DOI: 10.21105/joss.02120
The Journal of Open Source Software

- Review
- Repository
- Archive

Editor: Marie E. Rognes
Reviewers:

- @srmnitc
- @nnadeau

Submitted: 02 December 2019
Published: 09 April 2020

ROSS - Rotordynamic Open Source Software

Raphael Timbó[1], Rodrigo Martins[2], Gabriel Bachmann[4], Flavio Rangel[5], Júlia Mota[3], Juliana Valério[5], and Thiago G Ritto[2]

1 Petrobras - Petróleo Brasileiro S.A. **2** Universidade Federal do Rio de Janeiro, Department of Mechanical Engineering, Rio de Janeiro, Brazil **3** Universidade Federal do Rio de Janeiro, Graduate Program in Informatics, Rio de Janeiro, Brazil **4** Universidade Federal do Rio de Janeiro, Department of Electrical Engineering, Rio de Janeiro, Brazil **5** Universidade Federal do Rio de Janeiro, Department of Computer Science, Rio de Janeiro, Brazil

Summary

There are several categories of critical rotating equipment crucial to industry, such as compressors, pumps, and turbines. Computational mechanical models aim to simulate the behavior of such mechanical systems and these models are used to support research and decision making. To this purpose, we present ROSS, an open source library written in Python for rotordynamic analysis.

Existing tools that have rotordynamic functionalities include commercial finite element software with rotordynamic modules ("ANSYS - Mechanical Rotordynamics," 2019; "COMSOL - Rotordynamics Module," 2019), packages based on proprietary runtimes (MATLAB) ("Dynamics of Rotating Machines," 2019; "MADYN 2000," 2019), and some standalone tools ("ROTORINSA," 2019; "XLTRC2," 2019). To use all of these options however requires the purchase of a license and they are not intended to be developed in an open, collaborative manner. Additionally for some of these commercial packages, the user is 'locked in' to the environment, interacting with the software only through a graphical user interface, which makes it harder (impossible sometimes) to automate analysis.

To our knowledge, ROSS is the first software being developed using the open source paradigm in rotordynamic field, with the code being clearly licensed and fully available on code hosting platforms, issues tracked online, and the possibility of direct contribution by the community.

ROSS allows the construction of rotor models and their numerical simulation. Shaft elements, as a default, are modeled with the Timoshenko beam theory (Hutchinson, 2001), which considers shear and rotary inertia effects, and discretized by means of the Finite Element Method (Friswell, Penny, Garvey, & Lees, 2010). Disks (impellers, blades, or other equipment attached to the rotor) are assumed to be rigid bodies, thus their strain energy is not taken into account and only the kinetic energy due translation and rotation is calculated. We can obtain the mass and gyroscopic matrices by applying Lagrange's equations to the total kinetic energy.

The mass matrix is given by:

$$\mathbf{M_e} = \begin{bmatrix} m_d & 0 & 0 & 0 \\ 0 & m_d & 0 & 0 \\ 0 & 0 & I_d & 0 \\ 0 & 0 & 0 & I_p \end{bmatrix} \tag{1}$$

The gyroscopic matrix is given by:

$$\mathbf{G_e} = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & I_p \\ 0 & 0 & -I_p & 0 \end{bmatrix} \tag{2}$$

Where:

- m_d is the disk mass;
- I_d is the diametral moment of inertia;
- I_p is the polar moment of inertia.

For most types of bearing, the load-deflection relationship is nonlinear. Furthermore, load-deflection relationships are often a function of shaft speed (i.e $\mathbf{K_e} = \mathbf{K_e}(\omega)$ and $\mathbf{C_e} = \mathbf{C_e}(\omega)$). To simplify dynamic analysis, one widely used approach is to assume that the bearing has a linear load-deflection relationship. This assumption is reasonably valid, provided that the dynamic displacements are small (Friswell et al., 2010). Thus, the relationship between the forces acting on the shaft due to the bearing and the resultant velocities and displacements of the shaft may be approximated by:

$$\begin{Bmatrix} f_x \\ f_y \end{Bmatrix} = -\begin{bmatrix} k_{xx} & k_{xy} \\ k_{yx} & k_{yy} \end{bmatrix} \begin{Bmatrix} u \\ v \end{Bmatrix} - \begin{bmatrix} c_{xx} & c_{xy} \\ c_{yx} & c_{yy} \end{bmatrix} \begin{Bmatrix} \dot{u} \\ \dot{v} \end{Bmatrix} \tag{3}$$

where f_x and f_y are the dynamic forces in the x and y directions, and u and v are the dynamic displacements of the shaft journal relative to the bearing housing in the x and y directions.

