PyOMA2: An Open-Source Python Software for Operational Modal Analysis

PyOMA2: an open-source python software for Operational Modal Analysis

Dag Pasquale Pasca[1,*][0000−0002−3830−2835], Diego Federico Margoni[3], Marco M. Rosso[3][0000−0002−9098−4132], and Angelo Aloisio[2][0000−0002−6190−0139]

[1] Norsk Treteknisk Institutt, Børrestuveien 3, 0373, Oslo, Norway
[2] Civil Environmental and Architectural Engineering Department, Università degli Studi dell'Aquila, via Giovanni Gronchi n.18, L'Aquila, 67100, Italy
[3] Politecnico di Torino, DISEG, Department of Structural, Geotechnical and Building Engineering, Corso Duca Degli Abruzzi, 24, Turin 10128, Italy
Corresponding Author: dpa@treteknsik.no

Abstract. In this paper, we introduce the new `pyOMA2` Python module, the updated version of `pyOMA`, an open-source Python module designed for Operational Modal Analysis (OMA). This new iteration transforms `pyOMA` from a basic function library into a sophisticated module, fully leveraging Python's class capabilities. `pyOMA2` expands its scope to support both single and multi-setup data measurements, effectively managing multiple acquisitions involving a combination of reference and roving sensors. A notable addition is interactive plotting, enabling users to directly select modes for extraction from algorithm-generated plots. Furthermore, a novel feature allows users to define the geometry of the structures under analysis, enhancing the visualization of mode shapes post-analysis. The core functions of these classes have undergone thorough revisions, leading to substantial improvements and optimizations. `pyOMA2` not only retains the algorithms found in its predecessor, including Frequency Domain Decomposition (FDD), Enhanced Frequency Domain Decomposition (EFDD), Frequency-Spatial Domain Decomposition (FSDD), and both the covariance-based and data-driven Stochastic Subspace Identification (SSIcov, SSIdat) methods but also introduces the poly-reference Least Square Frequency Domain (pLSCF) method, also known as Polymax. The inclusion of these advanced algorithms, along with the module's improved features, makes `pyOMA2` a valuable resource in the field of structural dynamics, providing researchers and practitioners with an effective tool for conducting operational modal analysis.

Keywords: operational modal analysis · open-source software · python · structural health monitoring

1 Introduction

Operational Modal Analysis (OMA) has rightfully gained significant attention from the engineering community in recent years. It has become the de facto standard for estimating the modal properties of structures in Structural Health

Monitoring (SHM) applications. The primary advantage of OMA over Experimental Modal Analysis (EMA) lies in its ability obtain the modal parameters from output-only measurements taken during the structure's regular service. This approach offers a more practical and efficient means of assessing structural health compared to the more traditional EMA method, which requires both input and output data.

In this contribution, the authors introduce `pyOMA2`, a Python module specifically developed for OMA applications. This module incorporates several significant enhancements and novel features compared to its predecessor `pyOMA` [14], improving its usability and functionality in the field. The source code is accessible from the following GitHub repository (`https://github.com/dagghe/PyOMA2`).

The main novelties introduced by the new release are:

- **Comprehensive Code Revision:** The underlying algorithms have undergone a thorough overhaul for more efficient and robust performance.
- **Specialized Class Structures:** Includes specialized classes for handling both single-setup and multi-setup measurements, enhancing data management versatility.
- **Interactive Plotting Feature:** An optional interactive plotting capability allows users to select desired modes directly from the algorithm's output plots, improving user interaction and mode selection.
- **Geometry Definition Option:** Additional feature for optional geometry definition to enable plotting and animation of mode shapes, providing a comprehensive understanding of results post-analysis.

The `pyOMA` module, originally developed by Dag Pasca with contributions from Angelo Aloisio, celebrated its first release in February 2021. A significant enhancement to the module came with the introduction of a GUI executable version in March 2022. This advancement was made possible thanks to the diligent efforts of the ArtiStE team from Politecnico di Torino, who had by that time joined the project. Now, three years post its initial launch, `pyOMA` has grown into a sophisticated and reliable tool, offering a comprehensive range of functionalities for professionals and researchers in the fields of OMA and SHM.

