Dynamics of Rotating Machines

M.I. Friswell, J.E.T. Penny, S.D. Garvey and A.W. Lees
Cambridge University Press, 2010

Rotordynamics Software Manual

1. Introduction

This software is a set of scripts written in MATLAB to accompany the above book. The primary purpose of the software is to illustrate features of rotating machines described in the book, in particular the lateral motion based on shaft-line models. Of course the software can be used to analyze other machines and for research purposes, although there are no guarantees implied with the software and it is the user's responsibility to confirm the suitability and accuracy of the results.

The software is open source and access is available to all of the scripts, allowing the user to fully understand the methods used. The derivation of most of the elements and the descriptions of the analysis performed is given in the book. There are many comments throughout the scripts, and emphasis has been placed on clarity within the software rather than pure numerical efficiency. Even so, the software can be used to model machines with a relatively large number of degrees of freedom.

Installation of the software is very straightforward. Merely create a new directory, copy the scripts to this directory, and add the location of this directory to the MATLAB path (-> File -> Set Path in recent versions of MATLAB). The examples can then be placed in a separate directory that is set as the default MATLAB directory. We have tried to use standard MATLAB commands, and so even relatively old versions of MATLAB should run the software. No extra toolboxes are required.

The toolbox consists of a set of functions and scripts called from the command window. Although a GUI placed around this basic toolbox might be desirable, we consider simple access to the basic functionality is the priority. This also allows scripts for the examples in the book to be included easily. These examples are a good basis for generating new models.

2. Overview of Software

At a basic level the software consists of three aspects:

- Defining the model, forcing and operating conditions;
- Analyzing the system and generating the results; and
- Graphical means for interpreting the model and results.

These aspects will be dealt with in detail in the following sections. The definition of the system (model, forcing and operating conditions) is all incorporated into a single MATLAB structured array. Once defined, this allows the model to be passed easily to analysis or plotting functions.

The rotating systems considered can be modelled in the stationary or the rotating frame of reference. Most of the analysis in the book is concerned with axi-symmetric rotors and these are conveniently modelled in the stationary frame. These systems form the core of the software, and a variety of bearing options are available, including short fluid bearings. It should be emphasised that no stator dynamics are included (apart from when the combined bearing and stator at a single node is modelled as a constant stiffness matrix). The software could be readily extended to incorporate stator dynamics if required. Some analysis of machines with asymmetric rotors is included, where the analysis is performed in the rotating frame. Here the assumption is that the bearings and stator are isotropic and hence some bearing definitions (for example fluid bearings) are not consistent with this assumption. Other specific analysis functions are available, for example the time simulation of a machine with non-linear fluid bearings.

The scripts for plotting basically give some useful tools to check the model and interpret the results. MATLAB has superb plotting and formatting facilities and these may be used easily to generate any figures required. A key plot is to check the model, since any errors in the model definition are easily spotted. A number of functions are included for this.

The way nodes, elements and forcing are defined is consistent with the book, and hence this manual should be read in conjunction with the book. Furthermore the order of degrees of freedom at nodes, the definition of element matrices, and so on, is also consistent with the book.

3. System Definition

The model will be defined using definitions of the nodes, shafts, bearings and disks. The model is defined as a structured array in MATLAB; the name of this array, and therefore the model, is given as `Model` here, although this can be renamed to something more meaningful for particular examples. Each row of the defining matrices gives a distinct node, element or force. If different definitions are included in the same matrix, they may required different numbers of parameters; however the row for the definitions with the smaller number of parameters should be filled with zeros so that each row has equal length.

3.1. Defining Nodes

We have to define the position of the nodes. Since we are only interested in shaft line models the axial position completely specifies the node location. The node definition matrix is:

```
Model.node = [Node_1,  z_1 ; ...
              Node_2,  z_2 ; ...
```

If there is only one column then we assume these are the *z* co-ordinates and the nodes are labelled sequentially (starting at 1). Normally the nodes are numbered sequentially, although for co-axial rotors the nodes this may not be so and their positions may overlap.

3.2. Shaft Elements

The main shaft elements are Euler and Timoshenko circular (possibly hollow) shafts that may be used for analysis in the stationary or rotating frames. Asymmetric shafts may be included by specifying shaft properties in the two directions. Tapered shaft are also available in the stationary frame only. Both Euler and Timoshenko beam theories are modelled. The shaft types are:

1-8	circular section shafts
11-18	asymmetric shafts
21-28	circular section *tapered* shafts

The shaft definition matrix for beam types 1-8 is:

```
Model.shaft = [Shaft_Type  Start_Node  End_Node  outer_diameter
inner_diameter  rho  E  G  damping_factor  axial_force  torque; ...
```

`damping_factor` gives the factor for proportional damping of the stiffness matrix for the element.

