PyOMA and PyOMA_GUI: A Python module and software for Operational Modal Analysis

SoftwareX 20 (2022) 101216

Contents lists available at ScienceDirect

SoftwareX

journal homepage: www.elsevier.com/locate/softx

Original software publication

PyOMA and PyOMA_GUI: A Python module and software for Operational Modal Analysis

Dag Pasquale Pasca [a], Angelo Aloisio [b], Marco Martino Rosso [c,*], Stefanos Sotiropoulos [c]

[a] Norsk Treteknisk Institutt, Børrestuveien 3, 0373, Oslo, Norway
[b] Department of Civil, Construction-Architectural and Environmental Engineering, Università degli Studi dell'Aquila, L'Aquila, Italy
[c] Department of Structural, Geotechnical and Building Engineering Politecnico di Torino, Corso Duca degli Abruzzi, 24, 10129, Torino, Italy

ARTICLE INFO

Article history:
Received 3 June 2022
Received in revised form 26 August 2022
Accepted 24 September 2022

Keywords:
Operational modal analysis
Ambient vibration test
Structural dynamics
Python
Output-only
Modal testing
Graphical User Interface

ABSTRACT

In structural health monitoring (SHM) paradigm, operational modal analysis (OMA) comprises several techniques and algorithms for estimating the dynamic characteristics of a structure in operational conditions from its vibration response. The OMA method has been spreading in the last years due to multiple advantages compared to input–output identification methods. In the current work the authors present the implementation of a Python module named PyOMA and its Graphical User Interface (GUI) PyOMA_GUI. This software provides a user-friendly framework for the first time in the Python environment for estimating the experimental modal parameters (natural frequencies, mode shapes, damping ratios) of a structure from output-only vibration measurements in operational conditions.

Code metadata

Current code version	v1.4
Permanent link to code/repository used for this code version	https://github.com/ElsevierSoftwareX/SOFTX-D-22-00147
Code Ocean compute capsule	None
Legal Code License	GPL-3.0 license
Code versioning system used	git
Software code languages, tools, and services used	Python, PyQt
Compilation requirements, operating environments & dependencies	numpy, pandas, scipy, matplotlib, seaborn, mplcursors
If available Link to developer documentation/manual	https://github.com/dagghe/PyOMA/wiki
Support email for questions	supportPyOMA@polito.it

Software metadata

Current software version	1.0
Permanent link to executables of this version	For example: https://github.com/dagghe/PyOMA/blob/master/PyOMA_GUI/PyOMA_GUI.exe
Legal Software License	GPL-3.0 license
Computing platforms/Operating Systems	Microsoft Windows
Installation requirements & dependencies	None, one single executable file
If available, link to user manual - if formally published include a reference to the publication in the reference list	https://github.com/dagghe/PyOMA/blob/master/PyOMA_GUI/PyOMA_GUI_user_manual_v1.0.pdf
Support email for questions	supportPyOMA@polito.it

1. Motivation and significance

The practice of operational modal analysis (OMA) on civil structures and infrastructures has been growing significantly in the last decades [1]. OMA allows estimating the modal properties (natural frequencies, mode shapes and damping ratios) from

* Corresponding author.
E-mail address: marco.rosso@polito.it (Marco Martino Rosso).

https://doi.org/10.1016/j.softx.2022.101216

output-only ambient vibration tests while the structure is in its operating conditions. Measuring the input ambient excitations for a civil engineering structure is nearly impossible. Neither can it adequately excite them using conventional devices, such as impact hammers and shakers. On the other hand, output vibration data acquired through accelerometers are relatively easy to obtain. In OMA, the deterministic knowledge of the input excitation is replaced by the assumption that the input is a realization of a stochastic process. Accordingly, OMA and output-only dynamic identification are considered synonyms. There is a large variety of algorithms for OMA. Among them, the Stochastic Subspace Identification (SSI) and the Frequency Domain Decomposition (FDD) have proved to be effective and reliable algorithms for output-only dynamic identification [1]. The SSI is a time-domain method that estimates the stochastic state-space model from stationary output-only data [2]. Furthermore, to deal with non-stationary output-only data [3], e.g. free decay vibrational responses, a variant of the covariance-based SSI algorithm, which is equivalent to the Natural Excitation Technique (NExT) with Eigensystem Realization Algorithm (ERA) method, named NExT-ERA [4,5] was developed. The FDD is a frequency domain method that estimates the modal parameters using the input/output data of an n degree of freedom (DOF) system stochastic process [6]. So in conclusion, PyOMA is an open-source Python module that allows the estimation of the modal parameters of a structure using six acknowledged techniques derived from SSI and FDD:

1. Frequency Domain Decomposition [6];
2. Enhanced Frequency Domain Decomposition [7];
3. Frequency Spatial Domain Decomposition [8];
4. Covariance driven Stochastic Subspace Identification [2];
5. Data driven Stochastic Subspace Identification [9];

The frequency domain methods (No 1–3) are based on the computation of the power spectrum density (PSD) matrix [10]. Conversely, the time domain methods (No 4–6) are based on correlation function or the analysis of response time histories. The FDD is one of the most popular OMA techniques [6]. It is an extension of the pick-peaking method based on the singular value decomposition (SVD) of the PSD matrix. The major limitation of the FDD is the lack of a reliable method for the damping ratio estimation. The EFDD [7] represents an extension of the FDD to overcome this issue and estimate the damping ratios. This method uses inverse discrete Fourier transform (IDFT) to transform the PSD into time domain impulse responses at the resonance peaks. Damping is obtained from the logarithmic decrement of the corresponding normalized auto-correlation function. The EFDD can be used to accurately estimate natural frequencies and closely spaced modes. The FSDD represents a further improvement of the FDD based on manipulating the PSD matrix with the modes estimates from preliminary FDD. Stochastic subspace identification (SSI) is the most typical time domain OMA method. It was developed in 1991 by Van Overschee, and De Moor [11]. SSI allows the identification of the state space model of dynamic systems under stochastic excitation [2]. There are two algorithms of SSI: data-driven SSI (SSI-DAT) and covariance-driven SSI (SSI-COV). In SSI-COV, the block Hankel matrix is obtained from data correlation. Conversely, in the SSI-DAT, the QR decomposition of the data Hankel matrix is used to project the row space of future outputs into the row space of past outcomes [12]. Both SSI-COV and SSI-DAT are capable of estimating system modes and forced oscillations. Nonetheless, SSI-COV is more computationally efficient compared to SSI-DAT. In addition, SSI methods allow a high parameter estimation accuracy and high computational efficiency compared to other OMA methods [13]. Due to these advantages, SSI has become a standard in OMA. There are two main limitations of OMA methods:

- Unscaled mode shapes: The lack of knowledge on the exciting force does not allow the mode shapes normalization.
- Gaussian white noise: It is assumed that all modes are equally excited by Gaussian white noise. Nonetheless, this is never true in practice, and harmonic components, non-stationary or narrow-band excitation, can bias the outcomes of OMA. It must be remarked that when dealing with non-stationary excitation, the above methods cannot be rigorously applicable (seismic response, vehicle–bridge interaction problems, e.g.)

2. Software description

PyOMA module is an open-source Python module that implements a complete output-only OMA framework for researchers, engineers and practitioners. Through the implemented functions, it is possible to estimate the modal parameters of a civil structure using dynamic identification techniques derived from SSI and FDD [1], as illustrated in Fig. 1. In addition, the authors provided a graphical user interface software version of the current module, called PyOMA_GUI. The PyOMA_GUI aims to improve the appeal of the existing open-source Python OMA module, which has already been used in several applications. Not secondarily, the graphical user interface does not require any Python expertise or Python coding knowledge prerequisite.