The genesis of the `pyOMA` project is a testament to the power of collaborative efforts, bringing together expertise from three institutions: the Norwegian Institute of Wood Technology (NTI) in Norway, the University of L'Aquila in Italy, and Politecnico di Torino, also in Italy. It is important to acknowledge that significant inspiration, particularly for the interactive plots, was drawn from the open-source module pyEMA (`https://github.com/ladisk/pyEMA`).

In the following section, the main contents of the module are described. Thereafter, two practical examples are presented to demonstrate the available functionalities.

2 PyOMA package description

Figure 1 presents a general schematic of the module's structure and the inheritance relationships between classes. The module is structured into three primary levels:

1. At the first level are the *setup classes*. Users instantiate these classes by providing a data array and the sampling frequency for a single setup scenario, or a list of data arrays and their respective sampling frequencies for a multi-setup scenario.
2. The second level comprises the *algorithm classes*. Users can instantiate the algorithms they wish to run and then add them to the setup class.
3. The third level contains the *support classes*, which serve as auxiliary components to the first two levels. This level includes various specialized classes:
   - *Result* classes, where outcomes are stored.
   - *Geometry* classes, for storing geometric data.
   - *Run_params* classes, where parameters used for running the algorithms are kept.
   - Dedicated classes for animating mode shapes and interacting with plots generated by the algorithm classes.

In addition to the levels depicted in the figure, there is a further level not shown, comprised of the set of functions internally called by the class methods. These functions represent an updated version of those available in our previous release, `pyOMA`. For users interested in a more direct approach, it's still possible to import only the functions and execute them sequentially to obtain the results. However, this approach precludes the use of interactive plots and the animation capabilities provided by the dedicated classes. Collectively, all these layers work together to create a robust and comprehensive framework, facilitating efficient module operation and data processing.

As the previous version this one also includes the following algorithms to perform OMA:

1. Frequency domain decomposition (FDD);
2. Enhanced frequency domain decomposition (EFDD);
3. Frequency spatial domain decomposition (FSDD);
4. Reference based covariance driven stochastic subspace identification (SSIcov);
5. Reference based data driven stochastic subspace identification (SSIdat);

For the readers seeking a more comprehensive understanding of the algorithms and the underlying theory, a wealth of literature is available for consultation. Key references include works by Brincker [9,7,8], Zhang [24], Peeters [15,16], Van Overschee and De Moor [23], Reynders [18], Rainieri and Fabbrocino [17], Amador [6], Döhler [11,12,13], and others.

The module's reliability and applicability for research purposes have been demonstrated by the authors through various studies, such as Alaggio et al. [2],

