Cite as:

Janko Slavič, Klemen Zaletelj, Domen Gorjup, Dag Pasquale Pasca, Angelo Aloisio, Knut Andreas Kvåle, Gunnstein Thomas Frøseth, Wout Weijtjens, Ahmed Mujtaba, Martin Česnik, Aleš Zorman, George Tsialiamanis, Ivan Tomac, Thiago G. Ritto, Domen Ocepek, Dan Rohe, Olavo M. Silva, Thijs C.P. Masmeijer, Krištof Čufar, Tomaž Bregar, Miha Kodrič, Ahmed El Mahmoudi, Miha Pogačar, Francesco Trainotti, Mert Göldeli, Gregor Čepon, Raphael Timbó, Aldemir A. Cavallini Jr., Vinicius T. Costa, Jéssica G.F. Santos, Aidan J. Hughes, Jacson G. Vargas, André Fernandes, Daniel J. Rixen, Keith Worden, Miha Boltežar, Python open-source signal processing and modeling/simulation for scientific research in structural dynamics a review, Mechanical Systems and Signal Processing, Volume 241, 2025, 113465, ISSN 0888-3270, https://doi.org/10.1016/j.ymssp.2025.113465.

Python Open-source Signal Processing and Modeling/Simulation for Scientific Research in Structural Dynamics - a review

Janko Slavič[a,*], Klemen Zaletelj[a], Domen Gorjup[a], Dag Pasquale Pasca[b], Angelo Aloisio[c], Knut Andreas Kvåle[d], Gunnstein Thomas Frøseth[d], Wout Weijtjens[e], Ahmed Mujtaba[e], Martin Česnik[a], Aleš Zorman[a], George Tsialiamanis[f], Ivan Tomac[g], Thiago G. Ritto[h], Domen Ocepek[a], Dan Rohe[i], Olavo M. Silva[j], Thijs C.P. Masmeijer[k], Krištof Čufar[a,l], Tomaž Bregar[m], Miha Kodrič[a], Ahmed El Mahmoudi[n], Miha Pogačar[a], Francesco Trainotti[o], Mert Göldeli[p], Gregor Čepon[a], Raphael Timbó[q], Aldemir A. Cavallini Jr.[r], Vinicius T. Costa[r], Jéssica G. F. Santos[r], Aidan J. Hughes[f], Jacson G. Vargas[j], André Fernandes[j], Daniel J. Rixen[o], Keith Worden[f], Miha Boltežar[a]

[a] *University of Ljubljana, Faculty of Mechanical Engineering, Aškerčeva 6, 1000 Ljubljana, Slovenia*
[b] *Norsk Treteknisk Institutt (Norwegian Institute of Wood Technology), Oslo, Norway*
[c] *Department of Civil, Construction-Architectural and Environmental Engineering, Università degli Studi dell'Aquila, L'Aquila, Italy*
[d] *Department of Structural Engineering, Norwegian University of Science and Technology, 7491, Trondheim, Norway*
[e] *Acoustics and Vibrations Research Group (AVRG), Vrije Universiteit Brussel (VUB), 1050, Brussels, Belgium*
[f] *University of Sheffield, School of Mechanical, Aerospace and Civil Engineering, Mappin St, Sheffield, S1 3JD, United Kingdom*
[g] *University of Split, Faculty of Electrical Engineering, Mechanical Engineering and Naval Architecture*
[h] *Department of Mechanical Engineering, Universidade Federal do Rio de Janeiro, Brazil*
[i] *Sandia National Lab.(SNL-NM), Albuquerque, NM (United States)*
[j] *Acoustics and Vibration Laboratory, Federal University of Santa Catarina, Florianopolis, Brazil*
[k] *University of Washington, William E. Boeing Department of Aeronautics and Astronautics, 3940 Benton Lane NE Guggenheim Hall, Seattle, Washington, 98195*
[l] *Domel, d.o.o. Otoki 21, 4228 elezniki, Slovenia*
[m] *Gorenje d.o.o., Partizanska 12, Velenje, 3503, Slovenia*
[n] *FEV Vehicle GmbH, Neuenhofstrae 181, Aachen, NRW 52078, Germany*
[o] *Technical University of Munich, School of Engineering and Design, Boltzmannstr. 15, 85748 Garching, Germany*
[p] *dSPACE GmbH, Rathenaustrae 26, 33102 Paderborn, Germany*
[q] *Petrobras, Rio de Janeiro, Brazil*
[r] *Department of Mechanical Engineering, Universidade Federal de Uberlndia, Brazil*