The shaft type determines whether the shear, rotary inertia or gyroscopic effects are included in the analysis. Table 1 shows the effects included for each shaft type. Note that *G* must in input even if shear effects are neglected (note *G* is ignored and can be anything in this case). $G = 0$ also means that shear effects are neglected.

Note that a constant axial force (where a tensile force is positive) and/or torque may be applied to the shaft element (see Sections 5.4.3 and 5.4.6 in the book).

Symmetric		Effects Included		
Shaft Type		Shear	Rotary Inertia	Gyroscopic
1				X
2		X	X	X
3		X	X	
4		X		X
5			X	X
6		X		
7			X	
8				

Table 1. The different shaft types for a circular section shaft.

For asymmetric shaft elements, the Shaft Type is given by adding 10 to the element type numbers given in Table 1. The inertia properties of asymmetric shaft elements are assumed to be symmetric. The effect of shear in the mass matrix (via the shape functions) is neglected. The element definition is

```
Model.shaft = [Shaft_Type  Node_1  Node_2  EI_x  EI_y  Phi_x  Phi_y
rhoA  rhoI  damping_factor  axial_force; ...
```

where `EI_x` and `EI_y` are the flexural rigidities, and `Phi_x` and `Phi_y` are the shear constants, in the *x* and *y* directions in the rotating frame. `rhoA` and `rhoI` are the products of mass density and shaft cross-sectional area and mass density and second moment of area respectively.

For tapered shafts elements, the Shaft Type is given by adding 20 to the element type numbers given in Table 1. Damping is not included for tapered elements. The definition is

```
Model.shaft = [Shaft_Type  Node_1  Node_2  outer_dia_1  outer_dia_2
inner_dia_1  inner_dia_2  rho  E  G  axial_force; ...
```

3.3. Disk Elements

Disks are specified as

```
Model.disc = [Disk_Type  Node  ... properties ...; ...
```

Disk type 1 is a circular disk, given by dimensions and mass density. The disk definition matrix for disk type 1 is

```
Model.disc = [1  Node  rho  thickness  outer_diameter  inner_diameter;
...
```

Disk type 2 is specified by the mass, and the diametral and polar moments of inertia:

```
Model.disc = [2  Node  mass  diametral_inertia  polar_inertia; ...
```

Disk type 3 is the same as disk type 1 except that the gyroscopic effects of the disk are ignored in the analysis.

Disk type 4 is the same as disk type 2 except that the gyroscopic effects of the disk are ignored in the analysis.

3.4. Bearings, Seals and Other Rotor-Stator Interactions

The bearing types are defined as follows:

1 Rigid short bearing - pinned boundary conditions
2 Rigid long bearing - clamped boundary conditions
3 Constant stiffness and damping - diagonal, no rotational stiffness
4 Constant stiffness and damping - diagonal
5 Constant stiffness and damping - no rotational stiffness
- 2x2 matrices required
6 Constant stiffness and damping - full 4x4 matrices required
7 Hydrodynamic short width bearing theory
8 Seals

Bearing types 1 and 2 are simply specified as

```
Model.bearing = [Bearing_Type  Node ; ...
```

Bearing types 3 to 6 are defined by specifying the constant stiffness and damping properties, as

```
Model.bearing = [3  Node  kxx  kyy  cxx  cyy ; ...

Model.bearing = [4  Node  kxx  kyy  kθθ  kψψ  cxx  cyy  cθθ  cψψ ; ...

Model.bearing = [5  Node  kxx  kxy  kyx  kyy  cxx  cxy  cyx  cyy ; ...

Model.bearing = [6  Node  kxx  kxy  kxθ  kxψ ... etc; ...
```

The hydrodynamic short bearing option is specified as follows:

```
Model.bearing = [7  Node  F  D  L  c  eta ; ...
```

where

`F` is the static load on the bearing (N)
`D` is the bearing diameter (m)
`L` is the bearing length (m)
`c` is the bearing radial clearance (m)
`eta` is the oil viscosity (Ns/m^2)

