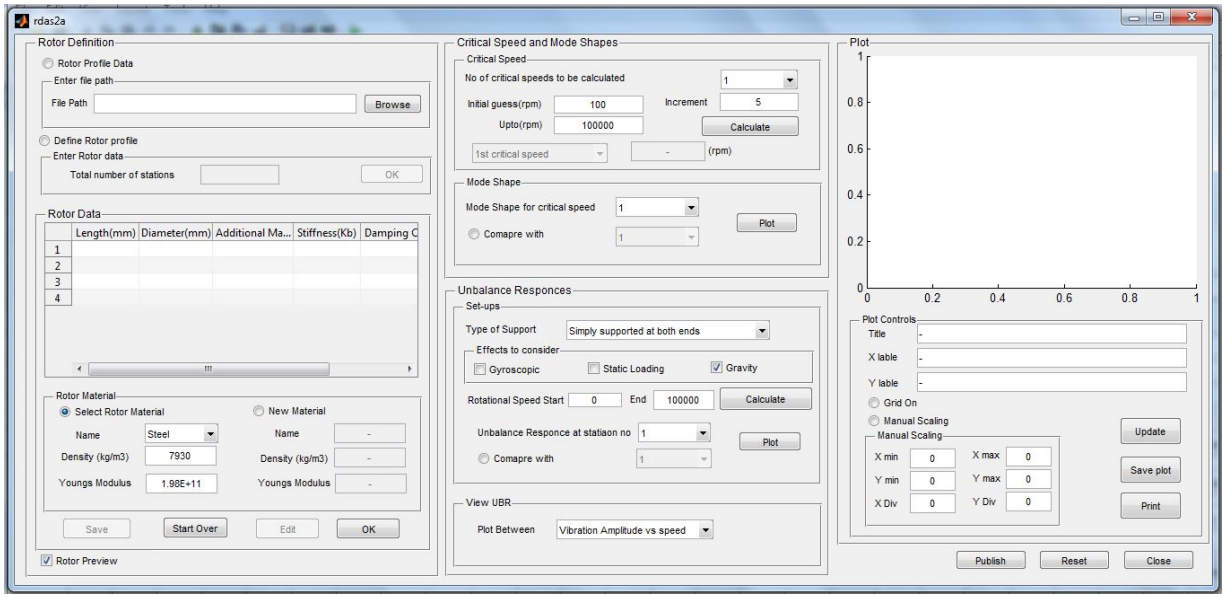


Fig5.4: User interface (type 3)

5.4.2 Requirements fulfilled:

1. All the information was at a glance so no need to switch between any window.
2. A new section was created to control the plots.
3. It is now possible to save any plot.

5.4.3 Functionalities

The user can either Import a .xls file that contain rotor data or choose to define one in the window itself. The user can define material properties also like density, Young's modulus etc. A new option was introduced to show a 2D sketch/Front view of the rotor. Save option was also implemented. Various plot related controls were made available.

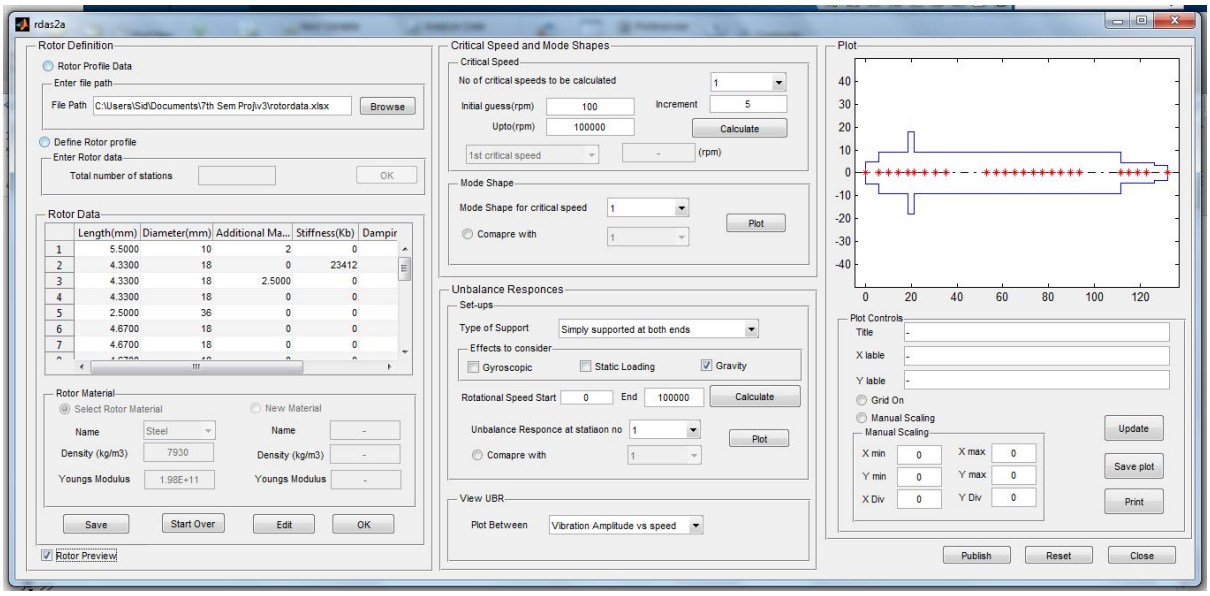


Fig5.5: User interface (type 3) while running

5.4.4Problems and new requirements

1. As the user interface grew bigger and lots of options were present, it became difficult for user to how to proceed. So a hint section was necessary.
2. As the plot section became larger, due to unavailability of space, the plot controller had to be shifted to a new window.
3. A print button had also become a necessity.
4. Some control over plot was also desirable.