Abstract

Open-source scientific research has become indispensable, because it fosters a global collaborative environment in which knowledge is freely shared, accelerating the pace of discovery. Emphasizing transparency and reproducibility ensures the credibility of scientific findings, while democratizing access to research gives a wider community the opportunity to contribute to progress. Avoiding duplication of effort leads to faster progress and greater societal benefit. The Python programming language provides readable code and is highly collaborative, leading to a large user community. Libraries such as NumPy (i.e. numerical Python) and SciPy (i.e. scientific computing) have been developed for scientific research. The latter was crucial, for

*Corresponding author

example, in the discovery of gravitational waves and the first imaging of a black hole.

This review focuses on selected open-source Python libraries that support scientific development in the field of signal processing and mechanical systems. In particular, the article focuses on packages related to signal processing, experimental and operational modal analysis, vibration fatigue, image-based identification of structural dynamics, vibration control, substructuring, rotor dynamics, vibrations in pipeline systems, and machine-learning. A basic theoretical overview of the presented topics and some typical use cases for selected open-source packages are given.

Finally, this review article proposes a strategy to increase collaboration among many researchers resulting in highly-related future scientific packages. The purpose of this review is to help new researchers get started in open-source-based signal processing, while those already active can learn the development principles for better and interconnected scientific packages.

---

1. Introduction

Open-source scientific computing has a significant impact on the speed of scientific development. There are several levels of collaboration in open-source development; the open release of the underlying code is only the basic level with limited potential [1]. For a higher level of openness, it is important that the code is developed publicly (including all communication) accepting changes and co-authors; it is also important that decisions are communicated and made publicly [2]. For open-source projects, development and management should ideally be shared between multiple institutions to ensure long-term stability [3]. Free licenses are crucial for open-source-based research; they facilitate the seamless sharing and modification of code, data and other resources, encourage collaboration and accelerate scientific progress. Without the freedom these licenses provide, researchers would face barriers to accessing and building on existing work, hindering innovation and limiting the potential impact of their discoveries.

The Python programming language [4] is easy to learn and used in various fields, including scientific computing, due to its readability, versatility, open license and strong community support. There are several essential Python packages for scientific computing, e.g. NumPy, SciPy, MatPlotLib, and SymPy. NumPy is the main library for array programming with some basic array concepts that are essential for organizing, exploring and analyzing scientific data. NumPy is used in almost every Python library that performs scientific or numerical computation [5]. SciPy builds on NumPy and is used for fundamental algorithms for scientific computing; it supports for example: linear algebra, Fourier transforms, algorithms for numerical optimization and integration, and functions for higher-level data analysis such as signal and image processing [2].

MatPlotLib is a comprehensive library for creating static, animated and interactive visualizations in Python; it is essential for creating high-quality plots for scientific publications [6]. SymPy is a computer-algebra system written in pure Python, enabling one to manipulate and solve mathematical expressions and equations exactly. NumPy, SciPy, MatPlotLib and SymPy are used in approximately 5 million other packages (as seen on the collaborative platform github.com); in addition, each of these three packages has approximately 1.5 thousand unique code contributors. According to pypistats.org, Numpy, Scipy, Matplotlib and SymPy (combined) have more than 15 million daily downloads; the development power and spread of influence is unmatched by any other commercial company.

Although these general-purpose scientific packages constitute a robust computational foundation, the situation within structural dynamics remains comparatively fragmented. Numerous research groups and institutions have independently developed domain-specific packages addressing particular sub-fields. This fragmentation often results in redundancy of effort, heterogeneity of interfaces, and limited interoperability, thereby constraining the ability of researchers to integrate complementary tools. To enhance scientific efficiency and foster sustainable software ecosystems, it is imperative that the structural dynamics community aligns its development practices. Beyond adherence to open-source principles, this alignment requires the establishment of common standards for software design, documentation, and interoperability, thereby enabling seamless interaction among packages and facilitating the broader dissemination and validation of novel methodologies.