After defining the element matrices, ROSS performs the assembling of the global matrices and the general form of the equation of the system is

$$\mathbf{M}\ddot{\mathbf{q}}(t) + \mathbf{C}(\Omega)\dot{\mathbf{q}}(t) + \omega\mathbf{G}\dot{\mathbf{q}}(t) + \mathbf{K}(\Omega)\mathbf{q}(t) = \mathbf{f}(t)\,, \tag{4}$$

where:

- $\mathbf{q}$ is the generalized coordinates of the system (displacements and rotations);
- $\mathbf{M}$ is the mass matrix;
- $\mathbf{K}$ is the stiffness matrix;
- $\mathbf{C}$ is the damping matrix;
- $\mathbf{G}$ is the gyroscopic matrix;
- Ω is the excitation frequency;
- ω is the rotor whirl speed;
- $\mathbf{f}$ is the generalized force vector.

After building a model with ROSS, the user can plot the rotor geometry, run simulations, and obtain results in the form of graphics. ROSS can perform several analyses, such as static analysis, whirl speed map, mode shapes, frequency response, and time response.

ROSS is extensible and new elements, such as different types of bearings or seals, can be added to the code. As an example, one can add a class for a tapered roller bearing by inheriting from `BearingElement`. The implementation of the `BallBearingElement` in our code uses this strategy.

Other elements that require more customization can be added by inheriting directly from `Element`, in this case it is necessary to implement the required methods that should return the element's mass, stiffness, damping, and gyroscopic matrices.

We have built the package using main Python packages such as NumPy (Walt, Colbert, & Varoquaux, 2011) for multi-dimensional arrays, SciPy (Virtanen et al., 2020) for linear algebra, optimization, interpolation and other tasks and Bokeh (Bokeh Development Team, 2019) for creating interactive plots. Developing the software using Python and its scientific ecosystem enables the user to also make use of this ecosystem, making it easier to run rotordynamics analysis. It is also easier to integrate the code into other programs, since we only use open source packages and do not depend on proprietary commercial platforms.

Besides the documentation, a set of Jupyter Notebooks are available for the tutorial and some examples. Users can also access these notebooks through a Binder server.

As an example, Figure 1 shows a centrifugal compressor modeled with ROSS.

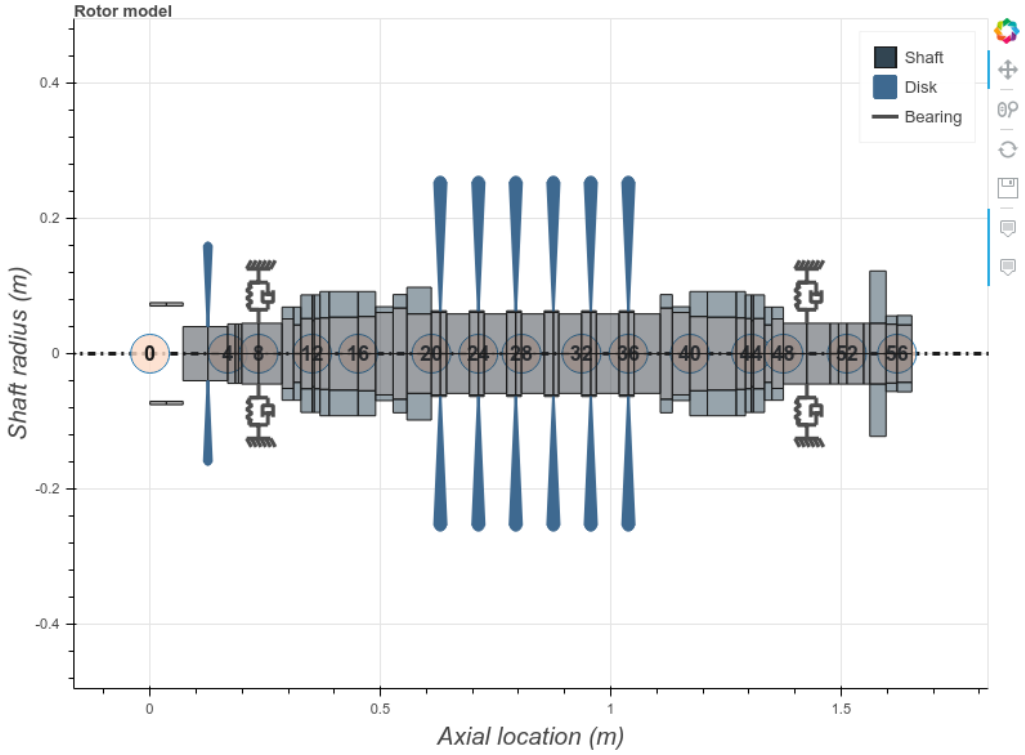

Figure 1: Centrifugal Compressor modeled with ROSS.

The shaft elements are in gray, the impellers represented as disks are in blue and the bearings are displayed as springs and dampers. This plot is generated with Bokeh, and we can use the hover tool to get additional information for each element.

One analysis that can be carried out is the modal analysis. Figure 2 shows the whirl speed map (Campbell Diagram) generated for this compressor; the natural frequencies and the log dec vary with the machine rotation speed.