2.1. Software functionalities

PyOMA_GUI is a graphical user interface software developed in PyQt5, which implements in a single integrated tool the operational modal analysis of civil structures with output-only measurement data. A general overview of the software functionalities are depicted in Fig. 2. This software employs the functionalities mentioned above offered by the PyOMA python module. Therefore, PyOMA_GUI provides an exceptionally user-friendly interface to improve the accessibility of the PyOMA module, ensuring widespread usage for scientists, researchers, and applied civil and structural engineers. The main features PyOMA_GUI provides are listed below:

- Importing data tab;
- Definition of the geometry of the structure and the monitoring system (channels and degrees of freedom, DOFs);
- Pre-processing of the acquired signals with detrending and decimation options;
- Dynamic identification algorithms with visualization of the results (graphs, modal shapes);
- Post-processing tabs and output export functionalities;

3. Illustrative example

In the present illustrative example, a 5 Degrees of Freedom (DOFs) shear type frame has been considered with lumped mass m = 25.91 Ns^2/mm at each floor, and same storey stiffness k = 10000 N/mm to each floor level. The function `oma.Exdata()` generates the data needed related to the current example. The analytical solution to the eigenvalue problem provides the natural frequencies of the system [14]:

$$f_n = \begin{bmatrix} f_1 \\ f_2 \\ f_3 \\ f_4 \\ f_5 \end{bmatrix} = \begin{bmatrix} 0.88995 \\ 2.59776 \\ 4.09511 \\ 5.2607 \\ 6.0001 \end{bmatrix} \text{ [Hz]} \quad (1)$$

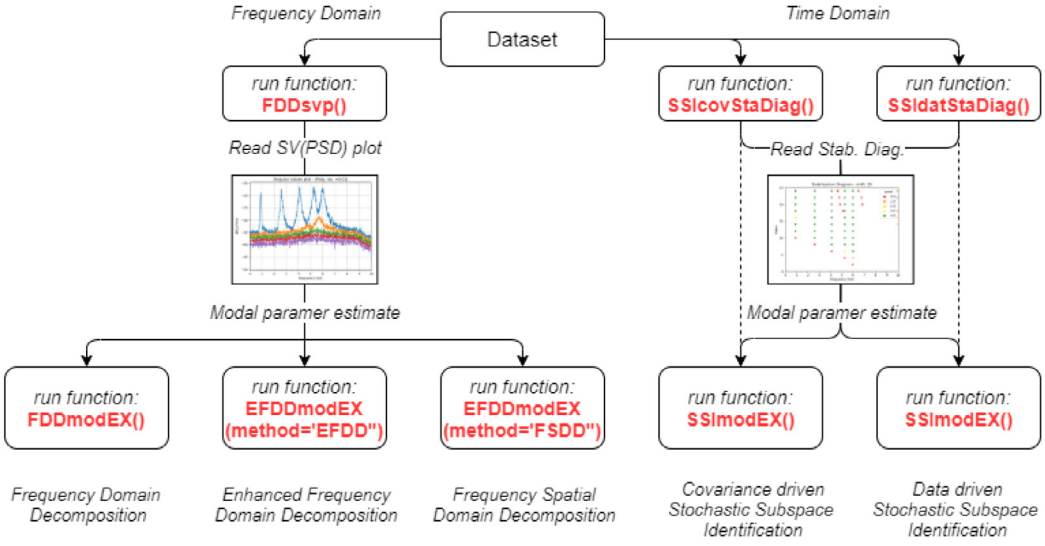


Fig. 1. "PyOMA" module flowchart.

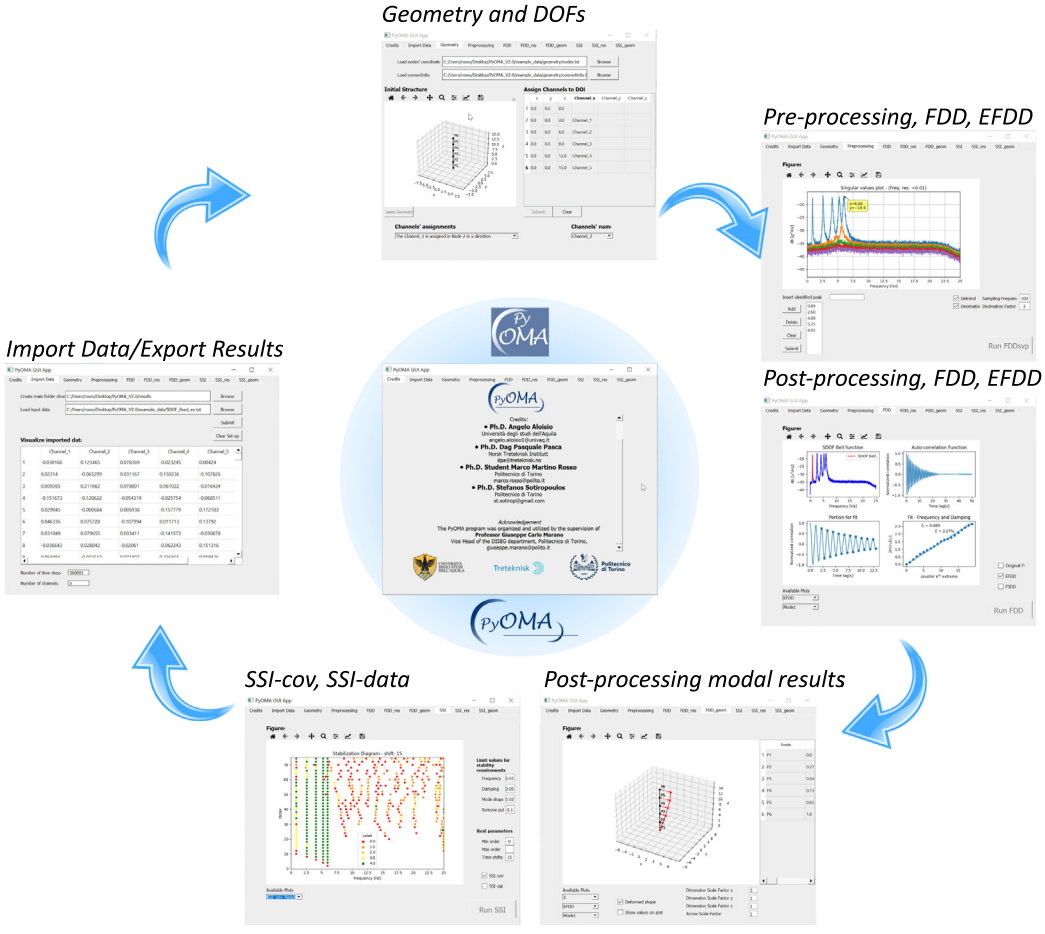


Fig. 2. "PyOMA_GUI" graphical user interface software general overview.

and the normalized mode shapes, retrieved by scaling each mode shape to set its largest component in absolute value to unity, are:

$$\Phi = [\phi_1\ \phi_2\ \phi_3\ \phi_4\ \phi_5] = \begin{bmatrix} 0.28463 & -0.763521 & 1 & 0.918986 & -0.5462 \\ 0.5462 & -1 & 0.28463 & -0.763521 & 0.918986 \\ 0.763521 & -0.5462 & -0.918986 & -0.28463 & -1 \\ 0.918986 & 0.28463 & -0.5462 & 10.763521 & \\ 1 & 0.918986 & 0.763521 & -0.5462 & -0.28463 \end{bmatrix} \quad (2)$$

The damping matrix is calculated assuming a constant damping ratio of 2% for every mode.

Synthetic signals, corresponding to the acceleration time history at each floor, are generated by the function using `scipy`'s `signal.StateSpace` class. All the 5 DOFs are excited by a Gaussian white noise input, then the results from each channel are polluted with a noise source with a signal-to-noise ratio (SNR) equal to 10%.

The input parameters assumed in each OMA method represent a trade-off between computational efficiency and accuracy of the results.