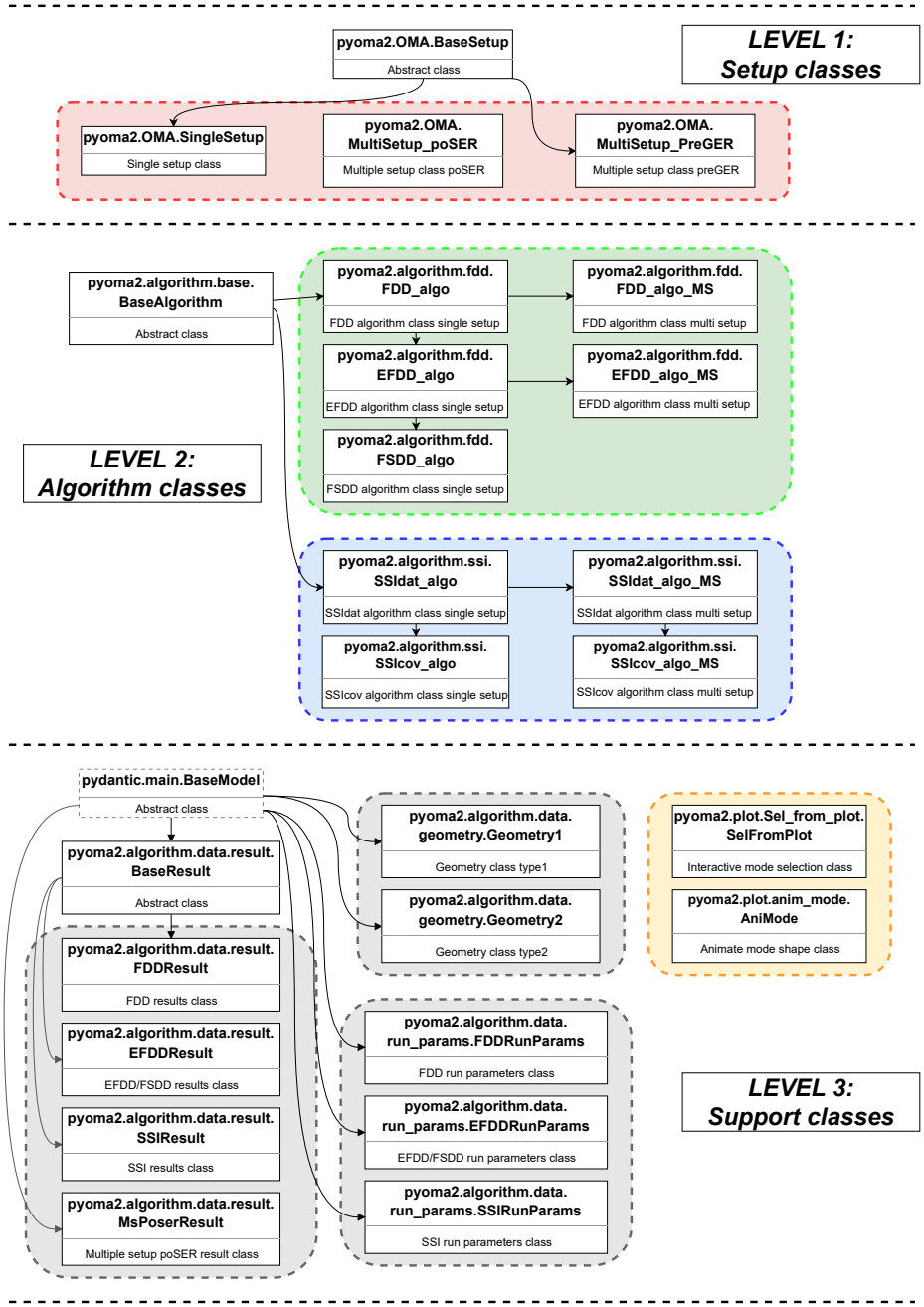

Fig. 1: Schematic organisation of the module showing inheritance between classes.

Aloisio et al. [4,3], Rosso et al. [19], Simoncelli et al. [21], and more. Additionally, the module has gained traction within the research community, as evidenced by its use in studies by Saharan et al. [20], Croce et al. [10], Talebi et al. [22], Abuodeh et al. [1], and others.

3 Single setup example

In this initial example, we showcase the module's functionalities and workflow, illustrating its application using a dataset included with the module. This dataset originates from a real-life experiment on an 8-story Cross-Laminated Timber (CLT) building in Ås, Norway. For a more comprehensive understanding of the structure and its data acquisition system, readers may refer to the study by Aloisio et al. [5].

The process begins by importing the necessary modules. Once the dataset is imported and assigned to a variable, the next step is to create an instance of the `SingleSetup` class, providing the dataset and the sampling frequency as parameters. To visualize the time histories of the channels, the `plot_data()` method can be utilized. Additionally, the `plot_ch_info()` method offers insights into the quality of the data acquisition see Figure 2. Moreover, the `plot_STFT()` method enables the plotting of a Short Time Fourier Transform magnitude plot (Spectrogram), facilitating the assessment of the signal's stationarity.

