Easy automated OMA with open source software

iomac
International Operational Modal Analysis Conference

11th International Operational Modal Analysis Conference

20 - 23 May 2025 | Rennes, France

# Easy automated OMA with open source software

*Dag Pasquale Pasca* [1], *Angelo Aloisio* [2], *Marco Civera* [3] *and Matteo Verzeroli* [3]

[1] Norwegian Institute of Wood Technology, dpa@treteknisk.no
[2] University of L'Aquila, angelo.aloisio1@univaq.it
[3] Politecnico di Torino, marco.civera@polito.it
[3] Politecnico di Torino, s315138@studenti.polito.it

**ABSTRACT**

In this paper, we introduce a newly developed clustering tool integrated into `pyOMA2`, an open-source Python module designed for conducting Operational Modal Analysis (OMA). `pyOMA2` provides easy access to the most popular algorithms developed over the past two decades, such as Stochastic Subspace Identification (SSI) and Frequency Domain Decomposition (FDD). It supports the analysis of both single and multi-setup data measurements and offers interactive plots and geometry definitions to facilitate the visualization of mode shapes after obtaining modal results. Since version 1.0.1, the software also includes the ability to estimate uncertainty bounds of modal properties for SSI algorithms.

One of the main advantages of `pyOMA2` is its modularity, which facilitates the development of additional functionalities. A prime example is the newly developed clustering handler, a specialized class that enables users to define and execute various clustering strategies to automate the selection of modal parameters from SSI results. This tool enables users to implement and compare a large number of the most popular algorithms introduced over the last 15 years, all within the same analysis framework. Furthermore, users have the flexibility to mix specific strategies from different algorithms to customise the clustering process accordingly to their needs. Additionally, the integration with the popular machine learning module scikit-learn has expanded the range of available clustering algorithms, providing users with even more options for their analyses. All these capabilities are illustrated in the paper through applications to both a numerical example and real-world datasets.

*Keywords: Automatic OMA, Clustering, Open-source software*

## 1. INTRODUCTION

Operational Modal Analysis (OMA) has become the de facto standard for estimating the modal properties of structures in Structural Health Monitoring (SHM) applications for civil engineering. However,

traditional OMA requires an operator to interpret a stabilisation diagram (or a spectral density plot) to identify the structural modes in the recorded data. To address this limitation, many researchers over the past 15 years have developed algorithms aimed at automating this process [1–10]. These efforts seek to eliminate the need for user interaction, which would otherwise make continuous monitoring applications impractical.

As highlighted by several authors [2, 8, 11, 12], one can identify three main steps to automate the extraction of modal parameters from the stabilisation charts typical of SSI: Removing spurious modes to retain physically meaningful ones, grouping modes with similar dynamic characteristics, and performing further cleaning to select representative modes for output.

In this contribution, the authors introduce a newly developed clustering tool for `pyOMA2`, a Python module specifically designed for OMA applications. `pyOMA2` incorporates several significant enhancements and novel features compared to its predecessor, `pyOMA` [13], improving its usability and expanding its functionalities. The source code is available at the following GitHub repository: `https://github.com/dagghe/PyOMA2`.

The remainder of the paper is organized as follows: in Section 2, a brief overview of the various algorithms proposed in recent years is provided. section 3 presents the implementation of the clustering tools in `pyOMA2`; section 4 demonstrates the tool using well-known case studies and a numerical simulation.

## 2. AUTOMATIC OMA

A key output of parametric modal identification methods, such as SSI, is the so-called stabilisation diagram, which plots the identified modes over various model orders against their corresponding frequencies. Since the true model order of a system is not known a priori, it is common practice to over-specify this parameter, considering the underlying empirical observation that physical modes, being true properties of the structure, will appear with nearly identical modal characteristics across different model orders [14]. In contrast, spurious mathematical modes will be identified inconsistently. In a stabilisation diagram physical modes manifest as vertical lines, indicating stable modal properties regardless of system order, while spurious modes scatter throughout the diagram.