3.1. Preliminary operations

First and foremost, it is fundamental to import all the necessary modules, and the input data as an array. In this example a call to the function `oma.Exdata()` permits to get the input data. On the other hand, if the user is importing its own data, he may take advantage of pandas module, e.g. `pd.read_csv` function. It is worth reminding that the time instant column need to be removed by the user, furthermore attention must be paid to the arguments "`header`", which locate the spreadsheet's table header, and separator character argument "`sep`", to correctly read and import the data file.

```
# Import modules
import numpy as np
import pandas as pd
import seaborn as sns
from scipy import signal
import matplotlib.pyplot as plt
import pyOMA as oma
# ======== PRE-PROCESSING ========
# To open a .txt file create a variable with the path to
    the file
_file = r'C:<Path to the txt file>\Ex_file.txt"

# open the file with pandas and create a dataframe
# N.B. whatchout for header, separator and remove time
    column(s)
data = pd.read_csv(_file, header=0, sep="\t",
    index_col=False)
data = data.to_numpy()

# to retrieve the example data
data, (fex, FI_ex, xi_ex) = oma.Exdata()
```

The user must provide the signals' sampling frequency, expressed in Hz. In the preliminary operations, the PyOMA module allows the user to perform the most common and basic signal pre-processing actions such as detrending, decimation, and filtering procedures, through the standard `scypy.signal` functions. Subsequently, the system is ready to execute the output-only dynamic identification algorithms.

```
# Sampling frequency
fs = 100 # [Hz] Sampling Frequency
```

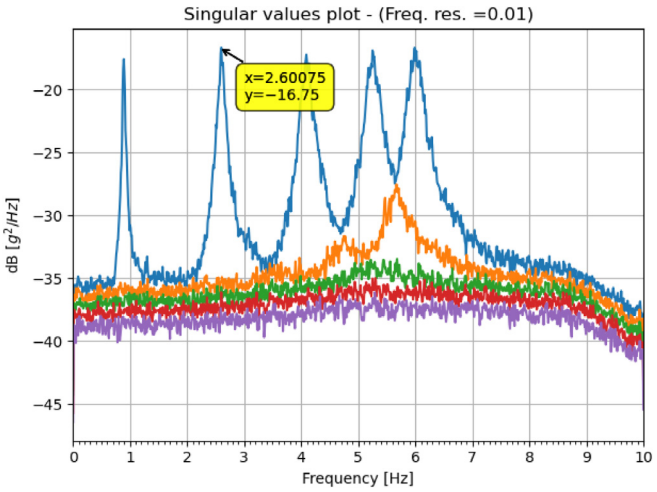


Fig. 3. SV diagram of the PSD matrix obtained from the FDD algorithm.

```
# Using SciPy's signal module user can pre-process the
    data
# e.g. decimation, trend removal and filtering.
# Detrend and decimate
data = signal.detrend(data, axis=0) # Trend removal
q = 5 # Decimation factor
data = signal.decimate(data, q, ftype='fir', axis=0) #
    Decimation
fs = fs/q # [Hz] Decimated sampling frequency

# Filter
_b, _a = signal.butter(12, (0.3,6.5), fs=fs,
    btype='bandpass')
filtdata = signal.filtfilt(_b, _a, data,axis=0) #
    filtered data
```

3.2. Identification — frequency domain

In the current example, the authors focus firstly on `oma.FDDsvp` to run the Frequency Domain Decomposition (FDD) algorithm. This function returns the Singular Values (SV) diagram of the Power Spectral Density (PSD) matrix, and a dictionary that contains the results that will be processed later to extract the modal properties.

```
# Run FDD
FDD = oma.FDDsvp(data, fs)
```

The peaks in Fig. 3 represent an estimate to the natural frequencies of the system. To help the identification of the peaks, the user can take advantage of `mplcursor` module, which enables the click interactivity with the graphed data which are depicted in Fig. 3.

```
# Define list/array with the peaks identified from the
    plot
FreQ = [0.89, 2.6, 4.1, 5.27, 6] # identified peaks
```

At this point, it is possible to run the `oma.FDDmodEX` and/or the `oma.EFDDmodEX` functions to extract the modal information according to the "FDD method" and/or the "Enhanced-FDD (EFDD) method" respectively. All "modEX" functions return a dictionary that contains the results of the identification in terms of modal

properties. The `oma.FDDmodEX` function will only extract the natural frequency and the mode shape, according to the original FDD algorithm as presented in [6]. The `oma.EFDDmodEX()` function has two methods that can be selected. Method "EFDD" extracts the modal properties (frequencies, mode shapes, damping) according to the EFDD algorithm as presented in [7]. The method "FSDD" instead extract the modal properties (frequencies, mode shapes, damping) according to the Frequency-Spatial Domain Decomposition (FSDD) method [8]. The latter method isolates the modal coordinates by modal filtering and provides enhanced output PSD estimates, which in return yield better auto-correlation functions. The required parameters by the functions are: "FreQ", which is the list of peaks previously identified in the SV diagram; "Results" which is the dictionary of results returned by `oma.FDDsvp`. These functions allows the user to customize the return, permitting the extraction of the single DOF (SDOF) bells extraction from the PSD peaks [1], one for each mode, as depicted for instance in Fig. 4.

```
# Extract the modal properties
# extracting modal properties using standard FDD
Res_FDD = oma.FDDmodEX(FreQ, FDD[1])
# extracting modal properties using Enhanced-FDD
Res_EFDD = oma.EFDDmodEX(FreQ, FDD[1], method='EFDD')
# extracting modal properties using FSDD with additional
    input
# arguments to customize the return of function, e.g.
    return plot
Res_FSDD = oma.EFDDmodEX(FreQ, FDD[1], method='FSDD',
    npmax = 35, MAClim=0.95, plot=True)
```

It is also possible to visualize the results of the identification by inspecting the returned dictionaries:

```
Res_FDD['Frequencies'] = [0.89, 2.6, 4.09, 5.25, 5.99]
Res_EFDD['Frequencies'] = [0.888922, 2.60223, 4.08213,
    5.25452, 6.00018]
Res_FSDD['Frequencies'] = [0.888299, 2.60237, 4.09628,
    5.23675, 6.02448]
Res_EFDD['Damping'] = [0.0226723, 0.0211802, 0.0221737,
    0.0186768, 0.0162947]
Res_FSDD['Damping'] = [0.0207433, 0.0199027, 0.0214159,
    0.0196272, 0.0187716]

Res_FDD['Mode Shapes'] =
| 0.25134 | 0.769842 | 1         | 0.885276 | 0.561732 |
| 0.53948 | 1        | 0.270606  | -0.781869 | -0.936679 |
| 0.748727 | 0.567836 | -0.892552 | -0.262885 | 1       |
| 0.920345 | -0.27664 | -0.533285 | 1       | -0.785869 |
| 1        | -0.920871 | 0.73329 | -0.543235 | 0.253549 |

Res_EFDD['Mode Shapes'] =
| 0.267461 | 0.761337 | 1         | 0.91893  | 0.528522 |
| 0.540725 | 1        | 0.283499  | -0.762619 | -0.89945 |
| 0.73491 | 0.544052 | -0.89023  | -0.286976 | 1        |
| 0.931503 | -0.283387 | -0.545335 | 1       | -0.769609 |
| 1        | -0.874993 | 0.741409 | -0.551697 | 0.278106 |

Res_FSDD['Mode Shapes'] =
| 0.271745 | 0.759193 | 1         | 0.920024 | 0.531951 |
| 0.542999 | 1        | 0.282955  | -0.767985 | -0.904347 |
| 0.741326 | 0.542259 | -0.900547 | -0.284978 | 1       |
| 0.932505 | -0.280532 | -0.544536 | 1       | -0.77068 |
| 1        | -0.881809 | 0.745016 | -0.552481 | 0.277287 |
```

3.3. Identification — time domain

Regarding to time domain methods, PyOMA module allows the user to run the stochastic subspace identification (SSI) algorithm, both the data-driven version denoted as "SSI-dat" [9] with the command `oma.SSIdatStaDiag`, and even the covariance-based approach denoted as "SSI-cov" [2] through the command `oma.SSIcovStaDiag`. To analyze the data with these time domain parametric procedures one can run the following functions:

```
# Run SSI
br = 15 # number of block rows (time lags)
# running SSI-cov
SSIcov= oma.SSIcovStaDiag(data, fs, br)
# running SSI-dat with additional input parameters
SSIdat = oma.SSIdatStaDiag(data, fs, br, ordmax=60,
    lim=(0.01, 0.05, 0.02, 0.1))
```

For these functions, the required parameters are the dataset, the sampling frequency and the number of block rows (time lags). The optional parameters allow the user to define the maximum model order, minimum order, and the limit values to be used for the stability criteria of the poles in the stabilization diagram, which are depicted in Fig. 5. The scholars can also use the `mplcursor` module to identify the stable frequency lines in a click-based interactive diagram approach.