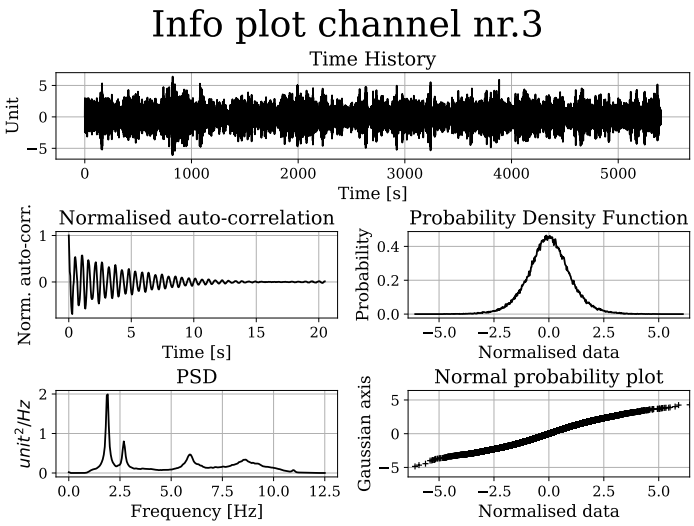

Fig. 2: Plot of dataset's time histories.

If the geometry files have been successfully imported and assigned to variables, it becomes possible to define the structure's geometry using two distinct

methods, each offering unique plotting capabilities. The first method, `def_geo1()`, enables users to visualise mode shapes with arrows that represent the placement, direction, and magnitude of displacement for each sensor. On the other hand, `def_geo2()` allows for the plotting and animation of mode shapes, with sensors mapped to user defined points. Subsequently, the `plot_geo1()` and/or `plot_geo2()` methods can be used to generate visualisations of the geometry and the placement of sensors.

We are now ready to instantiate the algorithm classes of our choice, such as `EFDD_algo` and `SSIcov_algo`. These can be added to the setup class using the `add_algorithms()` method. Subsequently, the algorithms can be executed either individually using the `run_by_name()` method or collectively with `run_all()`.

After executing the algorithms, a variety of plotting methods can be employed to visualise the results. Methods such as `plot_CMIF()`, `plot_STDiag()`, and are used to produce the plots depicted in Figures 3 a) and b), respectively. Notably, the plots in Figures 3 a) and b) can also be interactively accessed through the `MPE_fromPlot()` method. This feature allows users to directly extract the desired modes from the plots. Additionally, the modal properties can also be manually extracted using the `MPE()` method.

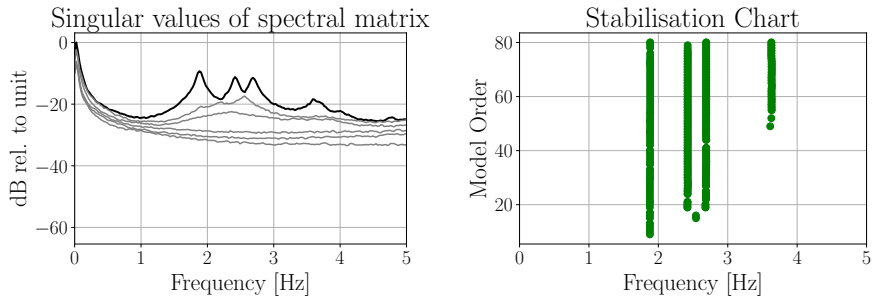

Fig. 3: a) Plot of the singular values of the PSD matrix; b) Stabilisation diagram.

Once the modes have been extracted, the results can be efficiently stored in dictionaries for ease of access. For deeper insights into the EFDD and FSDD algorithms, the `plot_FIT()` method can be used for visualisation, as shown in Figure 4. Users also have the option to plot or animate the mode shapes using the `plot_mode_g1()`, `plot_mode_g2()`, and `anim_mode_g2()` methods. It is important to note that the animation feature for mode shapes is available only when the geometry is defined using the `def_geo2()` method, as previously mentioned. An example of the output from `plot_mode_g2()` is showcased in Figure 5.