This article deals with selected packages that are used at the interface of signal processing and mechanical systems. The packages are discussed in topic-related sections; in each section, a brief overview of the relevant and current scientific literature is given first, followed by the presentation of one or more topic-related Python packages. Sec. 2 discusses general signal-processing methods in the context of the SciPy package. Sec. 3 discusses experimental and operational modal analysis, particularly in relation to the pyEMA, pyOMA, and KOMA packages. Sec. 4 discusses image processing-based experimental modal analysis, focusing on digital image correlation and related methods; packages discussed are pyIDI, pyMRAW, OpenCV, Scikit Image and DICe. Sec. 5 discusses vibration fatigue and the associated packages fatpack, pyfatigue and FLife. Rotordynamics is discussed in Sec. 6 and is related to the open-source package ROSS. Sec. 7 deals with substructuring and transfer path analysis and the implementation of these theories in the pyFBS package. Sec. 8 gives an overview of the machine learning and selected Python packages like sklearn, PyTorch. Sec. 9 deals with vibration control theory and its implementation in the Rattlesnake package. The Sections 10 and 11 are dedicated to two Graphical User Interface (GUI)-based frameworks for structural

dynamics in general and acoustically-induced vibration in pipeline systems, respectively. Finally, Sec. 12 presents the cross-institutional efforts on the SDyPy package, which aims to become the SciPy of structural dynamics by providing a unified, extensible framework that integrates methods and tools across the diverse sub-fields discussed in this article, thereby addressing the current fragmentation and promoting interoperability within the community.

Contents

List of Acronyms

- AI: Artificial Intelligence
- AIV: Acoustically-Induced Vibration
- CFD: Computational Fluid Dynamics
- CLI: Command Line Interface
- CSD: Cross Spectral Density
- DFT: Discrete Fourier Transform
- DIC: Digital Image Correlation
- DOF: Degree-Of-Freedom
- DS: Dynamic Substructuring
- EFDD: Enhanced Frequency Domain Decom-

position

- EMA: Experimental Modal Analysis
- FBS: Frequency-Based Substructuring
- FDD: Frequency Domain Decomposition
- FEM: Finite Element Method
- FETM: Finite Element Transfer-Matrix Method
- FFT: Fast Fourier Transform
- FRF: Frequency Response Function
- FSDD: Frequency-Spatial Domain Decomposition
- GANs: Generative Adversarial Networks
- GPL: Genefral Public License
- GPU: Graphics Processing Unit
- GUI: Graphical User Interface
- ISC: Internet Systems Consortium
- JSON: JavaScript object notation
- LF: Low-Frequency
- LM-FBS: Lagrange Multiplier - Frequency Based Substructuring
- LRF: Low-Reduced Frequency
- LSCF: Least-Squares Frequency Domain technique
- LSFD: Least-Squares Complex Frequency technique
- M-SEMM: modal system equivalent model mixing
- MAC: Modal Assurance Criterion
- MIMO: Multiple-Input-Multiple-Output
- MISO: Multiple-Input-Single-Output
- MIT: Massachusetts Institute of Technology
- MMM: Mobility-Matrix Method
- OMA: Operational Modal Analysis
- PDF: Probability Density Function
- PLA: Polylactic Acid
- pLSCF: Poly-reference Least Squares Frequency Domain
- PoSER: Post Separate Estimation Re-Scaling
- PreGER: Pre-Global Estimation Re-Scaling
- PSD: Power Spectral Density
- RAM: Random Access Memory
- RFC: Rainflow Counting
- ROI: Region of Interest
- ROSS: Rotordynamic Open Source Software
- SCADA: Supervisory Control and Data Acquisition

- SCF: Stress Concentration Factor
- SDOF: Single Degree of Freedom
- SEMM: System Equivalent Model Mixing
- SHM: Structural Health Monitoring
- SIMO: Single-Input-Multiple-Output
- SSIcov: Stochastic Subspace Identification covariance-driven
- SSIdat: Stochastic Subspace Identification data-driven
- SVD: Singular Value Decomposition
- SVT: Singular Vector Transformation
- TMM: Transfer-Matrix Method
- TPA: Transfer Path Analysis
- UPC: Unweighted Principal Component
- VAE: Variational AutoEncoders
- VAP: Virtual Acoustic Prototyping
- VPT: Virtual Point Transformation