Once the identified frequencies have been collected, it is possible to extract the estimates of the modal properties. Therefore, the user may just run the function `SSIModEX()`, passing to it the array/list of frequencies identified in the stabilization diagram. The dictionary of results are returned by either `oma.SSIcovStaDiag()` or `oma.SSIdatStaDiag()`.

```
# Extract the modal properties
Res_SSIcov = oma.SSIModEX(FreQ, SSIcov[1])
Res_SSIdat= oma.SSIModEX(FreQ, SSIdat[1])
```

It is also possible to visualize the identification results by inspecting the output dictionaries:

3.4. Post-processing

The PyOMA module provides the user with simple functions to perform post-processing procedures for the results. For instance, it is possible to evaluate the similarity between mode shapes of different identified modes by calculating the modal assurance criterion (MAC) between the eigenvectors. It is also possible to define a diagram with `seaborn` module with the function `heatmap` for the cross-MAC indicator, as depicted in Fig. 6.

```
# ======
# Make some plots
# ======
MS_FDD = Res_FDD['Mode Shapes'].real
MS_EFDD = Res_EFDD['Mode Shapes'].real
MS_FSDD = Res_FSDD['Mode Shapes'].real
MS_SSIcov = Res_SSIcov['Mode Shapes'].real
MS_SSIdat = Res_SSIdat['Mode Shapes'].real
_nch = data.shape[1]

MAC = np.reshape(
        [oma.MaC(FI_ex[:,l], MS_FSDD[:,k]).real for k in
            range(_nch) for l in range(_nch)], #
            (_nch*_nch) list of MAC values
        (_nch, _nch)) # new (real) shape (_nch x _nch) of
            the MAC matrix

crossMAC = np.reshape(
        [oma.MaC(MS_SSIcov[:,l], MS_SSIdat[:,k]).real
            for k in range(_nch) for l in range(_nch)], #
            (_nch*_nch) list of MAC values
        (_nch,_nch)) # new (real) shape (_nch x _nch) of
            the MAC matrix
```

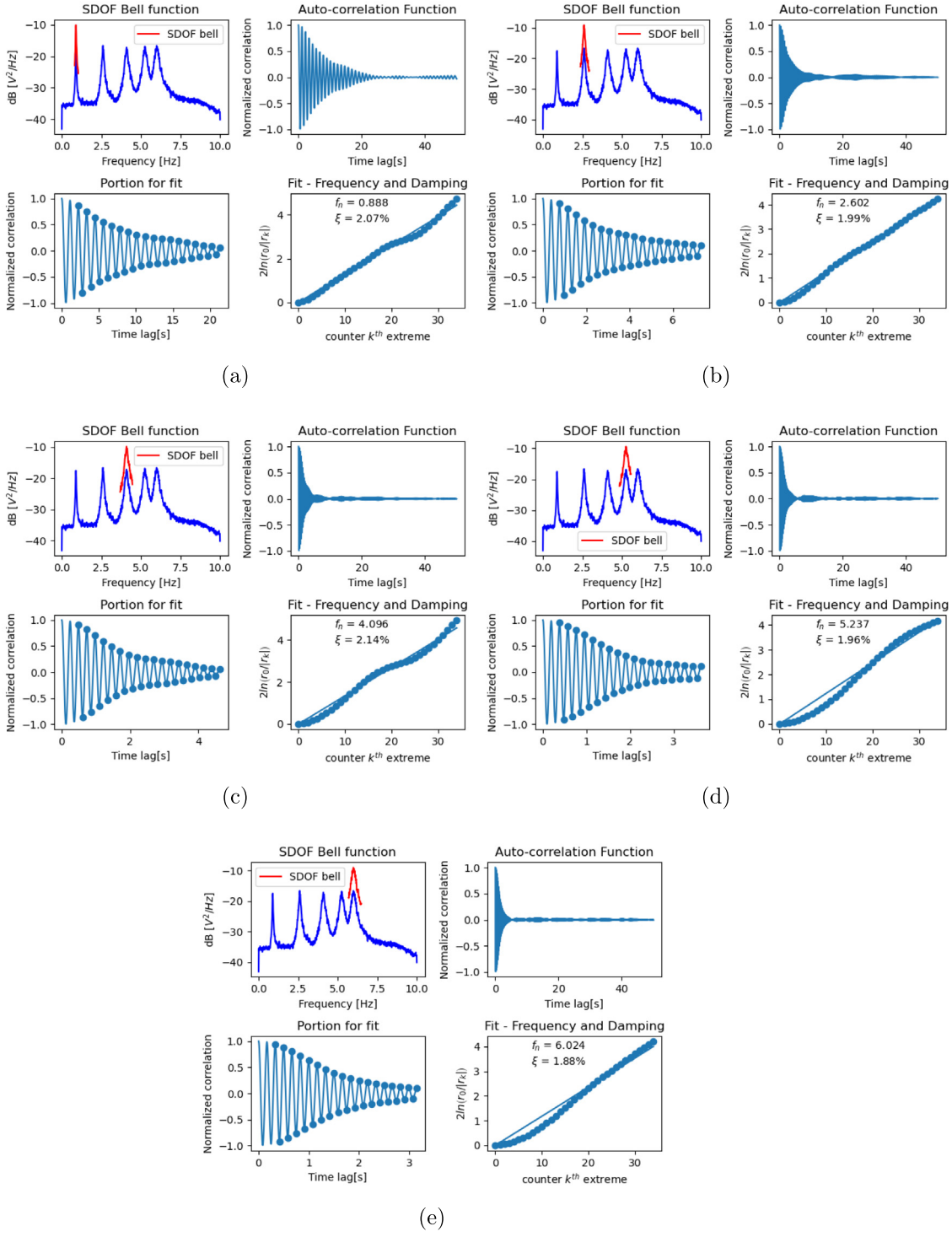
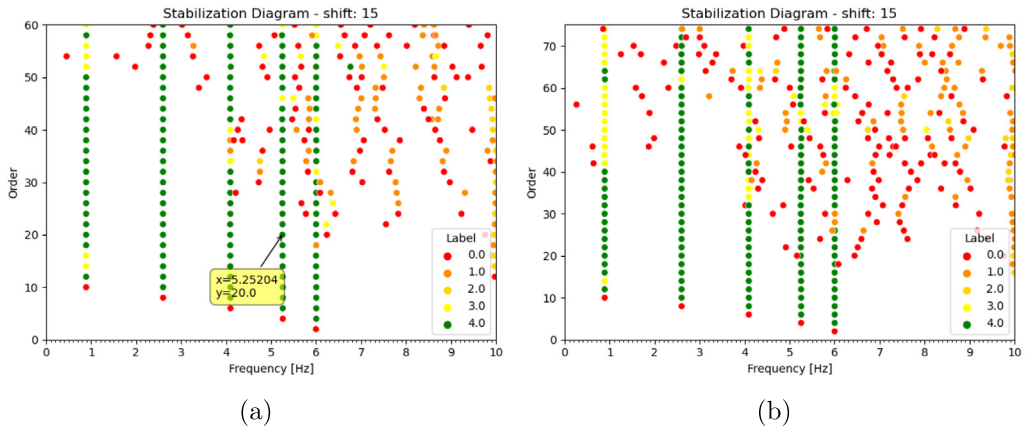