The seal option is specified as:

```
Model.bearing = [8  Node  P  R  L  c  V  fric ; ...
```

where

`P`	is the pressure difference across the seal
`R`	is the seal radius
`L`	is the seal length
`c`	is the seal radial clearance
`V`	is the seal average axial stream velocity
`fric`	is the seal friction coefficient

3.5. Forcing Definition

We also need to specify the forcing. The options for `Force_Type` are

1	Mass Unbalance
2	Couple Unbalance
3	Bent Shaft
4	Foundation Excitation – frequency domain
5	Foundation Excitation by a pulse – time domain
6	Spinner
7	Excitation through an Auxiliary Bearing

For mass or couple unbalance (Force Types 1 and 2) we have

```
Model.force = [Force_Type  Node  unbalance_mag unbalance_phase; ...
```

where `unbalance_mag` and `unbalance_phase` specify the unbalance magnitude (for example, disk mass x offset, or unbalance mass x radius) and the phase of the unbalance in rad/s. The definition of phase is based on the definitions in Section 6.3.1 of the book, where an unbalance with zero phase would be in the *x*-direction at t=0.

A bent rotor (Force Type 3) is specified using the syntax

```
Model.force = [3; ...
```

It is clear from this description that the bend deformation must be specified elsewhere, and is contained in the parameter `Model.bend`. This parameter is a matrix with the same number of rows as the number of nodes in he model, and either 1, 2 or 3 columns. If `Model.bend` only has a single column, then the bend is in only in the *xz* plane and `Model.bend` gives the transverse displacement at each node. If `Model.bend` has two columns then the bend is in only in the *xz* plane and the first column is the node number and the second column is the transverse displacement at each node. If `Model.bend` has three columns then the first column is the node number, the second column is the transverse displacement in the *x* direction at each node and the third column is the transverse displacement in the *y* direction at each node. This displacement is expanded to all degrees of freedom (both translations and rotations) using the transformation from Guyan reduction.

The frequency response due to foundation excitation (Force Type 4), is specified by

```
Model.force = [4 amp_x_1 amp_y_1 amp_x_2 amp_y_2 ...]
```

where `amp_x_i` and `amp_y_i` are the amplitudes in the *x* and *y* directions at bearing number `i`. These amplitudes can be real and negative, or complex, to specify phase relationships.

The time response due to a half-sine pulse excitation through the foundation (Force Type 5), is specified by

```
Model.force = [5 amp_x_1 amp_y_1 ... pulse_duration]
```

where `amp_x_i` and `amp_y_i` are the amplitudes in the *x* and *y* directions at bearing number `i`. These amplitudes must be real; any imaginary terms are ignored. The `pulse_duration` gives the duration of the half-sine pulse.

For a spinner (Force Type 6), attached to the rotor through an auxiliary bearing, the forcing definition is

```
Model.force = [6  Node  unbalance_mag unbalance_phase; ...
```

Note this is very similar to the unbalance definition, and the difference is that the spinner can rotate at spin speeds other than the rotor spin speed, and these spinner frequencies are defined in the analysis function.

For an arbitrary force applied through an auxiliary bearing (Force Type 7), the forcing definition is

```
Model.force = [7  Node  f_x f_y; ...
```

where `f_x` and `f_y` are the coefficients of the forcing in the *x* and *y* directions. This definition is used to calculate the frequency response function, and the so `f_x` and `f_y` may be complex to allow for a phase difference in the force applied. This defines the (constant) direction of the force applied at all frequencies.

4. Analysis Functions - Stationary Frame

There are a range of analysis functions that require the input of the model structured array and also some other parameters such as the rotor spin speed (either a single spin speed or a vector of spin speeds). The output for the functions may be saved, plotted or printed.

Zeros are introduced for any DoF that are constrained out (for example, for short or long rigid bearings), so that the number of responses (either mode shapes or actual response) is always equal to four times the number of nodes. For couplings, or pinned / clamped connections for coaxial rotors, the response is also given at all of the constrained degrees of freedom.

4.1. Calculate Characteristic Roots and Modes (Eigenvalues and Eigenvectors)

To calculate eigenvalues and eigenvectors at a given speed we use the following function

```
[eigenvalues,eigenvectors,kappa,eccentricity]
                                              = chr_root(Model,Rotor_Spd)
```

`Rotor_Spd` defines the rotor speed at which to calculate the eigensystem. This can either be a single rotor spin speed, or a vector of spin speeds (in rad/s), which gives the data for a natural frequency map or a Campbell diagram. For a vector of speeds the eigenvectors are given as a three dimensional array (the order of the dimensions are: degrees of freedom; mode number; rotor speed). Note if there is only one output argument then the eigenvectors are not computed. If there is only two output arguments then `kappa` is not computed.