2. General signal processing - SciPy.signal

This section briefly introduces the essential signal-processing tools, available in the SciPy package, with basic theoretical background and use-case examples. This section should help researchers entering the field of open-source research to understand the basic concepts of signal processing within the Python ecosystem. Signal-processing methods have a rich history rooted in the need to analyze complex signals, especially prevalent in applications for measuring vibrations. Joseph Fourier laid the foundation in the early 19th century with the development of the Fourier series [7], a mathematical technique for converting signals between the time and frequency domains. This was followed by the Discrete Fourier Transform (DFT), which made it possible to apply Fourier analysis to digital data. In 1965, Cooley and Tukey [8] revolutionized the field with the Fast Fourier Transform (FFT) algorithm, which drastically reduced the computational cost of the DFT and made real-time signal analysis practical. Shortly after, Welch's method emerged [9], offering a robust approach to estimating power spectral density by averaging modified periodograms to reduce variance and provide a clearer spectral estimate. In parallel to these methods, digital-filtering techniques were developed to isolate and analyze specific frequency components within a signal [10]. Although these methods were developed a long time ago and are well established, progress is still being made in this field. Recent publications have focused on new techniques and refinements that improve the precision and applicability of signal processing in modern vibration measurement applications via novel approaches such as variational mode decomposition [11] or piecewise power fitting [12]. Machine learning and Artificial Intelligence (AI)-based methods are also regularly being developed to solve complex signal processing tasks for vibration analy-

sis, utilizing techniques such as deep learning using convolutional neural networks [13, 14] or generative diffusion models [15].

Among Python libraries for signal processing, SciPy, particularly its `scipy.signal` module, stands out [2]. Since its 2001 launch, SciPy has become the *de facto* standard for signal processing and scientific computing, offering FFT, spectral filtering, windowing, and convolution tools. It continues to evolve with enhanced performance, Graphics Processing Unit (GPU) and distributed-computation support, and expanded hardware compatibility as outlined in its roadmap [16].

This section outlines the concepts behind the fundamental signal-processing algorithms, as well as provides short code examples to serve as a reference for their application in structural dynamics.

2.1. Spectral analysis

When dealing with dynamic systems and vibration measurements, one is usually interested in the frequency-domain representation of the observed signals. The Fourier integral transform is used to transform continuous signals between the time- and frequency-domains [17]. For finite discrete signals, typical for vibration measurements, the Fourier integral transform becomes a summation over the N samples, acquired at discrete time steps $n\Delta t$, called the DFT:

$$X_k = \sum_{n=0}^{N-1} x_n \, \mathrm{e}^{-2\pi \mathrm{j} \frac{kn}{N}}, \tag{1}$$

which transforms the discrete time signal x_n, into its frequency-domain representation at discrete frequency steps $k\Delta f = k/(N\Delta t)$ and $\mathrm{j} = \sqrt{-1}$.

The FFT is an efficient algorithm for DFT computation, taking into account the periodicity and symmetry of the DFT (1) [17]. Via the sub-module `SciPy.fft`, SciPy includes extensive tools for various spectral-analysis related operations. The following code example computes the time-frequency transform of a synthetic signal x:

```
from scipy import fft                 # Import SciPy's fft module
X = fft.fft(x) / len(t)               # FFT and normalize the amplitude scpectrum
freq = fft.fftfreq(len(t), d=1/fs)    # Compute the frequency vector
```

The time- and frequency-domain representations of a signal x with two harmonic components at $f_1 = 10$ Hz and $f_2 = 20$ Hz are shown in Fig. 1.

Another useful quantity when analyzing dynamic signals is the Power Spectral Density (PSD), helping to estimate the signal's power distribution across the frequency spectrum. A simple PSD estiamtor is the periodogram [17, 18]:

$$S_{xx}(f) = \frac{1}{T} X^*(f) X(f), \tag{2}$$

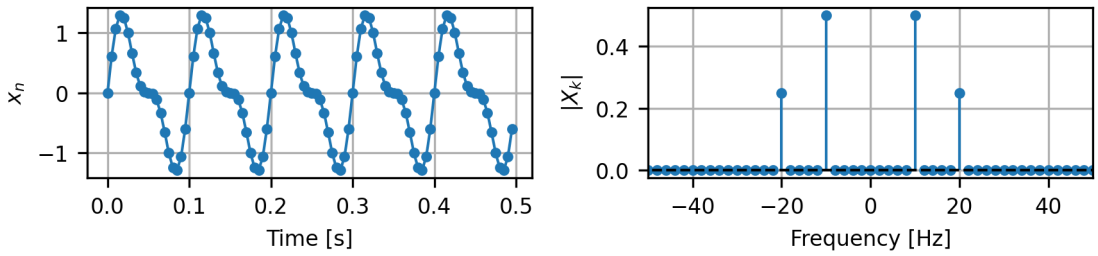


Figure 1: Time- and Frequency-domain representations of a discrete signal.

where $X(f)$ is the Fourier transform of the signal and * the complex conjugate operator.