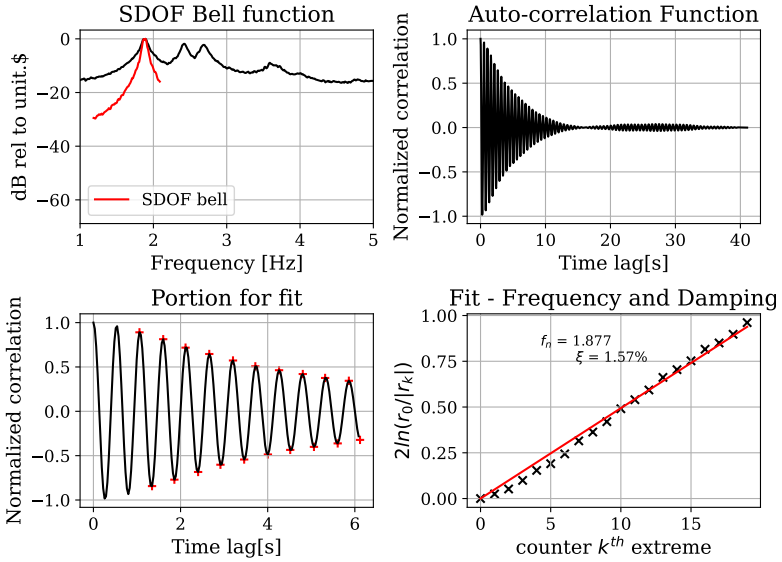

Fig. 4: Detailed results from EFDD and FSDD algorithms.

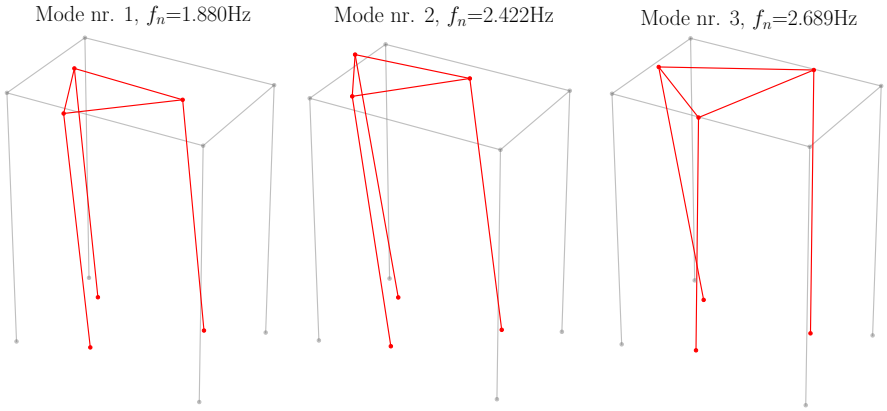

Fig. 5: First three mode shapes of the structure.

4 Multi setup example

The primary advantage of conducting a multi-setup experiment lies in the potential to obtain more detailed mode shapes compared to a single-setup scenario. However, a disadvantage is the necessity of data processing. As outlined in the literature [13,11,15,6], there are two main methods for this. The first, known as post Single Estimation Rescaling (poSER), involves running each setup individually and then merging the mode shapes. This is done by using values from the reference index to find coefficients that scale the mode shapes to a uniform level. The second method, pre Global Estimation Rescaling (preGER), involves scaling and merging either the spectral densities or the Hankel/observability matrices to directly obtain a global estimate of all the modal properties. This latter approach is more time-efficient and less prone to user errors compared to the first one.