`kappa` is the direction and size of the whirl orbits, for both translations and rotations, at each node. Note that the length of each column (representing a mode) is the same as the mode shape; the value of kappa is identical in the two directions for a particular node.

`eccentricity` gives the eccentricity of the fluid bearings at all spin speeds. Each row represents a bearing, in the order given in the bearing definition. If a bearing is not a fluid bearing then a zero eccentricity is returned.

4.2. Calculate Steady State Synchronous Response

This functions calculate the steady state response of the system to unbalance or a bent rotor (i.e. synchronous excitation). The syntax is

```
[response] = freq_rsp(Model,Rotor_Spd)
```

This gives unbalance response or the response of a bent rotor at all nodes (forcing definitions 1, 2 and/or 3; other force definitions are ignored). `Rotor_Spd` is a vector of rotor spin speeds (in rad/s) at which the response is calculated. This also gives operating response if a single rotor spin speed is specified.

4.3. Calculate the Frequency Response Function

For systems where the frequency of the excitation force may be different to the rotational speed the excitation frequency (in rad/s) must also be specified. The set of excitation frequencies is a vector, and the rotor spin speed is assumed to constant. Note that if more than one degree of freedom is excited (defined by the forcing definition) then the force is applied to all degrees of freedom simultaneously, and the forcing definition gives the relative amplitude and phase. For the spinner the `direction` parameter determines if the spinner if rotating in the forwards (positive) or backwards (negative) direction. For excitation through an auxiliary bearing or using a spinner (forcing definitions 6 and 7) the definition is

```
[response] = freq_aux(Model,Rotor_Spd,omega,direction)
```

For the steady state excitation through the foundation (forcing definition 4) the syntax is

```
[response] = freq_fdn(Model,Rotor_Spd,omega)
```

Note that for both of these functions only forcing at a single node is possible, and hence only the first row with the appropriate force definition in `Model.force` is taken, and the other rows are ignored.

4.4. Calculate Critical Speeds

Critical speeds may be calculated in a number of ways, depending on whether frequency dependent bearings properties are present or not (see Section 6.8).

For machines with constant stiffness bearings (bearing types 1 to 6) the direct method may be used, although for machines with damping the results are only approximate. If any of the bearings are of type 7 or 8 then the iterative method will be used. The syntax is

```
[critical_speeds,mode_shape] = crit_spd(Model,NX,damped_NF,
number_criticals)
```

Note that `NX` is the order of the forcing, so that `NX=1` (the default) considers the synchronous force. The parameter `damped_NF` determines whether the damped natural frequency (default, `damped_NF=1`) or natural frequency (`damped_NF=0`) is output as the critical speed. If `number_criticals` is not specified then all the critical speeds are calculated (note only solutions with complex eigenvalues are included). The `critical_speeds` are output in rad/s and the corresponding `mode_shape` are an optional output.

The alternative is the iterative approach. Here the critical speed is estimated, the eigenvalues estimated when the machine is run at this speed, and the critical speed estimate is updated. The procedure continues until convergence, or a preset number of iterations have been performed. The main difficulty with this method is choosing which of the eigenvalues should be chosen to calculate the next estimate of the critical speed, and two methods are possible here. The first picks a fixed eigenvalue number at every iteration, until convergence, and then picks then next eigenvalue number, and so on. The second approach requires the user to enter a vector of initial estimates of critical speeds, and at every iteration for each critical speed, the eigenvalue closest to the current estimate of critical speed is chosen. In both cases there is no guarantee that all of the critical speeds will be obtained, and the Campbell

diagram should be checked to make sure none have been missed. The syntax for the iterative solution is

```
[critical_speeds,mode_shape] = crit_spd(Model,NX,damped_NF,
number_criticals,max_iterations,convergence_tol,initial_estimates)
```

Here `max_iterations` and `convergence_tol` give the maximum iterations (default 20) and convergence tolerance (default 1e-6) for the iterative approach. If the `initial_estimates` are specified then the second approach for choosing the eigenvalue number is used, otherwise the first approach is used.