As already pointed out most of the automatic operational modal analysis (AOMA) algorithms follow a similar three-step approach to identify the structural modes from the outcome of the modal analysis. First, they separate physical modes from mathematical (spurious) ones. Second, they cluster the stable modes, grouping data points with similar characteristics. Finally, they refine the results by removing outliers and select the physical modes to output. This sequence of steps effectively automates what an analyst would traditionally do by manually inspecting the stabilisation diagram.

### 2.1. Step 1

The first step is to clean the stabilisation diagram. For this purpose, several Mode Validation Criteria (MVCs) have been proposed[2, 3, 15]. Reynders [2] provides an exhaustive list of MVCs that can be applied to this task. Generally, these criteria are divided into Hard Validation Criteria (HVCs) and Soft Validation Criteria (SVCs), and they serve to distinguish physical modes from spurious ones.

HVCs typically produce binary outcomes. Common HVC checks include verifying that the damping ratio is greater than 0 but less than 0.2 (for most civil engineering applications) and ensuring that modes occur in complex conjugate pairs. Any poles that do not meet these criteria are considered mathematical (non-physical) and are removed from the stabilisation diagram.

In contrast, SVCs yield values within a specific range and involve verifying that a pole exhibits relatively low variation compared to its closest neighbour at a lower modal order. Notably, SVCs can be transformed into HVCs by imposing specific thresholds. This approach leverages the observation that physical poles tend to exhibit minimal variation with increasing modal ordera principle that has long

been used in manual analyses to highlight low variability in the results.

Commonly used SVCs include measures such as the relative difference between a pole and its nearest neighbour at a lower order (in terms of eigenvalues, frequencies, damping ratios, or mode shapes, assessed using the Modal Assurance Criterion, MAC), the complexity of a mode shape (evaluated through Modal Phase Collinearity or Mean Phase Deviation), and the contribution of a mode to the overall response (assessed via the Modal Transfer Norm). Most SVCs are computed as relative difference values, calculated according to the following equation:

$$d(a_j; a_i) = \frac{|a_j - a_i|}{\max(|a_j|, |a_i|)} \tag{1}$$

where $a_i$ could represent an eigenvalue $\lambda_i$, a frequency $f_i$, a damping $\xi_i$, etc., of the current mode, while $a_j$ is the corresponding value from the nearest neighbour at a lower modal order.

As mentioned earlier, during a manual analysis, the user selects the desired MVCs and manually sets static thresholds for each chosen criterion. To fully automate the cleaning process, partitioning methods have been proposed. These methods classify the modes into two clusters: one containing certainly spurious modes and the other containing potentially physical modes. Several partitioning algorithms have been successfully employed over the years, including k-means (KM) [2, 4, 7], Fuzzy C-means (FCM) [15, 16], and Gaussian Mixture Models (GMM) [10]. Notably, some authors suggest transforming the feature vector before passing it to the partitioning algorithm [7, 15], e.g. with a Box-Cox transform. The k-means algorithm partitions the data into $k$ clusters by minimizing the sum of squared distances between data points and their respective cluster centroids. In contrast, Fuzzy C-means allows each data point to belong to multiple clusters with varying degrees of membership. Gaussian Mixture Models assume that the data is generated from a mixture of several Gaussian distributions and use probabilistic assignments of data points to clusters, which can capture more complex cluster shapes.

### 2.2. Step 2

After cleaning the stabilisation diagram and removing most of the spurious modes, the next step is to automatically group the remaining poles to extract the physical modes. This grouping is achieved through unsupervised clustering algorithms that collect similar modes based on a predefined similarity (or distance) measure between the poles. Over the years, various clustering techniques have been proposed, with Hierarchical Clustering being the most popular choices [1–3, 7, 15]. In addition, density-based algorithms such as DBSCAN [17, 18], HDBSCAN[9], and OPTICS [8] have also been widely adopted.