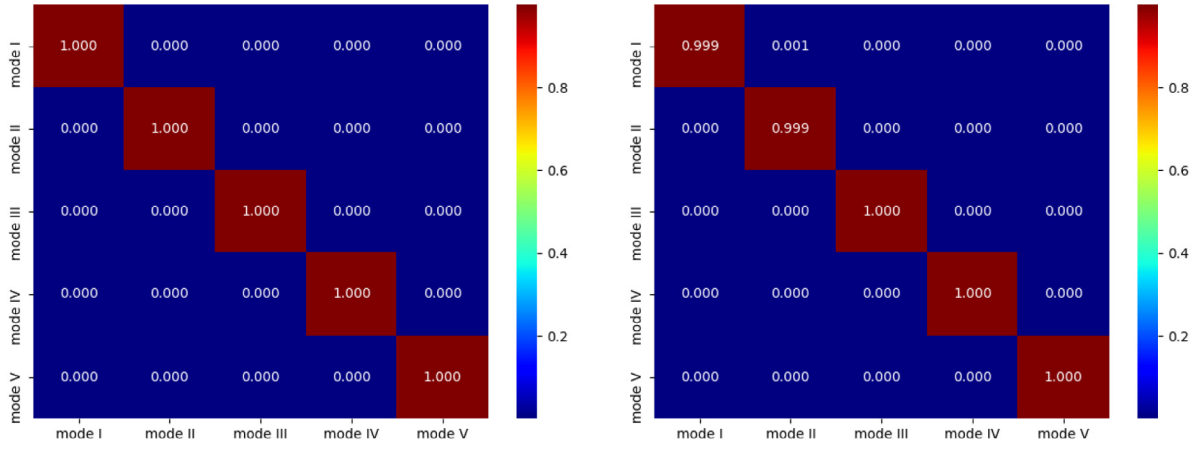
Fig. 4. Single DOF (SDOF) bell extraction from the post-processing of FSDD results.

```
col = ["mode I", "mode II", "mode III", "mode IV", "mode V"]

MAC = pd.DataFrame(MAC, columns=col, index=col)
crossMAC = pd.DataFrame(crossMAC, columns=col,
     index=col)

fig, ax = plt.subplots()
sns.heatmap(MAC,cmap="jet",ax=ax,annot=True, fmt='.3f',)
fig.tight_layout()
plt.show()

fig, ax1 = plt.subplots()
sns.heatmap(crossMAC,cmap="jet", ax=ax1, annot=True,
     fmt='.3f',)
fig.tight_layout()
plt.show()
```



(a) (b)

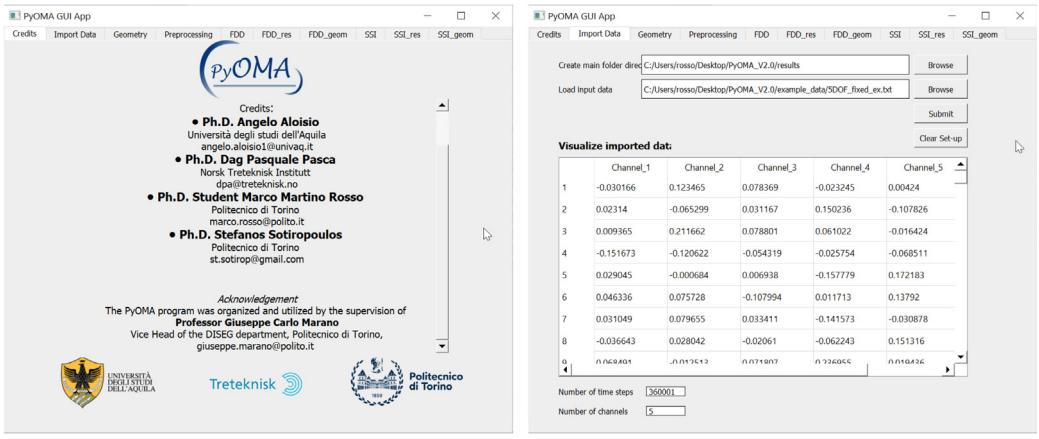
Fig. 5. Stabilization diagrams resulting for SSI-dat (a) and SSI-cov (b) methods.



(a) (b)

Fig. 6. Representation of MAC (a) among the modes from the same technique and cross-MAC (b) comparing optimal results provided by SSI-cov and SSI-dat approaches.

```
Res_SSIcov['Frequencies'] = [0.888782, 2.5977, 4.09384, 5.25337, 6.00183]
Res_SSIdat['Frequencies'] = [0.888743, 2.59783, 4.0945, 5.25339, 6.00224]
Res_SSIcov['Damping'] = [0.0205092, 0.0198478, 0.0211763, 0.0207207, 0.0190943]
Res_SSIdat['Damping'] = [0.0216251, 0.0198527, 0.0229909, 0.0208779, 0.0200183]

Res_SSIcov['Mode Shapes'] =
| 0.265703 | 0.780006 | 1        | 0.911646 | 0.547661 |
|----------|----------|----------|----------|----------|
| 0.538122 | 1        | 0.275115 | -0.76086 | -0.923849 |
| 0.741008 | 0.56412  | -0.899589 | -0.280149 | 1       |
| 0.928411 | -0.288147 | -0.536196 | 1       | -0.774282 |
| 1        | -0.944359 | 0.742725 | -0.545645 | 0.275831 |

Res_SSIdat['Mode Shapes'] =
| 0.279095 | 0.772773 | 1        | 0.925599 | 0.542878 |
|----------|----------|----------|----------|----------|
| 0.53965 | 1        | 0.285698 | -0.768124 | -0.916186 |
| 0.742472 | 0.551479 | -0.914432 | -0.286194 | 1       |
| 0.937415 | -0.282122 | -0.550359 | 1       | -0.770326 |
| 1        | -0.924122 | 0.754373 | -0.555319 | 0.275067 |
```



(a) (b)

Fig. 7. PyOMA_GUI 5 DOFs shear type example overview. (a) Initial tab after starting the software execution; (b) Import data tab.

It is possible to visit the documentation page for further details on the PyOMA implemented functions.

3.5. OMA with the graphical user interface approach

The PyOMA_GUI has been implemented with PyQt5 library, adopting a notebook-style with tabs that noticeably help every type of user follow the right steps to perform OMA in a semi-assisted approach. In the following, the main steps in Fig. 2 are briefly described and discussed. However, with the software, the authors provided the files with the same example of the 5 Degrees of Freedom (DOFs) shear-type frame discussed in the previous sections.

3.5.1. Import data

After running the executable PyOMA_GUI file, the user must select the first tab to import vibration data, as illustrated in Fig. 7. In the first place, the user must select the current working directory, i.e. a folder in which the output files will be stored. It can be in any location on the user's machine, and it can be an existing folder or a new one that may be created on the dialogue box. After that, the user must browse the input data file, which will also be displayed in the table to check the success of the data uploading. At the user's will, it is possible to customize the imported data table headers' names with a simple double click. The data visualization permits the user to check the number of time steps and the number of channels. To proceed with the following stages of the OMA procedure, the user must push the `Submit` button. Furthermore, to start a new analysis, the user may press the `Clear Set-up` button to completely reset the PyOMA_GUI software memory and cleanse the uploaded data.