5.5 Type 3(Iteration 4)

Improved User Interface of type 3

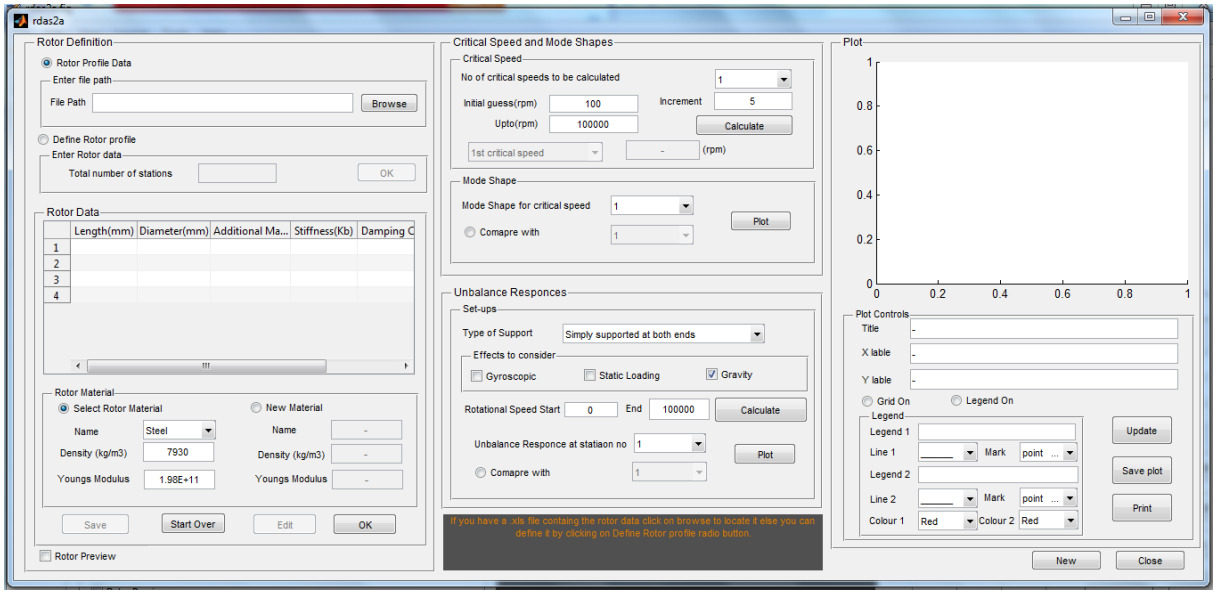


Fig5.6: User interface (type 4)

Some new features are introduced give more flexibility to the user and guide him/her to work around the software. Like a dedicated hint section was introduces.

A detailed description of the implementation procedure of this user interface is given below. It is divided into 4 parts.

1. Rotor Definition
2. Critical speed and mode shapes
3. Unbalance Responses
4. Plot

5.5.1 Rotor Definition

In this section user is asked to define the rotor profile in terms of rotors geometry. This could be done in two ways.

i) Rotor profile Data ii) Define Rotor Profile

Either of these two can be selected by clicking on corresponding radio button. If the former is selected the user is supposed to browse the necessary file. On clicking on the browse button an ‘open file’ window comes up. There the user can navigate to the file like any other programs ‘open’ option. MATLAB stores the selected file’s filename and file path, then

imports the data contained in the '.xls' file. It then stores it in its handles structure to use it further. For the user's convenience it is displayed on a table.

```
[filename, path]=uigetfile({'*.xlsx','*xls'},'File Selector');
filepath=strcat(path,filename);
set(handles.editFilePath,'String',filepath);
data = importfile(filepath,'sheet1');
set(handles.uitableRotorData,'Data',data);
```

In Define Rotor Profile section, the 'Number of Station' data would create a table or Number of data*6. Material would store the material properties to the table. User can now enter parameter values in corresponding field of the table.

In rotor data, set the Length and Diameter column in the corresponding fields. Assign zero to 'Stiffness' and 'Damingcoef.'for stations other than 'Bearing' type stations. The 'Hint' will also change to say what to do next.

Save button would open up a save window and prompt the user to save the rotor file in .xls file format. This is done similarly to that of Browse button only in place of 'uigetfile' function 'uiputfile' is used, instead of 'importfile', 'xlswrite' is used. 'Start Over' sets everything to the very initial condition of the Rotor Definition section.

'Okey' button does the following.

1. It gives hint about what to do next.
2. It makes a preview of the rotor based on rotor data.
3. It calculates the necessary data needed for the future calculation and tabulates them.

5.5.2 Critical speed and mode shapes

This section is responsible for the critical speed and mode shapes calculation. The way it does it is described below.

After Clicking on 'Calculate' button

It gets the values given in 'No of crtitical speeds to be calculated', 'initial guess', 'Increment', 'upto'.

The function passes them as argument along with rotor table which was made after clicking on ‘Okey’ button, into a function whose job is to calculate critical speed and mode shapes.

[cspeed, mode shape] = criticalspeed(noCriticalspeed, guess, Inc, Upto, rotorData)

The function returns the critical speed and mode shape data. The calculation is based on the method discussed in the Numerical Analysis chapter.

From the drop down box to see the critical speed user can select the appropriate option. The criticalspeed() function returns the cspeed as row vector. So by suitable speed can be shown easily.

After clicking on Plot

The criticalspeed() function returns mode shape data in a tabular/matrix form. Rows correspond to the critical speed and columns to stations. So in the Mode Shape section, in the Mode shape for critical speed dropdown box the selected option will access the columns of critical speed row of the matrix and plot it. If ‘Compare with’ is enabled it would also access the matrix data and add to the plot, thus giving the user a sense of comparison. It also gives hint about what to do next.

5.5.3 Unbalance Responses

There are many options shown in the ‘Type of Support’ dropdown list but only bearing supported is available as it was not possible to do in the given time frame. Similarly Only Gravity effect is considered.

After clicking on Calculate

It calls the function unbalanceresponce(). The arguments passed to it are rotordata, startSpeed, endSpeed, effects, support type. It returns a matrix whose rows correspond to stations and columns to rotational speed and the value to unbalance response. Just like the Plot button of Critical section, the plot button of this section does similar job.

unbalance = unbalanceResponce(rotordata, startSpeed, endSpeed, effects, supportType)

5.5.4Miscellaneous

5.5.4.1 Error Dialogs

For every input field an appropriate error message is raised for any input error like invalid input and not providing any input at all. The following syntax shows how to do it.

Syntex:

errordlg(errorstring,dlgname);

example:

errordlg('Select a station type!','Selection Error');

errordlg ,helpdlg, msgbox, warndlg , inputdlg are also some of the in-built dialog box readily available with similar syntax.

5.5.4.2 Plot controls

The user can modify the graphical properties of the plot like enabling grid, entering title, labelling the axes, controlling the legends etc.The following way it was done.

1. Store the user input value in matlab variable.
2. For line style like colour, line format, data point format, each of the chosen style are accessed and stored in an array.
3. Access the handles to the axes and assign the necessary change.

5.5.4.3 Saving and Printing Plots

Matlab provides inbuilt functions to save figure and plots. To save a plot:

Saveas(axes_handles,filename,'fileformat');

Here axes_handles is the handle to the axes. Supported file format are jpg, tff, bmp, pdf etc.