To illustrate a multi-setup scenario effectively, we generated the datasets using a Finite Element Method (FEM) model, developed with the openseespy software [25]. The Python script for generating these datasets is provided in our GitHub repository for reference. The authors conducted three distinct simulations, during which we recorded the accelerations at points corresponding to accelerometers placed at the model's nodes. Each simulation utilised ten sensors, consisting of three reference sensors and seven roving sensors. This configuration resulted in a total of twenty-four unique values within the mode shape vector. Random noise was introduced to the data at the conclusion of the simulations. The geometry and position of the recorded points is shown in Figure 6

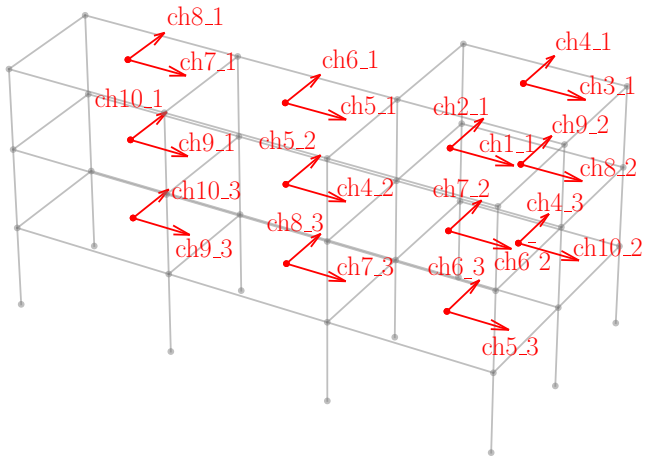

Fig. 6: Geometry and sensors placement for the multi setup example.

Next, we outline the workflow for employing the two distinct classes of multi-setup analysis that our module offers.

4.1 poSER

For the poSER approach, after importing the necessary modules and data, we need to create as many `SingleSetup` (SS) class instances as there are datasets available. The process for obtaining the modal properties from each setup remains the same as described in the previous section. After analyzing all datasets, the `MultiSetup_PoSER` (poSER) class can be instantiated by passing the processed SS and the lists of reference indices. Subsequently, the `merge_results()` method is used to combine the results. However, there are certain prerequisites for instantiating the poSER class. Firstly, all SS classes must have executed the same number and type of algorithm. Secondly, each SS class should contain only one instance of each type of algorithm. Finally, it is the user's responsibility to ensure consistency between the modes extracted from each SS. This step is crucial for the validity of the final merged results in the multi-setup analysis.

Regarding the geometry, it needs to be defined within the poSER class by invoking the `def_geo1()` and/or `def_geo2()` methods, similar to the procedure in the SS case. After successfully merging the results, we gain also access to the various methods for visual representation, such as `plot_mode_g1()`, `plot_mode_g2()`, and `anim_mode_g2()`. For instance, an example of the visualisation generated by `plot_mode_g1()` is demonstrated in Figure 6.

4.2 preGER

For the preGER merging procedure, we adopt a strategy similar to that used for the SS class. The first step involves instantiating the `MultiSetup_PreGER` class and passing the list of datasets, the lists of reference sensors, and their sampling frequencies. Subsequently, we need to instantiate the multi-setup versions of the algorithms we wish to execute, such as `SSIdat_algo`. Similar to the SS class, the methods `add_algorithms()`, `run_by_name()`, and `run_all()` are employed to add and run the chosen algorithm(s).

After running the algorithms, we gain access to methods like `plot_STDiag()`, `MPE_fromPlot()`, and `MPE()`. These are used for plotting the stabilisation diagram of the merged datasets and for extracting the modes either directly from the plot or manually. An example of a stabilisation diagram produced by this process is presented in Figure 7.

Similar to the poSER method, a global geometry must be defined within the preGER class. Afterwards, the mode shapes can be visualised in the same manner as with the other methods. For instance, an example of some mode shapes extracted using the SSIdat algorithm is illustrated in Figure 8.