3.5.2. Geometry

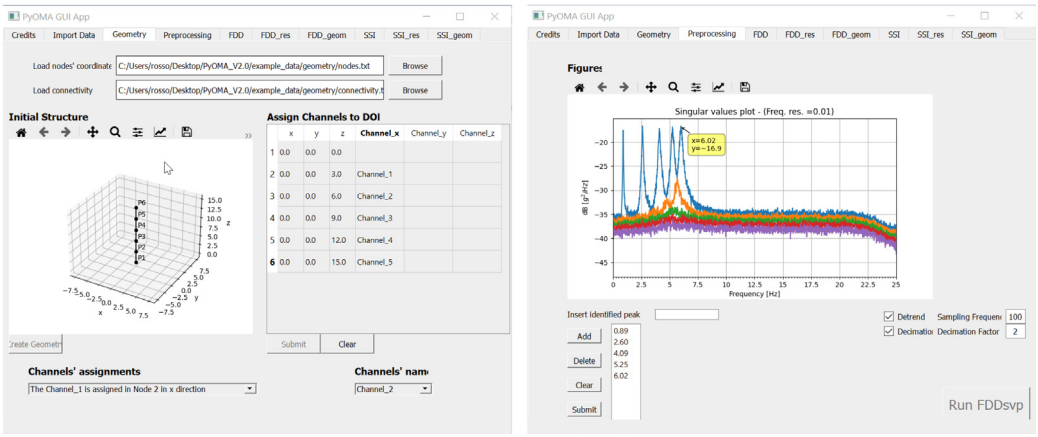
The geometry definition is fundamental to provide the user with a simplified visualization of the identified mode shapes according to the monitored DOFs and the available measurement channels, as depicted in Fig. 8(a). The user must browse a text file which contains the nodes' coordinates and connectivity to visualize a starting undeformed scheme of the structure under study according to the monitored DOFs only. Pushing the button `Create Geometry` a wire-frame graph of the structure appears. In the table `Assign Channels to DOF`, the user must insert the exact index of the table (Channel_x, Channel_y, Channel_z) and the precise name of the monitored channel/DOF. Those identical names are shown in the combo-box Channels' name to remind the user, preventing him from returning to the previous imported data tab tediously. In the combo box, the exact assignments of the channels are provided.

3.5.3. Pre-processing and peak-peaking approach

The PyOMA_GUI allows the user to perform signal basic pre-processing procedures in the same manner as the PyOMA module, illustrated in Fig. 8(b). The user must select the parameters of the problem, e.g. the sampling frequency and decimation factor and some default values are provided if the user does not set them explicitly. Thereafter, it is possible to run the `FDD_svp` function. In the SV graph figure, it is possible to perform the peak-peaking approach to select the peak of interest, and with the button `Add`, they are added to the list of the identified peaks. The other buttons close to the list permits customizing the identified peaks list, e.g. deleting a single item or clearing the whole list. With the `Submit` button, the user confirms the selection of the identified peaks, and it is possible to proceed with analyzing the results of the FDD approaches.

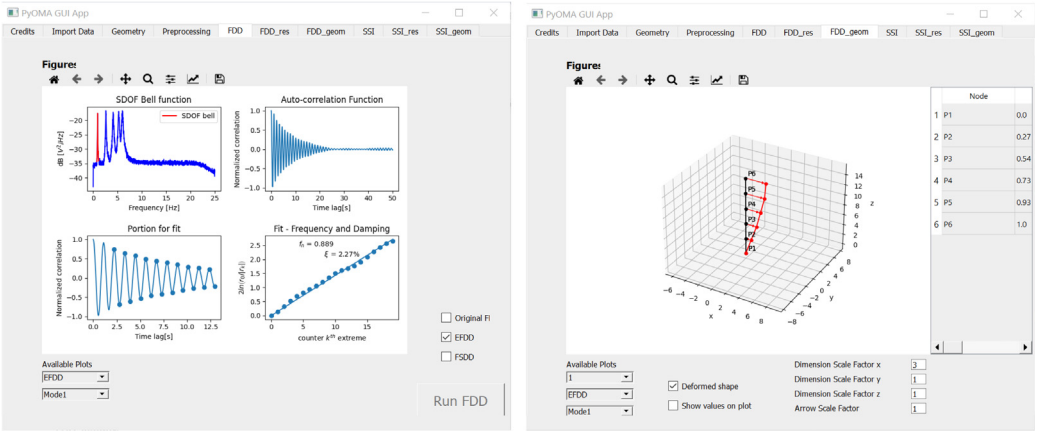
3.5.4. FDD, SSI and their implemented variants with PyOMA_GUI

Passing to the next tab of the notebook, it is possible to run the FDD, SSI and their implemented variants in the PyOMA module with the graphical user interface software. In the FDD tab, the user can choose in the checkboxes the methods of interest among FDD, EFDD and FSDD. Pushing the button Run FDD, the results are illustrated in terms of SDOF bell extraction for each mode. The available diagrams are listed in the dropdown menu in the bottom-left part of the window, and the above figure refreshes the user anytime to modify its selection. In the `FDD_res` tab, the user may explore all the frequency domain algorithms' results in tabular form. All the results (figures included) are automatically stored in the working directory previously selected. In the `FDD_geom` tab, the final mode shape deformations are presented in several forms, as depicted in Fig. 9. By selecting the desired mode, the figure is updated. In the table, the deformation on each node is presented, while extra functionalities for the plot are provided. The checkbox `Deformed shape` illustrates in the same figure the deformed structure, and the checkbox `Show values` activates the functionality to show the information of the table directly on the figure. At the same time, four scaling factors are also provided to adjust and customize the mode shape appearance. The last remaining tabs are related to the SSI method, which working mechanisms are similar and equivalent to the so-far illustrated frequency-domain methods. The stabilization diagram obtained with the PyOMA_GUI software is illustrated in Fig. 10.



(a) (b)

Fig. 8. PyOMA_GUI 5 DOFs shear type example overview. (a) geometry definition; (b) Preprocessing and FDD algorithm execution with SV decomposition diagram for peak-peaking identification.



(a) (b)

Fig. 9. PyOMA_GUI 5 DOFs shear type example overview. (a) geometry definition; (b) Preprocessing and FDD algorithm execution with SV decomposition diagram for peak-peaking identification.

4. Impact

To this date, a few commercial software implements the two algorithms mentioned above. The most known presumably are ARTeMIS [15], by Structural Vibration Solutions, and MACEC, a Matlab toolbox for modal testing and OMA [16]. However, to the author's best knowledge, there is no Python module nor any other open-source complete toolbox to perform output-only OMA. For this reason, the authors have developed the present PyOMA module. The API for PyOMA provides a set of functions for a quick and straightforward estimation of the natural frequencies, mode shapes and damping using the experimental data recorded by the user. Specifically, the user needs to specify only a minimal amount of input parameters in addition to the measurement data. For a complete description of the functionalities, please refer to the documentation page. The flowchart in Fig. 1 shows the general architecture of PyOMA. Furthermore, the greatest impact is provided by the development of the PyOMA_GUI, for which a general overview of the main functionalities is depicted in Fig. 2. The software provides a graphical user interface (GUI) approach designed to be adopted by both researchers, civil engineers and practitioners without requiring any Python expertise or Python coding knowledge prerequisite. The PyOMA_GUI aims at increasing the impact of the current open-source Python OMA module, which has already been used in several applications, as proved by several scientific publications: [17–26].

5. Conclusions

The authors developed a new python module and open-source software, PyOMA and PyOMA-GUI, respectively, to perform dynamic identification of structures from output-only vibration measurements. To the authors' knowledge, there are no currently structured python modules for Operational Modal Analysis (OMA). OMA represents a standard practice in the diagnosis phase of structures within any structural health monitoring paradigm. By providing a structured open-source software and framework to perform OMA, the authors hope to provide the scientific community and practitioners with a fundamental tool. For this reason, the authors developed a graphical user interface version of this python module to increase the impact of the software among users without particular expertise in python coding.