CHAPTER - 6

Software Testing and deployment

6.1 Software Testing

Testing is the process of evaluating a system or its component(s) with the intent to find that whether it satisfies the specified requirements or not. It also helps in finding errors or missing requirements.

6.2 Software testing methods

6.2.1 Black Box Testing

If the person who is testing does not have any knowledge of internal framework or underlying procedures then that type of testing is called Black Box testing. The tester is unaware to the system architecture and the source code is not available to him/her. Usually, when performing a black box test, the tester interacts with the system's user interface (GUI) by providing inputs and then scrutinizing all the outputs without the understanding of the software's internal functioning.

6.2.1.1 Advantages

1. It is very efficient for large code sections.
2. Access to the code is not required.
3. clearly differentiates client's viewpoint from the developer's viewpoint

6.2.1.2 Disadvantages

1. Constrained Coverage since just a chose number of test situations are really performed.
2. Inefficient testing, because of the way that the tester just has restricted learning about the application.
3. Blind Coverage, since the tester can't target particular code segments or error inclined regions.

6.2.1.3 Bugs found and fixed

1. The input for total number of section in the Rotor Definition section was not showing

error if any alphabet or negative number was entered. So an appropriate error message was thrown when this happened.

2. In ‘Rotor definition’ section if clicked on ‘Start Over’ everything goes to the initial state but if OK is clicked immediately afterwards error occurs so a warning is now issued if the user does so.
3. In the ‘Unbalance Response’ section whether ‘Static loading’ or ‘Gyroscopic’ check box is checked does not affect the unbalance response. So to inform the user about the issue an appropriate message is shown.
4. Similarly for the Type of Support section, only ‘Bearing support at both ends’ works, so for other cases appropriate message is shown.

6.2.2 White Box Testing

If the tester has detailed knowledge of internal framework and code structure and he/she carries out a detailed analysis of code then this test technique is called white box testing. It is also known as glass testing or open box testing. The tester needs to have a look inside the source code and find out which unit/chunk of the code is behaving inappropriately.

6.2.2.1 Advantages

As the tester has knowledge of the source code, it becomes very easy to find out which type of data can help in testing the application effectively.

1. It helps in optimizing the code.
2. Extra lines of code can be removed which can bring in hidden defects.
3. Due to the tester's knowledge about the code, maximum coverage is attained during test scenario writing.

6.2.2.2 Disadvantages

1. Due to the fact that a skilled tester is needed to perform white box testing, the costs are increased.
2. Sometimes it is impossible to look into every nook and corner to find out hidden errors that may create problems as many paths will go untested.
3. It is difficult to maintain white box testing as the use of specialized tools like code analyzers and debugging tools are required.

6.2.2.3 Bugs found and fixed

In the function makedatatable() when the Rotor data table is passed to make the datatable which will be used later for the calculation of the critical speed and unbalance response had severe flaws in regard to identify what type of station is the current station. The code has been debugged to get correct result.

For the Rotor definition section if a file was browsed and rotor material is saved within it then for some rotor compiler was showing error. If in the 'Rotor Material' section the density, young's modulus and material name are saved along with the rotor data in a excel file format, it was not possible. So the approach was scratched.

6.3 Deployment

It is a process where the program code is converted into an installation package which is ready to be distributed. The end user can install it on his local machine like any other windows/ linux/ mac application.

In some scenarios compiling the codecan be beneficial:

- If one wants to distribute your code to people who do not have Matlab installed. So one need not to worry about version capability.
- If one wants to run a large number of Matlab program concurrently on a computing network, as limited number of licenses are generally available. Working with a compiled standalone version does not requiring a Matlab license.

After creation of the component with the MATLAB Compiler, one can distribute or deploy it to the world so that they can install it on their computers and they need not have to have MATLAB.

To deploy a program, following steps are carried out:

- As per the type of generated application, the necessary components are packaged.
- It is distributed to the end user/client.

- The end user is to install it on his/her computer. While installing the use has to run the MCRInstaller only once on their system which would in turn enable those to run any program compiled by matlab on their system.

6.3.1 Deploy process

1. To bring up the deployment tool, it can be done in either of the following ways. Type 'deploytool' in matlab window or click on MATLAB compiler from the APP tabs as shown in the figure.

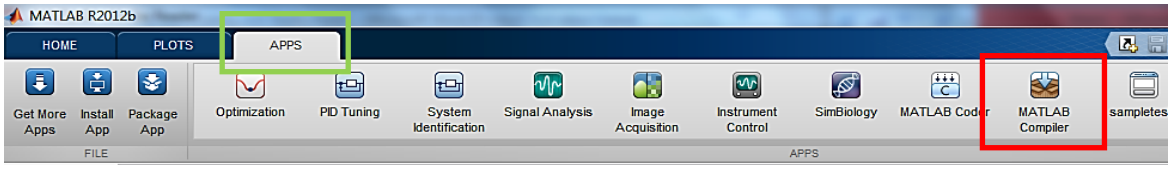


Fig 6.1: Deploy tool location

2. Deployment project window will come up. A name was given to the project, and a location to specify the output directory and it the 'type' field from the drop-down menu 'Windows Standalone Application' was chosen. Then clicked on 'OK' to proceed ahead.

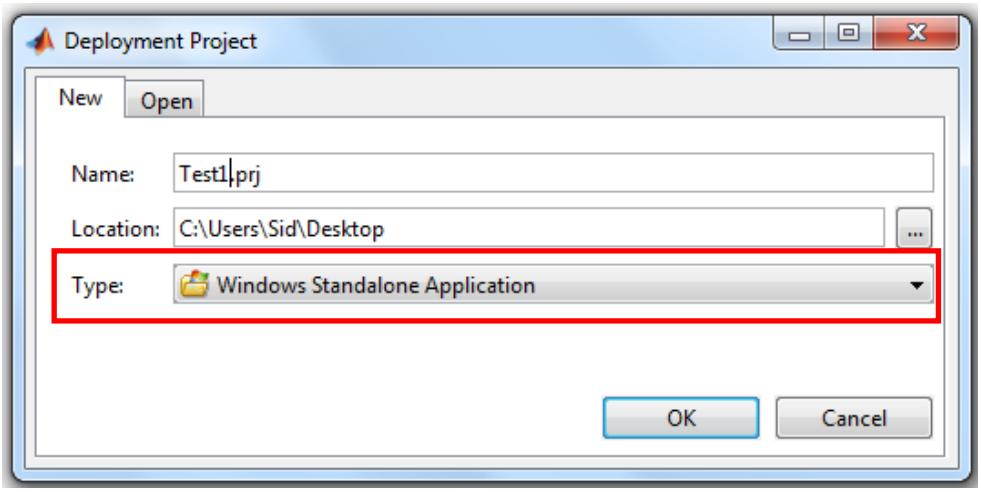


Fig 6.2: Deployment project window

3. In the Windows standalone Application window, clicked on 'Add main file' to add the main function file which in this case is the figure's '*.m' file. Then clicked on 'Add files/folder'to add functions which are referenced by any of the function. For more options, go to 'settings' from Project menu.