Hierarchical Clustering organizes the data into a nested tree-like structure (a dendrogram) by recursively merging or splitting clusters according to their similarity, with different linkage options available (e.g., single, complete, or average linkage). In contrast, DBSCAN (Density-Based Spatial Clustering of Applications with Noise) groups closely packed points together while identifying isolated points in low-density regions as outliers. HDBSCAN (Hierarchical DBSCAN) builds upon DBSCAN by constructing a hierarchy of clusters and then extracting the most stable ones. Similarly, OPTICS (Ordering Points To Identify the Clustering Structure) creates an augmented ordering of the dataset that reveals its density-based clustering structure, enabling the identification of clusters with varying densities.

### 2.3. Step 3

The third and final step is dedicated to the post-processing of the results. This involves removing outlier poles from the clusters, eliminating clusters that are not representative of structural modes, and selecting the detected modal properties to output to the user. Outlier removal has been performed in several ways, such as the Interquartile Range (IQR) method [1, 15], which uses percentiles to define a range outside of which points are flagged as outliers, the Modified Thompson Tau (MTT) method [7], a statistical test for small datasets using t-distribution critical values, and the adjusted boxplot method [19], which modifies

the upper and lower boundaries based on the skewness of the datas probability distribution. Other authors also emphasize the importance of including only one pole per modal order (per cluster).

Another common approach is to retain only clusters that contain a relatively large number of poles [2, 3, 6, 7, 10, 15]. This is a good indicator that the cluster is representative of a structural mode, as opposed to a spurious cluster that usually cannot form large groups. This can be achieved by imposing a minimum number of poles (often expressed as a percentage of the maximum modal order or as a percentage of the number of poles in the largest cluster [3, 6, 7, 15]) or by applying an additional clustering process using, for example, KM and GMM clustering algorithms [2, 10].

Finally, the detected modal properties for each mode can be defined as the average of all poles within a cluster, or by selecting the pole whose frequency or damping are closest to the median values.

## 3. IMPLEMENTATION IN PYOMA2

`pyOMA2` is a module that fully leverages Pythons object-oriented capabilities and embraces the open source philosophy. Distributed under the permissive MIT license, the module benefits from the active Python community and its powerful tools, such as the popular machine learning library scikit-learn [20], which provides access to several clustering algorithms. The modules modularity has greatly facilitated the creation of a robust clustering tool. At the time of writing, the clustering handler tool is available in beta (v1.2.0b2).

The module is organized into three primary levels:

1. **Setup Classes:** At this level, users instantiate classes by providing either a single data array and its sampling frequency (for a single-setup scenario) or a list of data arrays and their respective sampling frequencies (for multi-setup scenarios). Note that clustering is applicable only to the `SingleSetup` class.
2. **Algorithm Classes:** This level comprises the various algorithm classes. Users instantiate the desired algorithms and then add them to the setup class. For instance, the `AutoSSI` class, which implements the automatic SSI algorithm, is included in the available algorithms.
3. **Support Classes:** These auxiliary classes support the functionality of the first two levels. They include various specialized classes including `Step1`, `Step2`, `Step3`, and `Clustering`, which are dedicated to clustering operations.

In addition to these levels there is a further, comprised of the set of functions internally called by the class methods.

This update introduces a new algorithm class, `AutoSSI`, which as the name suggest provides an automatic implementation of the SSI algorithm. The following example demonstrates how to use the module. The process begins by importing the necessary modules and instantiating the setup class with the data and its sampling frequency. Following, the `AutoSSI` class is initialized passing to it the run parameters, such as number of block rows, maximum order etc., these are also defined within their own specific class (i.e. `AutoSSIRunParams`)

```
from pyoma2.setup import SingleSetup
from pyoma2.algorithms import AutoSSI

clust_test = SingleSetup(data, fs=100) # create single setup

# define AutoSSI run parameters
run_param = AutoSSIRunParams(ordmax=100, step=2, br=50, method="cov", calc_unc=False
                                                )
# Create autoSSI instance
autossi = AutoSSI(name="autossi", run_params=run_param)
```