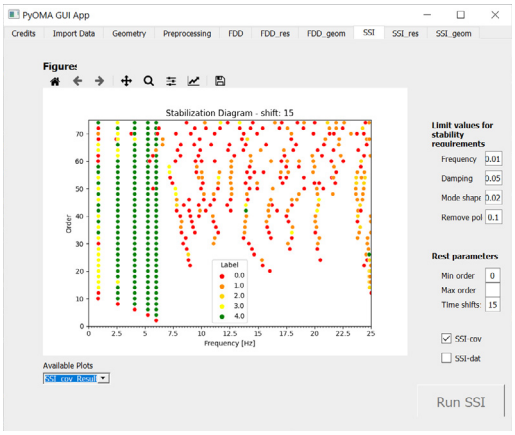


Fig. 10. PyOMA_GUI 5 DOFs shear type example overview. Stabilization diagram retrieved with SSI algorithm.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

Data will be made available on request.

Acknowledgments

The authors acknowledge the meaningful contribution of Professor Rocco Alaggio from Università degli Studi dell'Aquila, who encouraged the authors to study and develop these topics. The authors acknowledge the meaningful contribution of Professor Giuseppe Carlo Marano from Politecnico di Torino for promoting the Graphical User Interface programming and coordinating the team activities.

Appendix. Function's description

- **MaC(Fi1, Fi2):**
  This function returns the Modal Assurance Criterion (MAC) for two mode shape vectors. If the input arrays are in the form (n,) (1D arrays) the output is a scalar, if the input are in the form (n,m) the output is a (m,m) matrix (MAC matrix).
  Parameters
  - Fi1: array (1D or 2D)
    First mode shape vector (or matrix).
  - Fi2: array (1D or 2D)
    Second mode shape vector (or matrix).

  Returns
  - MAC : float or (2D array)
    Modal Assurance Criterion.

- **Exdata():**
  This function generates a time history of acceleration for a 5 DOF system. The function returns a (360001,5) array and a tuple containing: the natural frequencies of the system (fn = (5,) array); the unity displacement normalized mode shapes matrix (FI1 = (5,5) array); and the damping ratios (xi = float).
  Returns
  - acc : 2D array
    Time histories of the 5 DOF of the system.
  - (fn, FI1, xi) : tuple
    Tuple containing the natural frequencies (fn), the mode shape matrix (FI1), and the damping ratio (xi) of the system.

- **SSIcovStaDiag(data, fs, br, [ordmax = [None], lim = [(0.01,0.05,0.02,0.1)], method = ['1']])**
  This function perform the covariance-driven Stochastic subSpace Identification algorithm. The function returns the Stabilization Diagram (Plot) for the given data. Furthermore it returns a dictionary that contains the results needed by the function SSImodEX().
  Parameters
  - data: 2D array
    The time history records (N° data points *x* N° channels).
  - fs: float
    Sampling frequency of the time series.
  - br: int
    Number of block rows.
  - ordmax: None or int, optional
    The maximum model order to use in the construction of the stabilization diagram. None (default) is equivalent to the maximum allowable model order equal to br*data.shape[1].
  - lim: tuple, optional
    Limit values to use for the stability requirements of the poles. The first three values are used to check the stability of the poles.
  - lim: tuple, optional
    Limit values to use for the stability requirements of the poles. The first three values are used to check the stability of the poles.
    Frequency: (f(n)-f(n+1))/f(n) < lim[0] (default to 0.01)
    Damping: (xi(n)-xi(n+1))/xi(n) < lim[1] (default to 0.05)
    Mode shape: 1-MAC((phi(n),phi(n+1)) < lim[2] (default to 0.02)
    The last value (lim[3]) is used to remove all the poles that have a higher damping ratio (default to 0.1, N.B. in structural dynamics we usually deal with underdamped system)
  - method: '1', '2', optional
    Method to use in the estimation of the state matrix A: method "1" (default) : the first method takes advantages of the shift structure of the observability matrix. method "2" : the second method is based on the decomposition property of the one-lag shifted Toeplitz matrix.

  Returns
  - fig1 : matplotlib figure
    Stabilization diagram. Take advantage of the mplcursors module to identify the stable poles.
  - Results : dictionary
    Dictionary of results. This dictionary will be passed as argument to the SSImodEX() function.

- **SSIdatStaDiag(data, fs, br, [ordmax = [None], lim = [(0.01,0.05,0.02,0.1)], method = ['1']])**

This function perform the Data-driven Stochastic sub-Space Identification algorithm. The function returns the Stabilization Diagram (Plot) for the given data. Furthermore it returns a dictionary that contains the results needed by the function SSImodEX().
Parameters

- data: 2D array
  The time history records (N° data points *x* N° channels).
- fs: float
  Sampling frequency of the time series.
- br: int
  Number of block rows.
- ordmax: None or int, optional
  The maximum model order to use in the construction of the stabilization diagram. None (default) is equivalent to the maximum allowable model order equal to br*data.shape[1].
- lim: tuple, optional
  Limit values to use for the stability requirements of the poles. The first three values are used to check the stability of the poles.
- lim: tuple, optional
  Limit values to use for the stability requirements of the poles. The first three values are used to check the stability of the poles.
  Frequency: (f(n)-f(n+1))/f(n) $<$ lim[0] (default to 0.01)
  Damping: (xi(n)-xi(n+1))/xi(n) $<$ lim[1] (default to 0.05)
  Mode shape: 1-MAC((phi(n),phi(n+1)) $<$ lim[2] (default to 0.02)
  The last value (lim[3]) is used to remove all the poles that have a higher damping ratio (default to 0.1, N.B. in structural dynamics we usually deal with underdamped system)
- method: '1', '2', optional
  Method to use in the estimation of the state matrix A: method "1" (default) : the first method uses the kalman state sequence S(i+1)
  method "2" : the second method takes advantages of the shift of the observability matrix

Returns

- fig1 : matplotlib figure
  Stabilization diagram. Take advantage of the mplcursors module to identify the stable poles.
- Results : dictionary
  Dictionary of results. This dictionary will be passed as argument to the SSImodEX() function.

- **SSIModEX(FreQ, Results, [deltaf = [0.05], aMaClim = [0.95]]):**
  This function extracts the modal properties (frequencies, damping ratios, mode shapes) and returns the results organized in a dictionary. This function takes as second argument the results from either SSIdatStaDia() or SSIcovStaDia() functions.
  Parameters
  - FreQ: array (or list)
    Array containing the frequencies, identified from the stabilization diagram, which we want to extract.
  - Results: dict
    Dictionary of results (returned by either SSIcovStaDiag or SSIdatStaDiag).
  - deltaf: float, optional
    Tolerance to use when searching for FreQ[i] in the results. Default to 0.05.
  - aMaClim: float, optional
    Modal Assurance Criterion limit value. The poles which have a MAC value less than aMaClim are excluded from the calculation of the statistics of the modal properties.

  Returns
  - Results: dictionary
    Dictionary containing the modal properties (frequencies, damping ratios, mode shapes) of the system.

- **FDDsvp(data, fs, [df = [0.01], pov = [0.5], window = ['hann']]):**
  This function perform the Frequency Domain Decomposition algorithm. The function return the plot of the singular values of the power spectral density. The cross power spectral density is estimated using scipy.signal.csd() function, which in turn is based on Welch's method. Furthermore it returns a dictionary that contains the results needed by the function FDDmodEX().
  Parameters
  - data: array
    The time history records (Ndata *x* Nchannels).
  - fs: float
    Sampling frequency of the time series.
  - df: float, optional
    Desired frequency resolution. Default to 0.01 (Hz).
  - pov: float, optional
    Percentage of overlap between segments. Default to 50%.
  - window: str or tuple or array like, optional
    Desired window to use. Window is passed to scipy.signal's get window function (see SciPy.org for more info). Default to "hann" which stands for a "Hanning" window.

  Returns
  - fig1: Figure
    Plot of the singular values of the power spectral matrix.
  - Results: dictionary
    Dictionary of results to be passed to FDDmodEX().

- **FDDmodEX(FreQ, Results, [ndf = [2]]):**
  This function returns the modal parameters estimated according to the Frequency Domain Decomposition method.
  Parameters
  - FreQ: array (or list)
    Array containing the frequencies, identified from the singular values plot, which we want to extract.
  - Results: dict
    Dictionary of results obtained from FDDsvp().
  - ndf: float, optional
    Number of spectral lines in the proximity of FreQ[i] where the peak is searched.