5 Conclusions

In conclusion, PyOMA2 marks a significant advancement in the field of Operational Modal Analysis by offering an enhanced, open-source Python module that facilitates more efficient and comprehensive data analysis. Its class-based system,

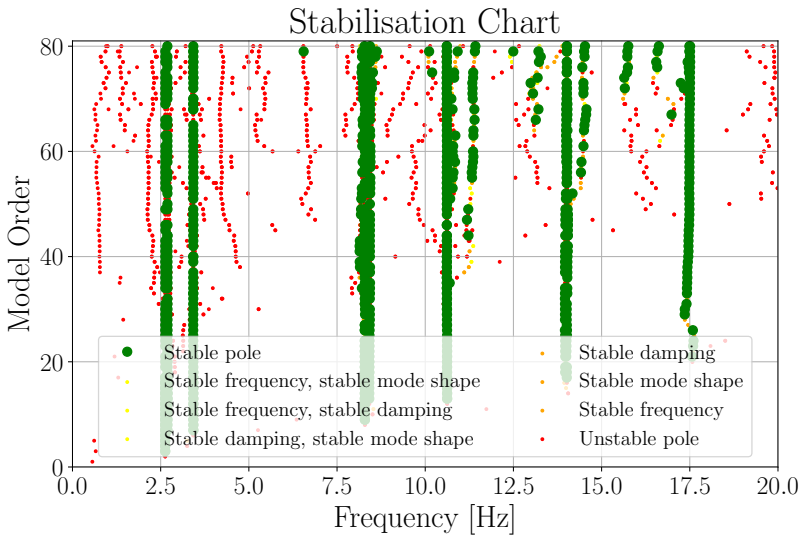

Fig. 7: Stabilisation diagram from SSI on merged datasets.

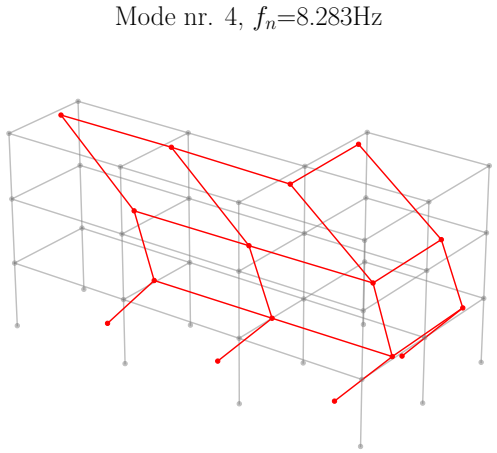

Fig. 8: Global mode shape of the structure obtained with preGER procedure.

interactive plotting features, and ability to handle both single and multi-setup data measurements greatly improve user interaction and analytical capabilities. This tool not only simplifies complex analyses but also promotes wider accessibility and collaboration in the scientific community. The development of PyOMA2 underscores the pivotal role of open-source resources in driving innovation and research in engineering and structural health monitoring, setting a new standard for future developments in the field.