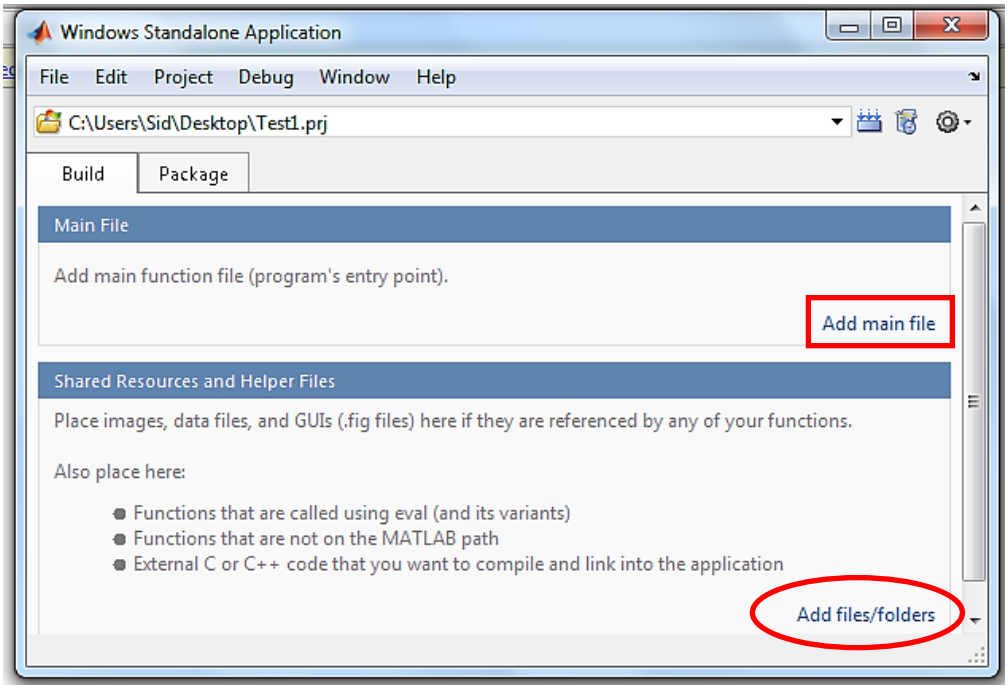


Fig 6.3: Windows standalone Application window

4. After the files are added, a decision has to be made, whether to include MCR (needed for the systems which do not have Matlab installed) or not. Here it was decided to add the file. File was included by going to 'Package' tab and selecting Add MCR or option or from Settings -> Add MCR.

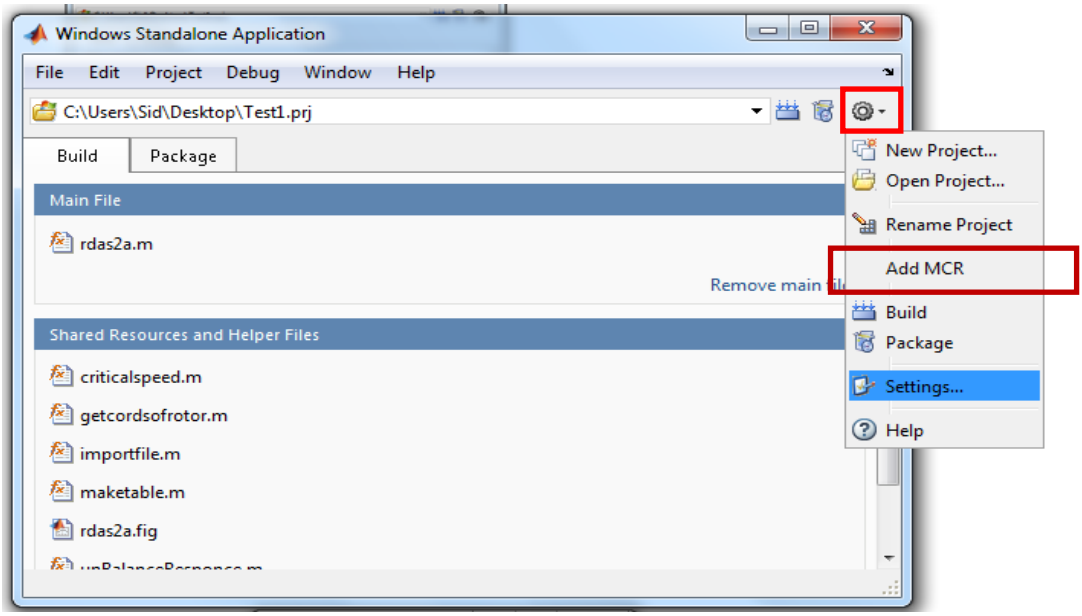


Fig 6.4: How to add MCR

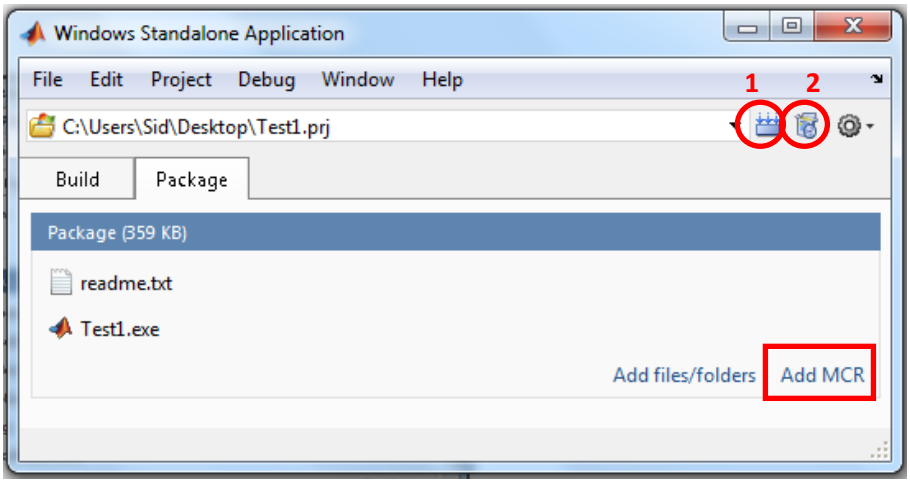


Fig 6.5: How to add MCR and build project

5. Now it's ready to be built. This can be done in 2 ways. Select the 'Build' option (labelled 1 in above fig.) to start process or click on package(labelled 2). The later gives 2 options, whether to make 'Self-extracting exe (*.exe)' or 'ZIP file (*.zip)' file. Here the former was chosen.