It should be emphasised that if a machine only has bearings with constant properties (bearing types 1 to 6) then `max_iterations` and `convergence_tol` (and `initial_estimates` if required) must be specified to force the function to use the iterative approach. Otherwise the direct method will be used. If a machine has at least one bearing with speed dependent properties, then the iterative method will always be used, and the defaults will be used as necessary for inputs that are not specified.

4.5. Calculate Time Response to Pulse at Foundation

The function

```
[response,force,time] = time_fdn(Model,Rotor_Spd,dt,npts,nr)
```

calculates the response to a pulse excitation at the foundations (Force Type 5). The properties of the pulse are given in the force definition. `dt` is the time step for the output, and `npts` are the number of time steps. `nr` is the number of degrees of freedom in a reduced model (`nr=0` or if `nr` is not included in the argument list, then the full model is used). The model reduction uses Guyan (static) reduction on the physical degrees of freedom. The outputs are the `response` at all degrees of freedom, the base excitation displacement (`force`) before it is multiplied by the amplitudes in the force definition, and the `time` vector.

4.6. Calculate Rundown and Runup

This calculates the response when running through critical speeds. The syntax is

```
[time,response,speed] = runup(Model,alpha,tspan,nr)
```

The vector `alpha` must of be length 3 and defines a constant acceleration, where the angle of the rotor is given as a function of time, *t*, by

$$\phi = \alpha_1 t^2 + \alpha_2 t + \alpha_3$$

The function is easily edited if different functions of rotor spin speed (for example exponential) are required. `tspan` is a vector of length 2 giving the start and end times of the simulation. `nr` is the number of degrees of freedom in the reduced order model (`nr=0` assumes no reduction). The model is reduced using Guyan (static) reduction based on the mass and stiffness matrices only. Note that this function only works for bearings with constant stiffness. The function uses the MATLAB variable step length ODE solver `ode45.m`

- other ode solvers may be used, such as the stiff ode solver `ode15s.m`, that may not require model reduction. Zero displacement and velocity are assumed as initial conditions. The outputs are the `response` at all degrees of freedom and the rotor spin speed `speed` as function of `time`.

4.7. Calculate Whirl Orbit Properties

Once the response (ODS) or mode shapes have been computed, the properties of the whirl orbits are of great interest. The function `whirl` calculates the whirl direction, and the dimensions of the response ellipse, for a vector of responses. This function is called by other functions that require this information, but is included here in for cases when the function should be called directly. The syntax is

```
[kappa,amplitude] = whirl(u,v)
```

where `u` and `v` are vectors of the response at different speeds (or other conditions), `kappa` is the whirl direction and shape of the orbit as given in the book (Section 3.6.1) and `amplitude` (which is optional) is the length of the semimajor axis.

5. Plotting Functions

Once the results are obtained from the analysis functions, they may be plotted using the standard MATLAB plotting functions. Often to obtain the figures required for reports or papers it is often best to format these directly. However we do provide some functions for particular plot formats that are useful, at least to get started. These functions may be edited for various specific uses if required.

5.1. Plot Schematic of the Model

This functions plots a schematic view of the model to check validity, including the node locations, shaft elements, disks, bearings, etc. The syntax is

```
[] = picrotor(model)
```

5.2. Plot Campbell Diagram

This function produces the Campbell diagram using the calculated results from `chr_root`. The syntax is

```
[] = plotcamp(Rotor_Spd,eigenvalues,NX,damped_NF,kappa)
```

where as usual `Rotor_Spd` is vector of rotor spin speeds (in rad/s), and `eigenvalues` and `kappa` are the eigenvalues and whirl directions calculated by `chr_root`. `NX` determines the range of the forcing lines plotted on Campbell diagram and also the scaling of the natural frequency axis; note that only positive and integer forcing lines are plotted. So, for example, `NX=1` only plots the 1X forcing line, and `NX=2.5` plots the 1X and 2X lines, and the range of the natural frequency axis is 2.5 times the rotor spin speed axis. The default is `NX=1.5`. If `NX` is negative then the 0.5X forcing line is also plotted, which is useful to identify half speed whirl. Note that rotor spin speed is plotted as rev/min and natural frequencies in Hz. The parameter `damped_NF` determines whether the damped natural frequency (default, `damped_NF=1`) or natural frequency (`damped_NF=0`) is plotted. Note that if `kappa` is provided then coloured dots at plotted at every natural frequency for every rotor spin speed to show backward whirl modes, forward whirl modes, or mixed modes.

```
[] = plotcamp(Rotor_Spd,eigenvalues)
```

will give a standard Campbell diagram with just the 1X line and plotting the damped natural frequency.