The `Clustering` class can be defined either using the quick argument or the steps argument. The quick argument allows for a quick definition of a clustering algorithm from a list of predefined ones, while the steps require the user to define each of the three steps that make up a custom-taylored clustering algorithm. Once the clustering algorithms have been defined they are added to the `AutoSSI` class instance, thereafter the `AutoSSI` class instance is added to the `AutoSSI`SingleSetup class instance. Once this is done the analysis can be executed.

```
# define STEPS
# STEP1
step1 = Step1(sc=True, pre_cluster=True, pre_clus_typ="GMM")

# STEP2
step2 = Step2(algo="hierarc", linkage="average")

# STEP3
step3 = Step3(
    post_proc=["merge_similar", "damp_IQR", "fn_IQR", "1xorder", "min_size_gmm", "
                                                    MTT"]
)

# Define Clustering algorithms
clus1 = Clustering(name="test-hierarc_avg", steps=[step1, step2, step3])
clus2 = Clustering(name="Reynders", quick="Reynders")

# Add clustering algorithms to AutoSSI class instance
autossi.add_clustering(clus1, clus2)

# add AutoSSI instance to SingleSetup instance
clust_test.add_algorithms(autossi)

# Run algorithm
clust_test.run_all()

# Run clustering either one by one or altogether
clust_test["autossi"].run_all_clustering()
```

All the possible options available so far for `Step1`, `Step2`, and `Step3` are summarized in Table 1.

## 4. VALIDATION

We present a validation of the clustering handler through two case studies: one numerical example and one from a widely recognized benchmark.

First Case Study (Numerical): A three-story, L-shaped building with a 3 by 2 bay configuration is modelled using the Python implementation of the OpenSees finite element software [21]. The structural model and associated dataset are included in the `pyOMA2` module, ensuring reproducibility and accessibility.

Second Case Study (Benchmark): We analyse the first dataset from the Heritage Court Tower (HCT) in Vancouver, a well-established benchmark in the OMA community. The HCT case is featured in the popular textbook by Brinker and Ventura [22], with its dataset available in the accompanying MATLAB toolbox.

For both cases the same clustering algorithms are applied, these are summarised in tab. 2.

### 4.1. Numerical, L-shape building

The files required for this example are available at the following link: `https://github.com/dagghe/pyOMA-test-data/tree/main/test_data/3SL`. The building was modeled assuming rigid diaphragm behavior for the floors, yielding nine modes in total. During the simulation,

**Table 1:** Available options for the definition of the clustering steps.

| Step no. | attribute | type/options | description |
|---|---|---|---|
| Step 1 | hc | (bool, 'after') | Whether to apply HC immediately, not at all , or after pre-clustering |
| | hc_dict | (xi_max, mpc_lim, mpd_lim, CoV_max) | Dictionary of hard validation criteria |
| | sc | (bool, 'after') | Whether to apply SC immediately, not at all, or after pre-clustering |
| | sc_dict | (err_fn, err_xi, err_phi) | Dictionary of soft validation criteria |
| | pre_clus | bool | Whether to pre-cluster data before analysis |
| | pre_clus_typ | ('GMM', 'kmeans') | Type of pre-clustering algorithm. |
| | pre_clus_dist | ('dfn', 'dxi', 'dlambda', 'dMAC', 'dMPC', 'dMPD', 'MPC', 'MPD') | Distance metrics used for pre-clustering. |
| | transform | ('box-cox') | Data transformation method. |
| Step 2 | distance | ('dfn', 'dxi', 'dlambda', 'dMAC', 'dMPC', 'dMPD') | Distance metrics for clustering. |
| | weights | ('tot_one', None) or list; bool | Weighting scheme for distance metrics |
| | sqrtsqr | bool | Whether to apply square-root to the sum of squares. |
| | algo | ('hdbscan', 'hierarc', 'optics', 'spectral', 'affinity') | Clustering algorithm to use |
| | dc | (float, 'auto', 'mu+2sig', '95weib') | Distance threshold for hierarchical clustering |
| | linkage | ('average', 'complete', 'single') | Linkage criterion for hierarchical clustering. |
| | min_size | (int, 'auto') | Minimum cluster size. |
| | n_cluster | (int, 'auto') | Number of clusters. |
| Step 3 | post_proc | ('merge_similar', 'damp_IQR', 'fn_IQR', 'fn_med', '1xorder', 'min_size', 'min_size_pctg', 'min_size_kmeans', 'min_size_gmm', 'MTT','Adj_boxplot') | list of post-processing steps to apply to clustering results |
| | merge_dist | (float, 'auto', 'deder') | Threshold for merging similar clusters. |
| | min_pct | float | Minimum cluster size as a percentage of the largest cluster. |
| | select | ('avg', 'fn_med_close', 'xi_med_close', 'medoid') | Method for selecting final clustering results. |
| | freq_lim | tuple of float | Frequency range limits. |