6. Matlab then made the .exe file and put it in project directory with two subfolders, namely 'src' and 'distrib', then it was ready to be deployed.

7. In this step, the end user or client can run the .exe file on their machine. If the executable file is opened then it would first install the MCR (Matlab Compiler Runtime) if not already installed on the targeted system and then the run the program's exe. The installation process is like any other normal software installation process with intuitive user friendly GUIs.

8. If the user does not have MCR and no MCR is provided by the Matlab executable file then he/she can download it from the link below.

http://www.mathworks.in/products/compiler/mcr/

CHAPTER - 7

Results and Discussion

1. The MATLAB coding for finding critical speed, mode shapes and unbalances response was done and tested for different rotor and operating conditions. On success of the code a Graphical User Interface was developed using MATLAB GUIDE, which can incorporated different rotor models and boundary conditions. The simulation/analysis obtained from the software was verified with standard problems and those are coming with reasonable accuracy.

2. The back-end programs were written with great care and attention to include all kind of possible cases.

3. The software was deployed as executable file, so that any user can install it in his/her local machine.

5. In the following figures the results obtained from the software are shown.

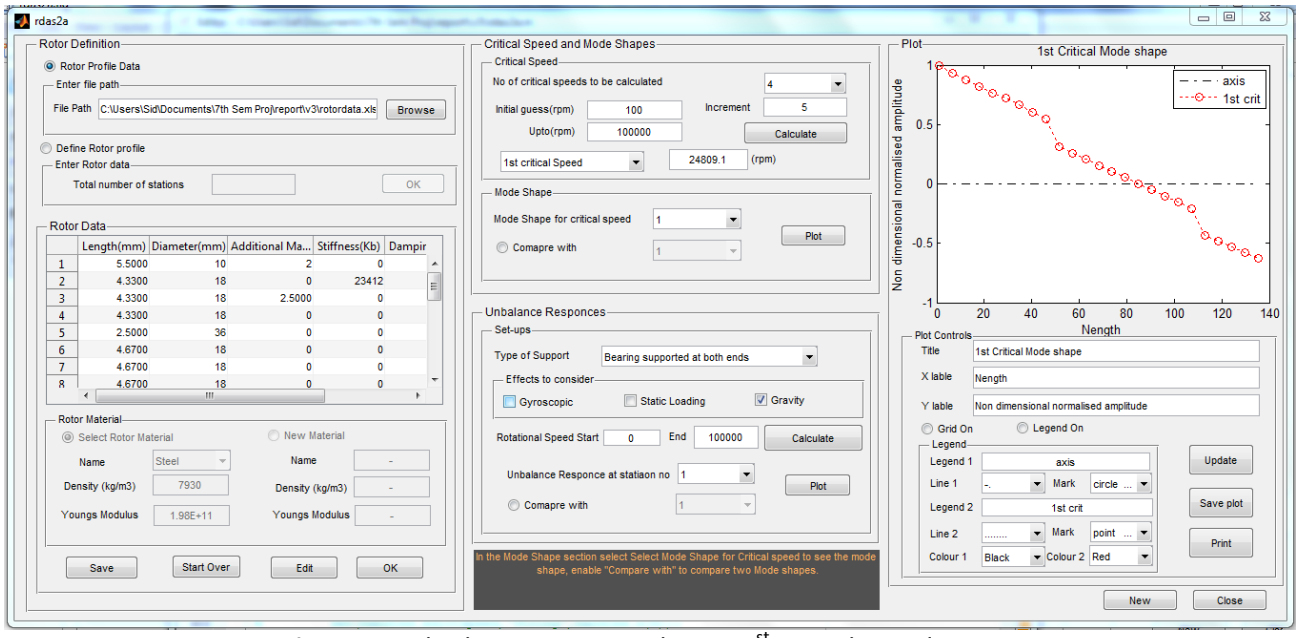


Fig 7.1: Mode shape corresponding to 1st critical speed

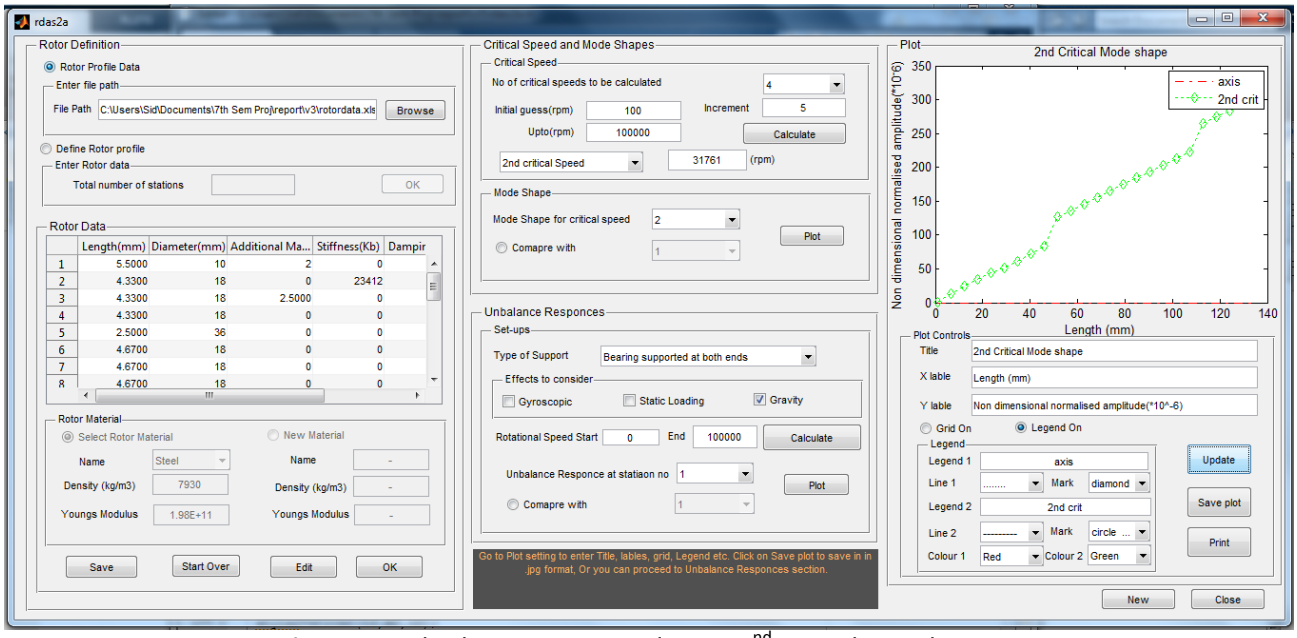


Fig 7.2: Mode shape corresponding to 2nd critical speed

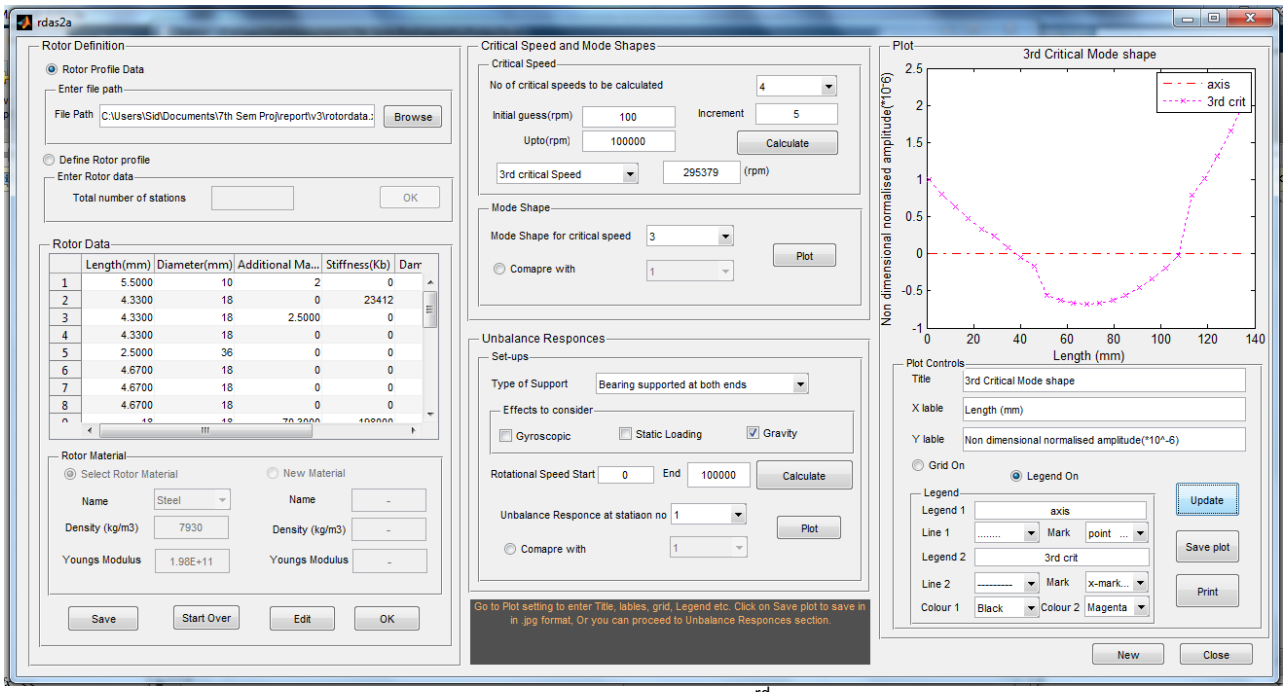


Fig 7.3: Mode shape corresponding to 3rd critical speed

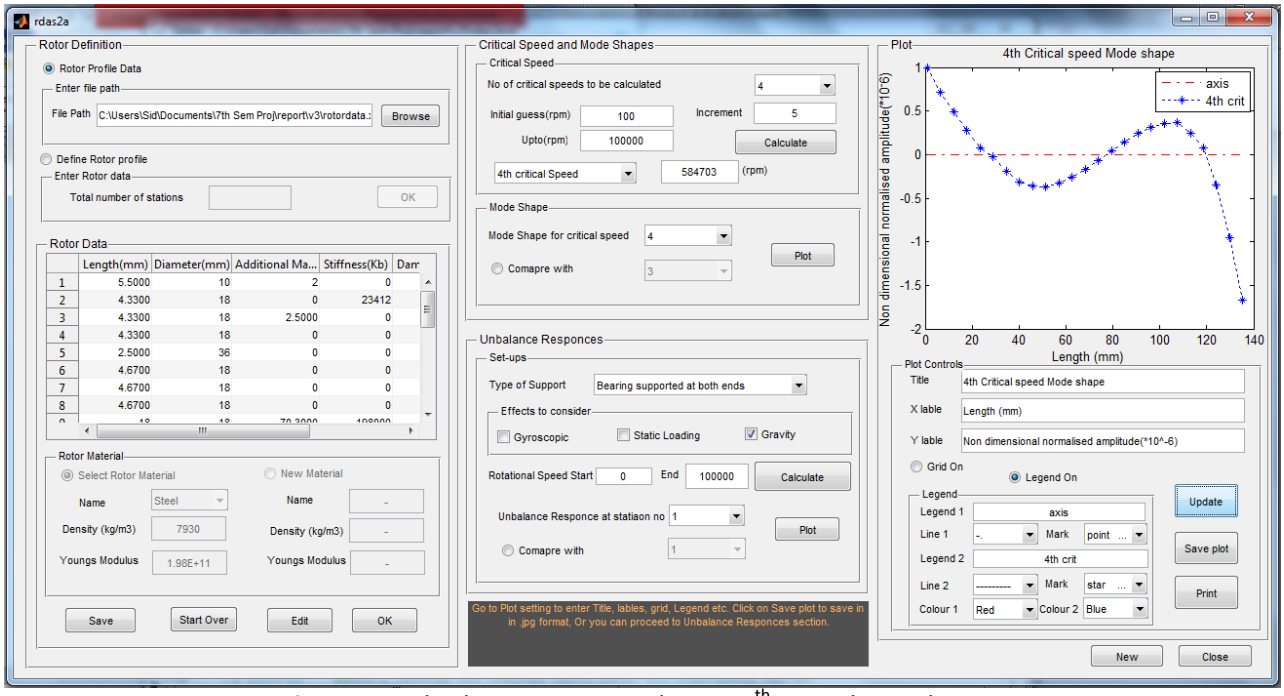


Fig 7.4: Mode shape corresponding to 4th critical speed

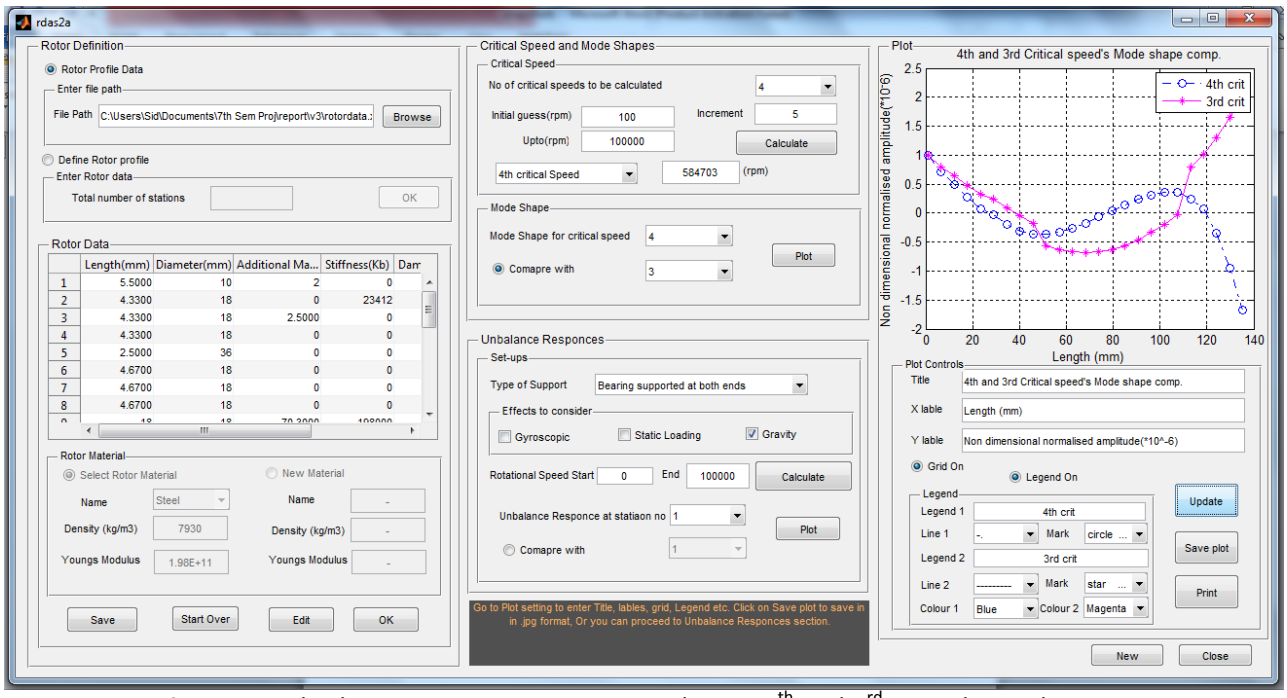


Fig 7.5: Mode shape comparison corresponding to 4th and 3rd critical speeds

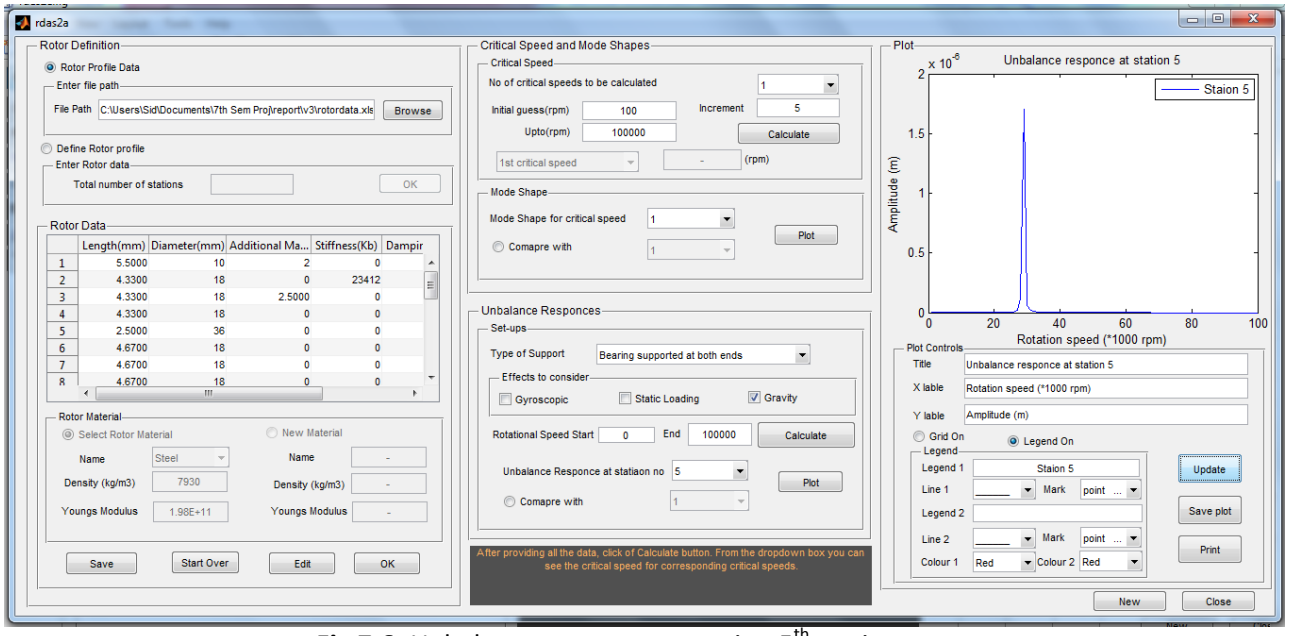


Fig 7.6: Unbalance response at station 5th station

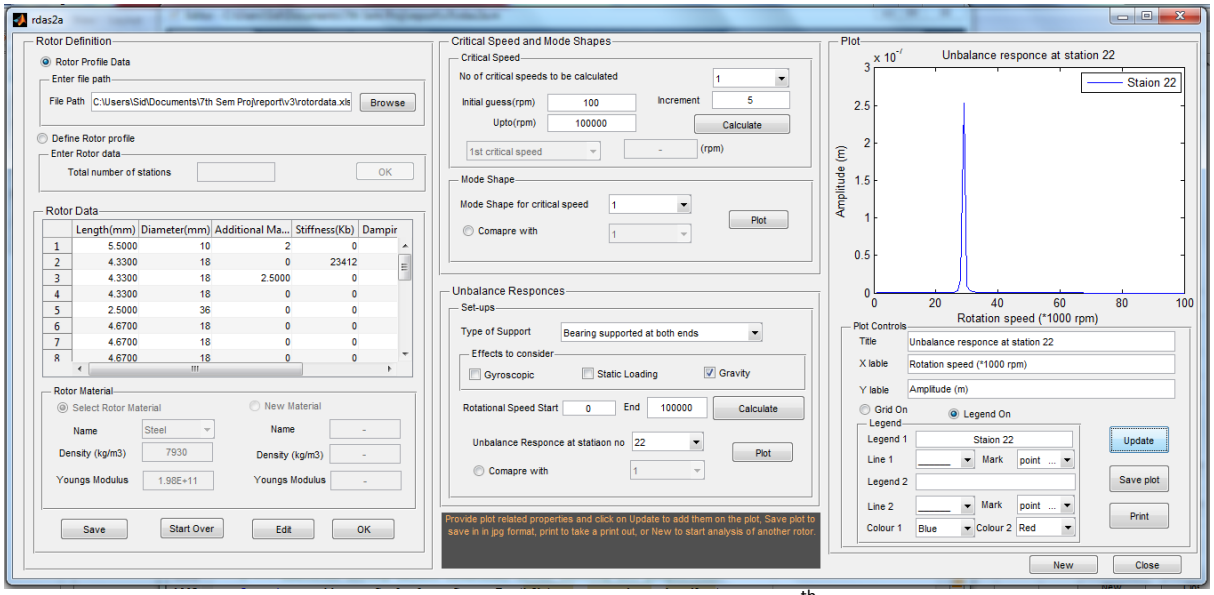


Fig 7.7: Unbalance response at station 22th station