Note this function sorts the natural frequencies by ascending numerical order, and then plots lines for each natural frequency. If natural frequencies cross then this can create plots that look odd near the crossing points. Although it would be possible to order the natural frequencies based on the underlying physical modes, the easiest method is to refine the increment in rotor spin speeds near the crossing points and replot the Campbell diagram.

5.3. Plot Root Locus

This function plots the real part of the eigenvalues against the imaginary part of the eigenvalues, as the rotor spin speed varies. The syntax is

```
[] = plotloci(Rotor_Spd,eigenvalues,NX)
```

where as usual `Rotor_Spd` is vector of rotor spin speeds (in rad/s), and `eigenvalues` are the eigenvalues calculated by `chr_root`. `NX` determines the scaling of the real and imaginary axes, where the maximum is `NX` times the maximum rotor spin speed.

This function can also suffer from eigenvalues that cross, as described for the Campbell diagram above.

5.4. Plot Mode and ODS Shapes

This function plots the deformations of the rotor as a three dimensional plot, where the orbit at each node is given by an ellipse. Note only the translational degrees of freedom are used. The syntax is

```
[] = plotmode(model,Mode,eigenvalue)
```

where `Mode` is the mode or ODS to plot. Note that this must be a vector and so only one mode may be plotted at a time. The inclusion of the `eigenvalue` is optional; if included then this will be included on the plot as the natural frequency in Hz (the eigenvalue is given in rad/s).

5.5. Plot Orbits

This function plots the two dimensional orbits of the rotor at specified nodes. The syntax is

```
[] = plotorbit(Mode,outputnode,titletext,eigenvalue)
```

where `Mode` is the mode or ODS to plot (must be a vector the same length as the number of degrees of freedom in the system), `outputnode` is a vector of the nodes of interest, and `titletext` is an optional string to add to the orbit plot. Note that only translational degrees of freedom are plotted. The inclusion of the scalar `eigenvalue` (the eigenvalue corresponding to the mode shape) is optional. However for mode shapes it is important that the eigenvector is chosen that corresponds to the eigenvalue with positive imaginary part, otherwise the direction of mode will be incorrect. Including `eigenvalue` in the call to `plotorbit` will check and correct the direction if necessary.

5.5. Plot Frequency Response

This function plots the synchronous response of the rotating machine as a function of rotor spin speed. Its syntax is

```
[] = plotresp(Rotor_Spd,response,outnode)
```

where as usual `Rotor_Spd` is vector of rotor spin speeds (in rad/s), and `response` is the matrix of responses at all degrees of freedom and at all rotor spin speeds, obtained from `freq_rsp.m`. `outnode` gives the nodes and degrees of freedom for the output. The node number is the integer part and decimal part, .1, .2, .3 or .4, gives the relevant degree of freedom at that node. So, for example, 3.2 means node 3 in the *y* direction.

Normally `plotresp` will plot the response, both the amplitude and phase, at all of the degrees of freedom specified. This will not indicate if the rotor is whirling forwards or backwards. There is a special call to `plotresp` when only two degrees of freedom are specified that are the translations or rotations at a single node, for example `outnode = [3.1 3.2]` or `outnode = [5.3 5.4]`. In this case `plotresp` will determine the direction of whirl at the node for every rotor spin speed, and shade those spin speeds where the rotor is whirling backwards.

The function `plotresp` assumes that the excitation frequency is the rotor spin speed. For a spinner or external excitation through an auxiliary bearing the excitation frequency is independent of the rotor spin speed. For a single rotor speed speed the frequency response may be plotted using

```
[] = plotfrf(omega,response,outnode)
```

The only difference between `plotfrf` and `plotresp` is the use of the excitation frequency `omega` (in rad/s). The frequency axis is given on the plot in Hz.

5.5. Plot Time Response

There are no specific functions to plot the time response of the machine. The response at all degrees of freedom and at all times is obtained from the analysis function, together with the time vector. These data are easily plotted using the standard MATLAB functions.

6. Analysis of Coaxial Rotors in the Stationary Frame

The main differences with coaxial rotors is that the rotor spin speed in different rotors may be different, and the coupling between the rotors needs to be specified. The functions available are somewhat limited and designed to illustrate the section in the textbook; however they are easily extended if required.