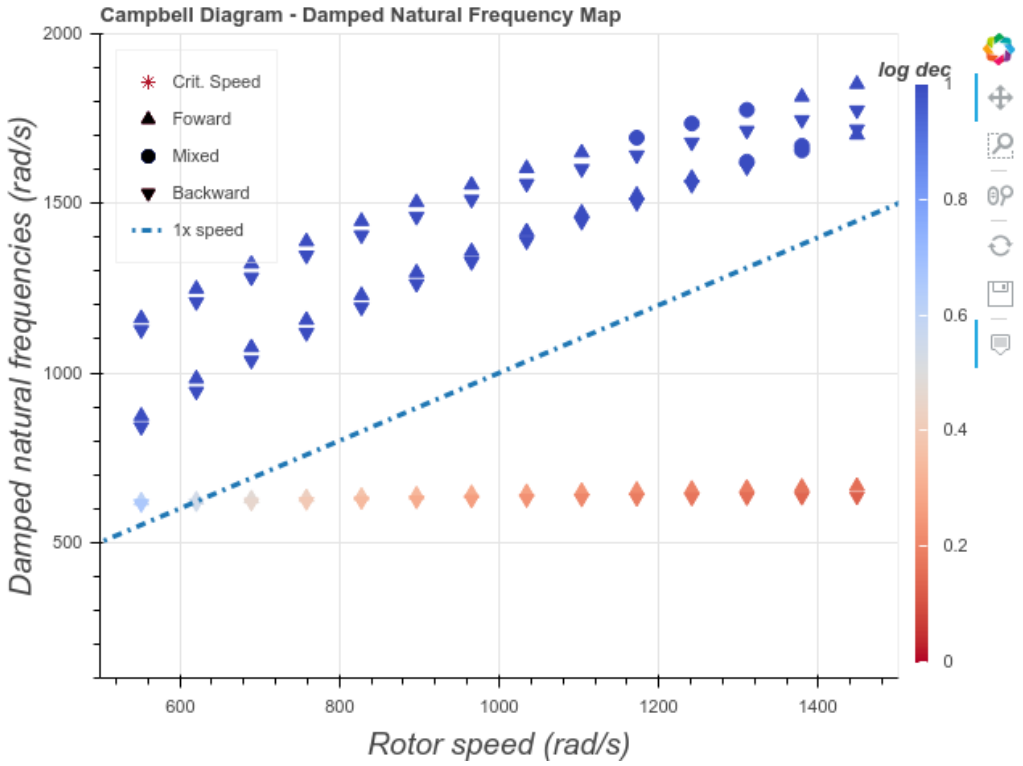

Figure 2: Whirl speed map (Campbell Diagram) for the Centrifugal Compressor.

The whirl speed map is one of the results that can be obtained from the model, other types of analyses can be found in the documentation.

ROSS has been used to evaluate the impact of damper seal coefficients uncertainties in a compressor rotordynamics (Timbó & Ritto, 2019). Other projects are also using ROSS to test machine learning algorithms that can diagnose machine failure. Here we can create a model with some problem such as unbalance or misalignment and then use the output data from this model to test the learning algorithm and check the accuracy of this algorithm in diagnosing the problem.

Acknowledgements

We acknowledge that ROSS development is supported by Petrobras, Universidade Federal do Rio de Janeiro (UFRJ) and Agência Nacional de Petróleo, Gás Natural e Biocombustíveis (ANP).

References

ANSYS - Mechanical Rotordynamics. (2019). Retrieved from https://www.ansys.com/services/training-center/structures/ansys-mechanical-rotordynamics

Bokeh Development Team. (2019). *Bokeh: Python library for interactive visualization*. Retrieved from https://bokeh.org/

COMSOL - Rotordynamics Module. (2019). Retrieved from https://www.comsol.com/blogs/tag/rotordynamics-module

Dynamics of Rotating Machines. (2019). Retrieved from http://www.rotordynamics.info/

Friswell, M., Penny, J., Garvey, S., & Lees, A. (2010). *Dynamics of rotating machines*. Cambridge University Press. doi:10.1017/CBO9780511780509

Hutchinson, J. R. (2001). Shear Coefficients for Timoshenko Beam Theory. *Journal of Applied Mechanics*, *68*(1), 87–92. doi:10.1115/1.1349417

MADYN 2000. (2019). Retrieved from http://www.delta-js.ch/en/software/

ROTORINSA. (2019). Retrieved from http://lamcos.insa-lyon.fr/logiciels_rotorinsa.php?L=2

Timbó, R., & Ritto, T. G. (2019). Impact of damper seal coefficients uncertainties in rotor dynamics. *Journal of the Brazilian Society of Mechanical Sciences and Engineering*, *41*(4), 165. doi:10.1007/s40430-019-1652-8

Virtanen, P., Gommers, R., Oliphant, T. E., Haberland, M., Reddy, T., Cournapeau, D., Burovski, E., et al. (2020). SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. *Nature Methods*, *17*, 261–272. doi:10.1038/s41592-019-0686-2

Walt, S. van der, Colbert, S. C., & Varoquaux, G. (2011). The NumPy array: A structure for efficient numerical computation. *Computing in Science and Engineering*, *13*(2), 22–30. doi:10.1109/MCSE.2011.37

XLTRC2. (2019). Retrieved from https://turbolab.tamu.edu/trc-software/