Similarly, to observe the phase and amplitude relationships between two signals, the Cross Spectral Density (CSD) is used, and can be estimated as:

$$S_{xy}(f) = \frac{1}{T} X^*(f) Y(f). \tag{3}$$

Although Eqs. (2) and (3) yield the correct spectral representations for deterministic signals, and are asymptotically unbiased for stationary ergodic processes, they exhibit high variance when applied to random data. Since this variance does not decrease with increasing observation period, the estimators are considered inconsistent [17]. When dealing with random signals and imperfect measurements in practice, Welch's method is used to improve the estimation of CSD and PSD [9]. It performs signal segmentation and averaging of overlapping segments to reduce variance in the result at the expense of frequency-resolution. The following code example computes the PSD of signal x with added random noise using Welch's method in `scipy.signal.csd`:

```
import numpy as np                                  # import numpy

rng = np.random.default_rng()
x_noise = x + rng.normal(0, 0.5, len(x))            # Generate noisy (Gaussian) signal
freq, Sxx = signal.csd(                             # PSD using Welch's method
  x_noise, x_noise, fs,
  window='boxcar', scaling='density',
  nperseg=100, noverlap=50
  )
```

Fig. 2 shows the signal x with added random noise and its one-sided PSD, computed using Welch's method, which matches that of the original signal well, despite the considerable level of added noise.

Another important tool, often used to counter the effects of spectral leakage when the acquired signal violates the periodicity assumption of the DFT, is the use of windowing [17]. More than 20 different windowing

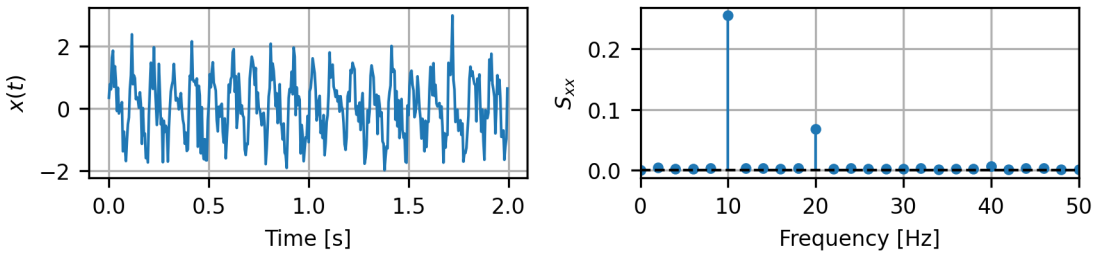


Figure 2: Synthetic signal x with added noise and its PSD, computed using Welch's method.

functions are included in Scipy and can be selected via the `window` parameter in the above example.

2.2. *Filtering*

Spectral-filtering techniques are employed to remove unwanted noise from vibration data, isolating relevant frequency components and improving the clarity and accuracy of measurements. A code example of implementing a digital band-pass filter, removing frequency content significantly below and above the first harmonic component of the noisy signal x, is shown below, with the results illustrated in Fig. 3:

```
from scipy import signal
# Design a band-pass Butterworth filter
sos = signal.butter(N=3, Wn=[7, 13], btype='bandpass', fs=fs, output='sos')

# Apply the digital filter to noisy signal (forward and backward)
x_filtered = signal.sosfiltfilt(sos, x_noise)
```

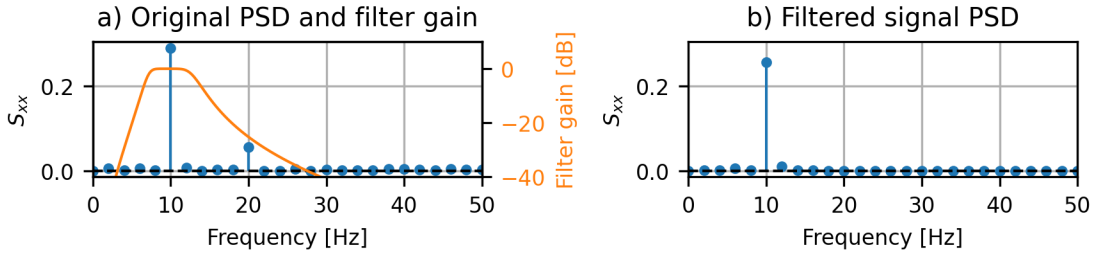


Figure 3: Digital filter design (a) and filtering results (b).