**Table 2:** Summary of the options applied to the step used in the analysis of the two case studies

| Algorithm | Step 1 | Step 2 | Step 3 |
|---|---|---|---|
| Dederichs | HC;<br>pre_clus_typ="GMM" | algo="hierarc";<br>n_clusters="auto" | post_proc=("merge_similar", "1xorder",<br>"min_size_gmm"); select="avg" |
| Kvaale | err_fn=0.04,<br>err_xi=0.2, err_phi=0.1 | algo="hdbscan";<br>min_size=20 | post_proc=["1xorder"]; select="avg" |
| Neu | pre_clus_typ="kmeans";<br>transform="box-cox"; HC | algo="hierarc"; dc="95weib";<br>linkage="average" | post_proc=("min_size_pctg", "MTT");<br>min_pctg=0.5; select="avg" |
| Reynders | hc=after;<br>pre_clus_typ="kmeans" | algo="hierarc"; dc="mu+2sig";<br>linkage="average" | post_proc=["min_size_kmeans"];<br>select="xi_med_close" |
| test1 | HC; SC<br>pre_clus_typ="GMM" | algo="affinity" | post_proc=["merge_similar", "damp_IQR",<br>"fn_IQR", "1xorder", "min_size_pctg", "Adj_boxplot"] |
| test2 | HC; SC | algo="hierarc";<br>linkage="average" | post_proc=["merge_similar", "damp_IQR",<br>"fn_IQR", "1xorder", "min_size_gmm", "MTT"] |
| test3 | HC; SC | algo="hierarc"; linkage="single";<br>dc=None; n_clusters="auto" | post_proc=["merge_similar", "damp_IQR",<br>"fn_IQR", "1xorder", "min_size_kmeans", "MTT"] |
| test4 | HC; SC<br>pre_clus_typ="kmeans" | algo="spectral" | post_proc=["merge_similar", "damp_IQR",<br>fn_IQR, "1xorder", "min_size_kmeans", "MTT"] |

accelerations at 12 nodes (four nodes per floor corresponding to the centers of gravity of the bays) were extracted. Three simulations were run, exporting results from 10 nodes, two of which correspond to the reference sensors. However, in this analysis only the first dataset is considered. The true frequencies and the results of the clustering algorithms are shown in Table 3.

**Table 3:** Comparison of the clustering results for the numerical case. Values between parenthesis indicate duplicate modes, and dashes denote modes that were not identified.