  Returns
  - Results: dictionary
    Dictionary of results

- **EFDDmodEX(FreQ, Results, [ndf = [2], MAClim = [0.85], sppk = [3], npmax = [20], method = ['FSDD'], plot = [False]):**

This function returns the modal parameters estimated according to the enhanced version of the Frequency Domain Decomposition method.

Parameters

- FreQ: array (or list)
  Array containing the frequencies, identified from the singular values plot, which we want to extract.
- Results: dict
  Dictionary of results obtained from FDDsvp().
- ndf: float, optional
  Number of spectral lines in the proximity of FreQ[i] where the peak is searched.
- MaClim: float, optional
  MAC rejection level for the extraction of the SDOF bell function.
- sppk: int, optional
  Number of peaks to skip at the beginning of the autocorrelation function when calculating the damping ratio (through the fit on the log decrement)
- npmax: int, optional
  Number of (consecutive) points to use in the autocorrelation function calculating the damping ratio (through the fit on the log decrement)
- method: 'FSDD', 'EFDD', optional
  Method used to extract the SDOF bell function. Default to "FSDD", uses the Frequency Spatial Domain Decomposition algorithm. Method "EFDD" uses the classical Enhanced Frequency Domain Decomposition algorithm.
- plot : True or False, optional
  Whether to plot or not the results. Default to False.

Returns

- Figure:
  Figures with spectral density, corr. funct., fit on results
- Results: dictionary
  Dictionary of results

References

[1] Rainieri C, Fabbrocino G. Operational modal analysis of civil engineering structures. Vol. 142, Springer, New York; 2014, p. 143. http://dx.doi.org/10.1007/978-1-4939-0767-0.

[2] Peeters B, De Roeck G. Reference-based stochastic subspace identification for output-only modal analysis. Mech Syst Signal Process 1999;13(6):855–78. http://dx.doi.org/10.1006/mssp.1999.1249, URL https://www.sciencedirect.com/science/article/pii/S0888327099912499.

[3] Doebling SW, Farrar CR, Prime MB, Shevitz DW. Damage identification and health monitoring of structural and mechanical systems from changes in their vibration characteristics: a literature review. 1996.

[4] James GH, Carne TG, Lauffer JP, Nord AR. Modal testing using natural excitation. In: Proceedings of the international modal analysis conference. Sem Society for Experimental Mechanics Inc; 1992, p. 1208.

[5] Juang J-N, Pappa RS. An eigensystem realization algorithm for modal parameter identification and model reduction. J Guid Control Dyn 1985;8(5):620–7.

[6] Brincker R, Zhang L, Andersen P. Modal identification of output-only systems using frequency domain decomposition. Smart Mater Struct 2001;10(3):441. http://dx.doi.org/10.1088/0964-1726/10/3/303.

[7] Brincker R, Ventura CE, 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. Society for Experimental Mechanics; 2001, p. 698–703.

[8] Zhang L, Wang T, Tamura Y. A frequency–spatial domain decomposition (FSDD) method for operational modal analysis. Mech Syst Signal Process 2010;24(5):1227–39. http://dx.doi.org/10.1016/j.ymssp.2009.10.024, URL https://www.sciencedirect.com/science/article/pii/S0888327009003744.

[9] Van Overschee P, De Moor B. Subspace identification for linear systems: theory—implementation—applications. Springer Science & Business Media; 2012.

[10] Bendat J, Piersol A. Engineering applications of correlation and spectral analysis. 1980, cited By 2739.

[11] Van Overschee P, De Moor B, Suykens J. Subspace algorithms for system identification and stochastic realization. In: Proc. of the international symposium MTNS-91. 1991, p. 589–95, cited By 10.

[12] Zhang L, Brincker R, Andersen P. An overview of operational modal analysis: Major development and issues. In: Proceedings of the 1st international operational modal analysis conference, IOMAC 2005. 2005, cited By 174.

[13] Reynders E. System identification methods for (operational) modal analysis: Review and comparison. Arch Comput Methods Eng 2012;19(1):51–124. http://dx.doi.org/10.1007/s11831-012-9069-x, cited By 461.

[14] Chopra AK. Dynamics of structures. Pearson Prentice Hall; 1975.

[15] Solutions Structural Vibration. Artemis extractor: Ambient response testing and modal identification software, user's manual. 2001.

[16] Reynders E, Schevenels M, De Roeck G. MACEC 3.2: A matlab toolbox for experimental and operational modal analysis. Department of Civil Engineering, KU Leuven; 2014.

[17] Alaggio R, Aloisio A, Antonacci E, Cirella R. Two-years static and dynamic monitoring of the santa maria di collemaggio basilica. Constr Build Mater 2021;268:121069. http://dx.doi.org/10.1016/j.conbuildmat.2020.121069, URL https://www.sciencedirect.com/science/article/pii/S0950061820330737.

[18] Aloisio A, Di Pasquale A, Alaggio R, Fragiacomo M. Assessment of seismic retrofitting interventions of a masonry palace using operational modal analysis. Int J Archit Heritage 2020;1–13. http://dx.doi.org/10.1080/15583058.2020.1836531.

[19] Aloisio A, Battista LD, Alaggio R, Antonacci E, Fragiacomo M. Assessment of structural interventions using Bayesian updating and subspace-based fault detection methods: The case study of S. Maria di collemaggio basilica, L'Aquila, Italy. Struct Infrastruct Eng 2021;17(2):141–55. http://dx.doi.org/10.1080/15732479.2020.1731559.

[20] Aloisio A, Pasca DP, Alaggio R, Fragiacomo M. Bayesian estimate of the elastic modulus of concrete box girders from dynamic identification: A statistical framework for the A24 motorway in Italy. Struct Infrast Eng 2020;1–13. http://dx.doi.org/10.1080/15732479.2020.1819343.

[21] Aloisio A, Capanna I, Cirella R, Alaggio R, Di Fabio F, Fragiacomo M. Identification and model update of the dynamic properties of the san silvestro belfry in l'aquila and estimation of bell's dynamic actions. Appl Sci 2020;10(12):4289. http://dx.doi.org/10.3390/app10124289, URL https://www.mdpi.com/2076-3417/10/12/4289.

[22] Aloisio A, Antonacci E, Fragiacomo M, Alaggio R. The recorded seismic response of the Santa Maria di Collemaggio basilica to low-intensity earthquakes. Int J Archit Heritage 2020;1–19. http://dx.doi.org/10.1080/15583058.2020.1802533.

[23] Aloisio A, Alaggio R, Fragiacomo M. Dynamic identification and model updating of full-scale concrete box girders based on the experimental torsional response. Constr Build Mater 2020;264:120146. http://dx.doi.org/10.1016/j.conbuildmat.2020.120146, URL https://www.sciencedirect.com/science/article/pii/S0950061820321516.

[24] 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. Eng Struct 2020;215:110619. http://dx.doi.org/10.1016/j.engstruct.2020.110619, URL https://www.sciencedirect.com/science/article/pii/S014102961934101X.

[25] Capanna I, Cirella R, Aloisio A, Alaggio R, Di Fabio F, Fragiacomo M. Operational modal analysis, model update and fragility curves estimation, through truncated incremental dynamic analysis, of a masonry belfry. Buildings 2021;11(3):120. http://dx.doi.org/10.3390/buildings11030120, URL https://www.mdpi.com/2075-5309/11/3/120.

[26] Aloisio A, Pasca DP, Battista LD, Rosso MM, Cucuzza R, Marano GC, et al. Indirect assessment of concrete resistance from FE model updating and Young's modulus estimation of a multi-span PSC viaduct: Experimental tests and validation. Structures 2022;37:686–97. http://dx.doi.org/10.1016/j.istruc.2022.01.045, URL https://www.sciencedirect.com/science/article/pii/S2352012422000455.