3. Experimental and operational modal analysis

Experimental and operational modal analysis focuses on the estimation of the modal parameters, *i.e.* the natural frequencies, the modal shapes and the damping. The input data for the Experimental Modal Analysis

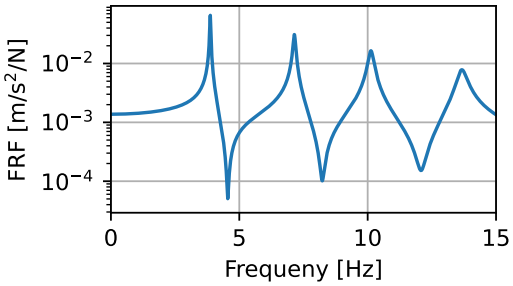


Figure 4: An example of a Frequency Response Function.

(EMA) are the Frequency Response Function (FRF) of different dimensions: Single-Input-Multiple-Output (SIMO), Multiple-Input-Single-Output (MISO), Multiple-Input-Multiple-Output (MIMO). Depending on the measured FRFs, the EMA algorithm differs. Some of the first algorithms are the half-power method, circle fitting and others [19].

Since the Least-Squares Frequency Domain technique (LSCF) [20] and Least-Squares Complex Frequency technique (LSFD) [21] methods have been presented, the combination of the two is a preferred approach for modal parameter identification. When the MIMO FRFs are available, the Poly-reference Least Squares Frequency Domain (pLSCF) [22] algorithm can be used, taking advantage of the redundancy of information contained in the FRFs.

Three packages are discussed in the subsections: SDyPy.EMA, pyOMA2 and KOMA. The first package is used for experimental modal analysis, while the other two are intended for operational modal analysis. While pyOMA2 is a more complete and object-oriented package, KOMA is a simpler and procedural package.

3.1. SDyPy.EMA

This section discusses the EMA sub-package of the SDyPy (Structural Dynamics Python) package. The package is developed at Github (`https://github.com/sdypy/sdypy`) and has an Massachusetts Institute of Technology (MIT) licence which lets people use, copy, modify, and distribute software freely, as long as they include the original license and copyright notice. SDyPy.EMA (can also be installed separately as pyEMA) provides a simple interface for estimating the modal parameters from the given FRF. The package uses a combination of the algorithms LSCF [22] and LSFD [21]. The LSCF algorithm is used to estimate the complex-valued natural frequencies using a least-squares estimation of the rational polynomial function:

$$\hat{H}_k(\omega) = \frac{\sum_{j=0}^{n} \Omega_j(\omega) B_{kj}}{\sum_{j=0}^{n} \Omega_j(\omega) A_j}. \tag{4}$$

In Eq. (4), A_j and B_{kj} are the coefficients to be estimated. The estimation can be performed efficiently using the Fourier transform as described by Guillaume *et al.* [22]. The roots of the polynomial (4) are the complex eigenfrequencies of the system defined after conversion from the z-domain:

$$\lambda_r = -\zeta_r \omega_r \pm \mathrm{j}\,\omega_r \sqrt{1-\zeta_r^2}, \tag{5}$$

where ζ_r and ω_r are the damping ratio and the radial natural frequency corresponding to the r-th mode.

Once the natural frequencies and damping of the system are identified, the LSFD method [21] is used to identify the modal shapes. The leading equation of the LSFD method is:

$$\hat{H}_j(\omega) = \sum_{r=1}^{N_m} \left(\frac{A_{r,j}}{\mathrm{j}\,\omega - \lambda_r} + \frac{A_{r,j}^*}{\mathrm{j}\,\omega - \lambda_r^*} \right) - \frac{A_L}{\omega^2} + A_U, \tag{6}$$

where $\hat{H}_j(\omega)$ is the measured FRF at location j at frequency ω. $A_{r,j}$ is the complex-valued r-th modal constant at location j, λ_r is the r-th complex-valued eigenfrequency and A_L and A_U are the lower and upper residuals, respectively. The residuals are used to simulate the effect of the modes that lie outside the range of ω. * denotes the complex conjugate. The modal constants $A_{r,j}$ are estimated by the least-squares method at all available locations j and the frequency range of interest ($\omega \in [\omega_{\text{start}}, \omega_{\text{stop}}]$).