| Mode No | True values | Dederichs | Kvaale | Neu | Reynders | test1 | test2 | test3 | test4 |
|---|---|---|---|---|---|---|---|---|---|
| 1 | 2.632 | 2.626 | 2.626 | 2.626 | 2.627 | 2.625 | 2.625 | 2.625 | 2.625 |
| 2 | 2.692 | 2.698 | 2.700 | - | 2.703 | 2.700 | 2.700 | 2.700 | 2.700 |
| 3 | 3.430 | 3.430 | 3.430 | 3.430 | 3.426 | 3.431 | 3.431 | 3.431 | 3.430 (3.753) |
| 4 | 8.297 | 8.293 | 8.293 | 8.293 | 8.291 | 8.290 | 8.290 | 8.290 | 8.290 |
| 5 | 8.429 | 8.413 | 8.413 | 8.414 | 8.420 | 8.415 | 8.415 | 8.415 | 8.415 |
| 6 | 10.627 | 10.586 | 10.588 | 10.571 | 10.415 | 10.589 | 10.589 | 10.589 | 10.589 |
| 7 | 14.005 | 13.953 | 13.954 | 13.954 | - | - | 13.934 | 13.934 | 13.933 |
| 8 | 14.093 | 14.059 | 14.046 | 14.046 | 14.049 | 14.049 | 14.032 | 14.033 | 14.033 |
| 9 | 17.574 | 17.421 | 17.431 | 17.380 | 17.377 | 17.552 | 17.376 | 17.376 | 17.379 |

### 4.2. Experimental, HTC building

The HCT s a relatively regular 15 story reinforced concrete shear core building, a detailed description of the building and its ambient vibration test can be found in APPENDIX B of reference [22]. The HCT vibration measurements are a well-known set of data in OMA literature, it comprise a total of four datasets, however only the first set was analysed. The results of manual OMA from reference [22] and the results of the clustering algorithms are shown in Table 4.

## 5. CONCLUSIONS

This paper introduces a new clustering handler for the open-source Python package `pyOMA2`, aimed at fully automating the identification of structural modes via Operational Modal Analysis. The tool integrates smoothly with existing OMA algorithms and can be customized to suit different analysis requirements. By leveraging Pythons flexibility and machine learning libraries (e.g., scikit-learn), `pyOMA2` now

**Table 4:** Comparison of the clustering results for the HCT building

| Mode No | FDD [22] | SSI [22] | Dederichs | Kvaale | Neu | Reynders | test1 | test2 | test3 | test4 |
|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 1.230 | 1.228 | 1.225 | 1.226 | 1.225 | 1.225 | 1.225 | 1.225 | 1.225 | 1.225 |
| 2 | 1.299 | 1.287 | 1.287 | 1.287 | 1.286 | 1.287 | 1.286 | 1.286 | 1.286 | 1.286 |
| 3 | 1.455 | 1.453 | 1.446 | 1.446 | 1.446 | 1.448 | 1.446 | 1.446 | 1.446 | 1.446 |
| 4 | 3.848 | 3.856 | 3.855 | 3.855 | 3.855 | 3.854 | 3.855 | 3.855 | 3.855 | 3.855 |
| 5 | 4.277 | 4.260 | 4.241 | 4.241 | 4.243 | 4.244 | 4.243 | 4.243 | 4.243 | 4.243 |
| 6 | 5.352 | 5.354 | 5.354 | 5.353 | 5.352 | 5.354 | 5.353 | 5.353 | 5.353 | 5.353 |
| 7 | 6.436 | 6.401 | 6.398 | 6.398 | 6.397 | 6.402 | 6.402 | 6.402 | 6.402 | 6.402 |

offers users a powerful and user-friendly framework for continuous-monitoring applications in structural health monitoring.

## REFERENCES


[1] Filipe Magalhães, Alvaro Cunha, and Elsa Caetano. Online automatic identification of the modal parameters of a long span arch bridge. *Mechanical Systems and Signal Processing*, 23(2):316–329, 2009.

[2] Edwin Reynders, Jeroen Houbrechts, and Guido De Roeck. Fully automated (operational) modal analysis. *Mechanical systems and signal processing*, 29:228–250, 2012.

[3] Guowen Zhang, Jinghua Ma, Zhuo Chen, and Ruirong Wang. Automated eigensystem realisation algorithm for operational modal analysis. *Journal of Sound and Vibration*, 333(15):3550–3563, 2014.

[4] Miao Sun, Mehrisadat Makki Alamdari, and Hamed Kalhori. Automated operational modal analysis of a cable-stayed bridge. *Journal of Bridge Engineering*, 22(12):05017012, 2017.