The definition of node positions is exactly the same as that for single rotors. However nodes on different rotors but at the same axial position should be given different node numbers. The definition of shaft elements and disks is exactly the same as before. There is an extra matrix to define the individual rotors; each row represents a separate rotor and the syntax is

```
model.rotors = [Node_1  Node_2  speed_factor; ...
```

where the nodes on this rotor are between `Node_1` and `Node_2`. The parameter `speed_factor` gives the factor that the reference rotor spin speed is multiplied by to obtain the spin speed of the rotor in question; this may be negative if the rotor spins in the opposite direction to the reference.

There is also an extra bearing type, number 20. This defines the coupling between individual rotors and the syntax is

```
Model.bearing = [20  Node_1  Node_2  kxx  kyy  cxx  cyy ; ...
```

`Node_1` and `Node_2` are the nodes to be coupled on the two rotors. Only translational coupling stiffness and damping coefficients are specified.

There are only two analysis functions. To calculate the eigenvalues and eigenvectors the function `chr_root_coax.m` is used with syntax

```
[eigenvalues,eigenvectors] = chr_root_coax(model,Rotor_Spd)
```

Currently the functions assumes that the rotor spin speed for fluid bearings or seals is the reference rotor speed. If speed dependent bearings are located on multiple rotors then the function will give incorrect results.

Although the eigenvectors are calculated the function to plot modes for single rotors does add extra lines joining the ends of the individual rotors (although this would be relatively easy to correct if required, or the standard function could be used to plot the modes for each rotor in turn).

The synchronous response to unbalance is computed by

```
[response] = freq_rsp_coax(model,Rotor_Spd)
```

Speed dependent bearings again assume that the rotor spin speed is the reference speed.

7. Analysis Functions - Rotating Frame

These functions analyse the machine in the rotating frame and hence assume that the bearings and supports are isotropic. Thus fluid bearings and seals cannot be analysed using these functions and an error will be given. Furthermore bearings with constant stiffness and damping can only be included if they are invariant under the transformation to rotating coordinates. Some checking is undertaken within these functions, but it is not completely robust and care should be exercised to ensure this condition is met.

7.1. Characteristic Roots and the Campbell diagram

To calculate eigenvalues and eigenvectors at a given set of rotor spin speeds we use the following function

```
[eigenvalues,eigenvectors] = chr_asym(Model, Rotor_Spd)
```

Note that the eigenvalues are given in the rotating frame, and may need to be transformed into the stationary frame (see Section 7.3.1). Note that the real parts of complex eigenvalues determine the damping and stability of the solutions.

7.2. Eigenvalue Plots

The function `ploteig.m` plots the real and imaginary parts of the eigenvalues. The syntax is

```
[] = ploteig(Rotor_Spd,eigenvalues,NX)
```

The eigenvalues in the rotating frame are obtained from `chr_asym.m`. These eigenvalues may be transformed to the stationary frame to give the pseudonatural frequencies (see Section 7.3.1); the results may also be plotted using this function. `NX` determines the scale of the frequency axis (imaginary part of the eigenvalue) and adds lines at integer multiples of the rotor speed to the frequency plots.

7.3. Unbalance Response

In the rotating frame unbalance forces and moments are actually static excitations. The function `freq_asym.m` calculates the unbalance response in the rotating frame in the frequency domain. The syntax is

```
[response] = freq_asym(model,Rotor_Spd)
```

where, as usual, the `response` is at all degrees of freedom and all rotor spin speeds given by `Rotor_Spd`.

8. Other Analysis Functions

8.1. Calculating the Frequency Response using the DFT

Sometime the time response has been computed and this has to be transformed to the frequency domain to highlight the frequency content of the response. MATLAB has a function `fft.m` that computes the DFT, but doesn't scale the result consistently and doesn't return the frequency vector. The function `fftscale.m` has been written to achieve this with syntax

```
[fft_out,omega] = fftscale(response,time)
```

Each row of the `response` corresponds to a degree of freedom, and the number of columns must be the same as the length of the `time` vector. The `time` vector is assumed to have equal time increments. `fft_out` is the scaled response in the frequency domain and `omega` is the corresponding frequency vector in Hz.