References

1. Abuodeh, O., Locke, W., Redmond, L., Sreenivasulu, R.V., Schmid, M.: Examining methods for modeling road surface roughness effects in vehicle–bridge interaction models via physical testing. In: Society for Experimental Mechanics Annual Conference and Exposition. pp. 33–47. Springer (2023)
2. Alaggio, R., Aloisio, A., Antonacci, E., Cirella, R.: Two-years static and dynamic monitoring of the santa maria di collemaggio basilica. Construction and Building Materials **268**, 121069 (2021)
3. Aloisio, A., Alaggio, R., Fragiacomo, M.: Time-domain identification of the elastic modulus of simply supported box girders under moving loads: Method and full-scale validation. Engineering Structures **215**, 110619 (2020)
4. Aloisio, A., Di Pasquale, A., Alaggio, R., Fragiacomo, M.: Assessment of seismic retrofitting interventions of a masonry palace using operational modal analysis. International Journal of Architectural Heritage pp. 1–13 (2020)
5. Aloisio, A., Pasca, D., Tomasi, R., Fragiacomo, M.: Dynamic identification and model updating of an eight-storey clt building. Engineering Structures **213**, 110593 (2020)
6. Amador, S.D., Brincker, R.: Robust multi-dataset identification with frequency domain decomposition. Journal of Sound and Vibration **508**, 116207 (2021)
7. Brincker, R., Ventura, C.: Introduction to operational modal analysis. John Wiley & Sons (2015)
8. Brincker, R., Ventura, C.E., Andersen, P.: Damping estimation by frequency domain decomposition. In: Proceedings of IMAC 19: A conference on structural dynamics: Februar 5-8, 2001, Hyatt Orlando, Kissimmee, Florida, 2001. pp. 698–703. Society for Experimental Mechanics (2001)
9. Brincker, R., Zhang, L., Andersen, P.: Modal identification of output-only systems using frequency domain decomposition. Smart materials and structures **10**(3), 441 (2001)
10. Croce, T., Girardi, M., Gurioli, G., Padovani, C., Pellegrini, D.: Towards a cloud-based platform for structural health monitoring: Implementation and numerical issues. In: International Conference on Experimental Vibration Analysis for Civil Engineering Structures. pp. 610–619. Springer (2023)
11. Döhler, M., Jaishi, B., Mevel, L., Brownjohn, J.M.: Data fusion for system identification of the humber bridge. In: Civil Engineering Topics, Volume 4: Proceedings of the 29th IMAC, A Conference on Structural Dynamics, 2011. pp. 83–98. Springer (2011)
12. Döhler, M., Mevel, L.: Fast multi-order computation of system matrices in subspace-based system identification. Control Engineering Practice **20**(9), 882–894 (2012)

13. Döhler, M., Mevel, L.: Modular subspace-based system identification from multi-setup measurements. IEEE Transactions on Automatic Control **57**(11), 2951–2956 (2012)
14. Pasca, D.P., Aloisio, A., Rosso, M.M., Sotiropoulos, S.: Pyoma and pyoma_gui: A python module and software for operational modal analysis. SoftwareX **20**, 101216 (2022)
15. Peeters, B., Dammekens, F., Magalhães, F., Van der Auweraer, H., Caetano, E., Cunha, A.: Multi-run operational modal analysis of the guadiana cable-stayed bridge. Proceedings of IMAC XXIV (2006)
16. Peeters, B., De Roeck, G.: Reference-based stochastic subspace identification for output-only modal analysis. Mechanical systems and signal processing **13**(6), 855–878 (1999)
17. Rainieri, C., Fabbrocino, G.: Operational modal analysis of civil engineering structures. Springer, New York **142**, 143 (2014)
18. Reynders, E.: System identification methods for (operational) modal analysis: review and comparison. Archives of Computational Methods in Engineering **19**, 51–124 (2012)
19. Rosso, M.M., Aloisio, A., Cirrincione, G., Marano, G.C.: Subspace features and statistical indicators for neural network-based damage detection. In: Structures. vol. 56, p. 104792. Elsevier (2023)
20. Saharan, N., Kumar, P., Pal, J.: Convolutional neural network–based structural health monitoring framework for wind turbine blade. Journal of Vibration and Control p. 10775463231213423 (2023)
21. Simoncelli, M., Aloisio, A., Zucca, M., Venturi, G., Alaggio, R.: Intensity and location of corrosion on the reliability of a steel bridge. Journal of Constructional Steel Research **206**, 107937 (2023)
22. Talebi, A., Potenza, F., Gattulli, V.: Interoperability between bim and fem for vibration-based model updating of a pedestrian bridge. In: Structures. vol. 53, pp. 1092–1107. Elsevier (2023)
23. Van Overschee, P., De Moor, B.: Subspace identification for linear systems: Theory—Implementation—Applications. Springer Science & Business Media (2012)
24. Zhang, L., Wang, T., Tamura, Y.: A frequency–spatial domain decomposition (fsdd) method for operational modal analysis. Mechanical systems and signal processing **24**(5), 1227–1239 (2010)
25. Zhu, M., McKenna, F., Scott, M.H.: Openseespy: Python library for the opensees finite element framework. SoftwareX **7**, 6–11 (2018)