[5] Alessandro Cabboi, Filipe Magalhães, Carmelo Gentile, and Álvaro Cunha. Automated modal identification and tracking: Application to an iron arch bridge. *Structural Control and Health Monitoring*, 24(1):e1854, 2017.

[6] Rhara Cardoso, Alexandre Cury, and Flávio Barbosa. A robust methodology for modal parameters estimation applied to shm. *Mechanical Systems and Signal Processing*, 95:24–41, 2017.

[7] Eugen Neu, Frank Janser, Akbar A Khatibi, and Adrian C Orifici. Fully automated operational modal analysis using multi-stage clustering. *Mechanical Systems and Signal Processing*, 84:308–323, 2017.

[8] Ruben L Boroschek and Joaquin A Bilbao. Interpretation of stabilization diagrams using density-based clustering algorithm. *Engineering Structures*, 178:245–257, 2019.

[9] KA Kvåle and O Øiseth. Automated operational modal analysis of an end-supported pontoon bridge using covariance-driven stochastic subspace identification and a density-based hierarchical clustering algorithm. In *Bridge maintenance, safety, management, life-cycle sustainability and innovations*, pages 3041–3048. CRC Press, 2021.

[10] AC Dederichs and O Øiseth. A new fully automated operational modal analysis algorithm intended for large civil structures. In *Journal of Physics: Conference Series*, volume 2647, page 192009. IOP Publishing, 2024.

[11] Giacomo Zini, Michele Betti, and Gianni Bartoli. A quality-based automated procedure for operational modal analysis. *Mechanical Systems and Signal Processing*, 164:108173, 2022.

[12] Anno Christian Dederichs and Ole Øiseth. Experimental comparison of automatic operational modal analysis algorithms for application to long-span road bridges. *Mechanical Systems and Signal Processing*, 199:110485, 2023.

[13] Dag Pasquale Pasca, Angelo Aloisio, Marco Martino Rosso, and Stefanos Sotiropoulos. Pyoma and pyoma_gui: A python module and software for operational modal analysis. *SoftwareX*, 20: 101216, 2022.

[14] P. Van Overschee and B. De Moor. *Subspace Identification for Linear Systems: Theory, Implementation, Applications*. Kluwer, Dordrecht, The Netherlands, 1996.

[15] Xiao-Mei Yang, Ting-Hua Yi, Chun-Xu Qu, Hong-Nan Li, and Hua Liu. Automated eigensystem realization algorithm for operational modal identification of bridge structures. *Journal of Aerospace Engineering*, 32(2):04018148, 2019.

[16] P-É Charbonnel. Fuzzy-driven strategy for fully automated modal analysis: Application to the smart2013 shaking-table test campaign. *Mechanical Systems and Signal Processing*, 152:107388, 2021.

[17] Marco Civera, Luigi Sibille, Luca Zanotti Fragonara, and Rosario Ceravolo. A dbscan-based automated operational modal analysis algorithm for bridge monitoring. *Measurement*, 208:112451, 2023.

[18] EM Tronci, Maurizio De Angelis, R Betti, and V Altomare. Multi-stage semi-automated methodology for modal parameters estimation adopting parametric system identification algorithms. *Mechanical Systems and Signal Processing*, 165:108317, 2022.

[19] Min He, Peng Liang, Jiuxian Liu, and Zhiqiang Liang. Review and comparison of methods and benchmarks for automatic modal identification based on stabilization diagram. *Journal of Traffic and Transportation Engineering (English Edition)*, 2024.

[20] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, et al. Scikit-learn: Machine learning in python. *the Journal of machine Learning research*, 12:2825–2830, 2011.

[21] Minjie Zhu, Frank McKenna, and Michael H Scott. Openseespy: Python library for the opensees finite element framework. *SoftwareX*, 7:6–11, 2018.

[22] Rune Brincker and Carlos Ventura. *Introduction to operational modal analysis*. John Wiley & Sons, 2015.