9. Tutorial Example

Most of the examples in the book have scripts that reproduce the results and the plots. Example 5.9.1 is given here to demonstrate how the software is used.

```
% File name:   Example_05_09_01.m
%
% Example 5.9.1
%
% This example has isotropic bearings
% A model with 6 Timoshenko beam elements
%

clear
close all


% Set the material parameters
E = 211e9;
G = 81.2e9;
rho = 7810;
damping_factor = 0;    % no damping in shaft


% Consider a model with 6 equal length elements
% Shaft is 1.5m long
model.node = [1 0.0; 2 0.25; 3 0.5; 4 0.75; 5 1.0; 6 1.25; 7 1.50];

% Assume shaft type 2 - Timoshenko with gyroscopic effects included
% Solid shaft with 50mm outside diameter
shaft_od = 0.05;
shaft_id = 0.0;
model.shaft = [2 1 2 shaft_od shaft_id rho E G damping_factor; ...
               2 2 3 shaft_od shaft_id rho E G damping_factor; ...
               2 3 4 shaft_od shaft_id rho E G damping_factor; ...
               2 4 5 shaft_od shaft_id rho E G damping_factor; ...
               2 5 6 shaft_od shaft_id rho E G damping_factor; ...
               2 6 7 shaft_od shaft_id rho E G damping_factor];

% Disk 1 at node 3 has diameter of 280mm and thickness of 70mm
% Disk 2 at node 5 has diameter of 350mm and thickness of 70mm
% Note inside diameter of disk is assumed to be the outside diameter
% of the shaft
disk1_od = 0.28;
disk2_od = 0.35;
disk_thick = 0.07;
model.disc = [1 3 rho disk_thick disk1_od shaft_od; ...
              1 5 rho disk_thick disk2_od shaft_od];

% Constant stiffness short isotropic bearing (1NM/m) with no damping
% Bearings at the ends of the shaft - nodes 1 and 7
bear_stiff = 1e6;
model.bearing = [3 1 bear_stiff bear_stiff 0 0; ...
                 3 7 bear_stiff bear_stiff 0 0];
```

```
% Draw the rotor
figure(1), clf
picrotor(model)


% Plot the Campbell diagram
% =========================

% Define the rotor spin speed range
Rotor_Spd_rpm = 0:100:4500.0;
Rotor_Spd = 2*pi*Rotor_Spd_rpm/60;  % convert to rad/s

% Calculate the eigensystem for the range of rotor spin speeds
[eigenvalues,eigenvectors,kappa] = chr_root(model,Rotor_Spd);

% Plot Campbell diagram
figure(2)
NX = 2;
damped_NF = 1; % plot damped natural frequencies
plotcamp(Rotor_Spd,eigenvalues,NX,damped_NF,kappa)


% Plot the modes and orbits at 4000 rev/min
% =========================================

% Calculate the eigensystem at 4000 rev/min
Rotor_Spd_rpm = 4000;
Rotor_Spd = 2*pi*Rotor_Spd_rpm/60;  % convert to rad/s
[eigenvalues,eigenvectors,kappa] = chr_root(model,Rotor_Spd);

% Plot the first 4 eigenvectors and annotate with corresponding
% eigenvalue
figure(3)
subplot(221)
plotmode(model,eigenvectors(:,1),eigenvalues(1))
subplot(222)
plotmode(model,eigenvectors(:,3),eigenvalues(3))
subplot(223)
plotmode(model,eigenvectors(:,5),eigenvalues(5))
subplot(224)
plotmode(model,eigenvectors(:,7),eigenvalues(7))

% Plot the orbits at the disks for the first 6 eigenvectors
% Note that the axes command is used here for different subplots -
% the MATLAB command subplot could also have been used
figure(4)
outputnode = [3 5];
axes('position',[0.2 0.53 0.2 0.2 ])
plotorbit(eigenvectors(:,1),outputnode,'Mode 1',eigenvalues(1))
axes('position',[0.39 0.53 0.2 0.2 ])
plotorbit(eigenvectors(:,3),outputnode,'Mode 2',eigenvalues(3))
axes('position',[0.58 0.53 0.2 0.2 ])
plotorbit(eigenvectors(:,5),outputnode,'Mode 3',eigenvalues(5))
axes('position',[0.2 0.25 0.2 0.2 ])
plotorbit(eigenvectors(:,7),outputnode,'Mode 4',eigenvalues(7))
axes('position',[0.39 0.25 0.2 0.2 ])
plotorbit(eigenvectors(:,9),outputnode,'Mode 5',eigenvalues(9))
axes('position',[0.58 0.25 0.2 0.2 ])
plotorbit(eigenvectors(:,11),outputnode,'Mode 6',eigenvalues(11))
```