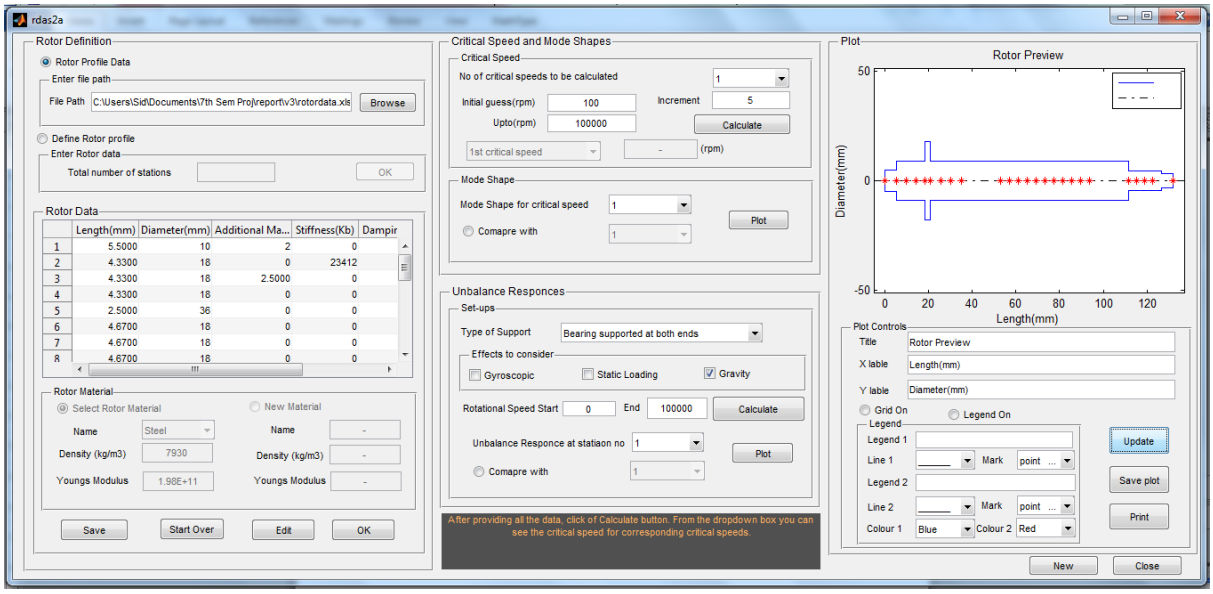


Fig 7.8: The rotor's profile (Front View)

CHAPTER – 8

Conclusion and Scopes ahead

Current needs to solve any engineering issue or new design need knowledge on multi-disciplinary subjects. Knowledge on all subjects is difficult to harness, so an attempt was done to reduce time for design engineers to do rotodynamic analysis. The objective of this thesis was to simplify the rotodynamic analysis of rotor to help the engineering society for faster decision making and design of modern machinery.

The developed software will also help the researchers to compare the solutions from TMM with different other methods to find critical speed, mode shapes and unbalance response.

There are many areas where improvements are possible, both in the front-end and back-end.

In the Front-end(GUI), the rotor definition section can be improved to add new features, like asking the user to choose the type of stations, then accordingly a new window will pop-up asking the relevant data. Options should also be there to copy a station data to decrease repetitiveness, Up/Down button to move the station and delete to remove one.

As for the back-end is concerned, program can be written to take care of gyroscopic effect, static loading condition to allow more realistic simulation of rotor. The support conditions can also be improved similarly.

Reference

(1)**Chakravarty A**., Analytical and Experimental Studies on Gas Bearings for Cryogenic turbo-expanders, B.Tech dissertation, IIT Kharagpur (2000)

(2) **Rao, J.S.** Rotor Dynamics Wiley Eastern Limited, New Age International Limited (1991)

(3) **Yuan-pin Shih and An-Chen Lee**, Identification of the unbalance distribution in flexible rotors (1996)

(4) **Francisco Beltran-Carbajal, Gerardo Silva-Navarro and Manuel Arias-Montiel**, Estimation and Active Damping of Unbalance Forces in Jeffcott-Like Rotor-Bearing Systems (2012)

(5)**Creating Graphical User Interfaces**, The MathWorks, Inc (2000)

(6) **www.mathworks.com**

(7)**https://admin.kuleuven.be**/icts/onderzoek/wetsoft/software/matlab/pdf/matlab-deploytool-standalone

(8)**www.stackoverflow.com**

(9) **Balje, O.E.** Turbomachines, John Wiley & Sons, New York USA (1981)