SDyPy.EMA uses the efficient implementation of LSCF and LSFD. Using the package is simple. First, the object of `Model` is created:

```
from SDyPy import EMA

model = EMA.Model(frf_matrix, frequency_array, lower=50,
    upper=10000, pol_order_high=60)
```

The FRF matrix is passed to the class `Model`. The FRF matrix has the shape ($N_{\text{LOC}} \times N_{\text{freq}}$), where N_{LOC} is the number of locations at which the FRF is known and N_{freq} is the number of frequency points at which the FRF is measured. `frequency_array` defines the N_{freq} frequencies at which the FRF is known. The arguments `lower` and `upper` define the lower and upper limits of the frequency range that is taken into account in the estimation. The argument `pol_order_high` is the polynomial order up to which the order is increased.

Next, the poles are computed using LSCF method. In SDyPy.EMA, this is done by calling:

```
model.get_poles()
```

Once the poles have been identified, the stable poles can either be selected using the stability card:

```
model.select_poles()
```

or by making initial assumptions about where the poles are located:

```
model.select_closest_poles([300, 350])
```

The final step in the EMA process is the identification of the modal constants using the LSFD algorithm:

```
model.get_constants()
```

The modal parameters are then accessible via:

```
model.nat_freq # natural frequencies
model.nat_xi   # damping coefficients
model.A        # modal constants
```

For more information, see the documentation `https://sdypy-ema.readthedocs.io/en/latest/`

3.2. pyOMA and pyOMA2

pyOma2 is a significantly-updated version of the pyOMA module [23] specifically designed to perform Operational Modal Analysis (OMA). pyOMA2 is developed on Github (`https://github.com/dagghe/pyOMA2`) and uses the MIT license. pyOMA2 supports the following algorithms: Frequency Domain Decomposition (FDD) [24], Enhanced Frequency Domain Decomposition (EFDD) [25], Frequency-Spatial Domain Decomposition (FSDD) [26], and both the Stochastic Subspace Identification covariance-driven (SSIcov) and Stochastic Subspace Identification data-driven (SSIdat)) [27, 28, 29], as well as the pLSCF technique, also known as Polymax [22, 30]. In addition, it supports the analysis of single and multi-setup measurements, including the processing of multiple acquisitions with a mixture of reference and roving sensors. The multi-setup analyzes can be performed with the so-called Post Separate Estimation Re-Scaling (PoSER) approach as well as with the so-called Pre-Global Estimation Re-Scaling (PreGER) approach [31, 32, 33, 34]. The module now includes interactive plots to select the modes for extraction directly from the plots generated by the algorithm. Users can also define the geometry of the tested structures, which facilitates the visualization of the mode shapes. For the SSI algorithms it is also possible to get the uncertainty bounds of the identified modal properties [33].
The following summarizes the theoretical background for the algorithms included in the module.

Stochastic Subspace Identification (SSI). The state-space representation of the output-only vibration-based structural monitoring leads to the following discrete-time model:

$$\begin{aligned} \mathbf{x}_{r+1} &= \mathbf{A}\,\mathbf{x}_r + \mathbf{v}_r \\ \mathbf{y}_r &= \mathbf{C}\,\mathbf{x}_r + \mathbf{w}_r \end{aligned}, \tag{7}$$

where $\mathbf{x}_r \in \mathbb{R}^n$, $\mathbf{y}_r \in \mathbb{R}^l$, $\mathbf{A} \in \mathbb{R}^{n\times n}$ and $\mathbf{C} \in \mathbb{R}^{l\times n}$ are the state vector, the output vector, the state transition matrix and the observation matrix, for the r-th sample, respectively. In addition, l is the number of monitored degrees of freedom (DOFs) and n is the order of the system. The excitation v_r is the process noise and w_r is the measurement noise [27].

Numerous algorithms for stochastic subspace identification are presented in the literature [27, 28, 29], all of which differ in how they construct the subspace matrix $\mathbf{H}_{p+1,q}$ from the data. In the following, we stick to the notation shown in [29, 33], as it is also used in pyOMA2. Given parameters p and q such that $p\,l \geq q\,l \geq n$, a matrix $\mathbf{H}_{p+1,q} \in \mathbb{R}^{(p+1)\,l\times q\,l}$ is constructed from the output data according to the selected subspace algorithm. Let $N+p+q$ represent the total number of available samples, then the data matrices are defined as follows:

$$\mathbf{Y}^+ = \frac{1}{\sqrt{N}}\begin{bmatrix} y_{q+1} & y_{q+2} & \cdots & y_{N+q} \\ y_{q+2} & y_{q+3} & \cdots & y_{N+q+1} \\ \vdots & \vdots & \vdots & \vdots \\ y_{q+p+1} & y_{q+p+2} & \cdots & y_{N+p+q} \end{bmatrix}, \quad \mathbf{Y}^- = \frac{1}{\sqrt{N}}\begin{bmatrix} y_{q} & y_{q+1} & \cdots & y_{N+q-1} \\ y_{q-1} & y_{q} & \cdots & y_{N+q-2} \\ \vdots & \vdots & \vdots & \vdots \\ y_{1} & y_{2} & \cdots & y_{N} \end{bmatrix}. \tag{8}$$

For covariance-driven SSI, the block Hankel matrix can be calculated from the empirical correlations, $R_{b_i} = \frac{1}{N(i)}\sum_{k=1}^{N} y_k y^T$, or as shown in [35, 28] as:

$$\widehat{\mathbf{H}}^{\text{cov}} = \mathbf{Y}^+ \left(\mathbf{Y}^-\right)^T. \tag{9}$$

For data-driven SSI with the Unweighted Principal Component (UPC) algorithm, the block Hankel matrix is defined as follows [27]:

$$\widehat{\mathbf{H}}^{\text{dat}} = \mathbf{Y}^+ \left(\mathbf{Y}^-\right)^T \left(\mathbf{Y}^- \left(\mathbf{Y}^-\right)^T\right)^{\dagger} \mathbf{Y}^-, \tag{10}$$

where the symbol † denotes the Moore-Penrose pseudo-inverse. For all subspace-identification algorithms the subspace matrix $\mathbf{H}_{p+1,q}$ can be decomposed into the extended observability, $\mathscr{O}_{p+1}$, and extended controllability matrices, $\mathscr{Z}_q$:

$$\mathbf{H}_{p+1,q} = \mathscr{O}_{p+1}\mathscr{Z}_q, \tag{11}$$

where:

$$\mathscr{O}_{p+1} = \begin{bmatrix} \mathbf{C} \\ \mathbf{CA} \\ \vdots \\ \mathbf{CA}^p \end{bmatrix}, \tag{12}$$

while $\mathscr{Z}_q$ depend on the selected subspace algorithm. The observation matrix C is then found in the first block-row of the observability matrix, while the state transition matrix $\mathbf{A}$ is obtained from the shift invariance property of the observability matrix.

$$\mathscr{O}_{p+1}^{\uparrow}\mathbf{A} = \mathscr{O}_{p+1}^{\downarrow}, \text{ where } \mathscr{O}_{p+1}^{\uparrow} = \begin{bmatrix} \mathbf{C} \\ \mathbf{CA} \\ \vdots \\ \mathbf{CA}^{p-1} \end{bmatrix}, \quad \mathscr{O}_{p+1}^{\downarrow} = \begin{bmatrix} \mathbf{CA} \\ \mathbf{CA}^{2} \\ \vdots \\ \mathbf{CA}^{p} \end{bmatrix}. \tag{13}$$

The modal parameters of the system (*i.e.* natural frequencies, damping ratios and mode shapes) can thus be estimated from the identified system matrices A and C. First, the eigenvalue decomposition of A leads to the diagonal matrix Λ of the discrete-time system poles λ_u and the corresponding right eigenvectors ψ_u:

$$\mathbf{A} = \mathbf{\Psi}\,\mathbf{\Lambda}\,\mathbf{\Psi}^{-1}, \quad \mathbf{A}\psi_u = \boldsymbol{\lambda}_u\psi_u, \tag{14}$$

with $1 \leq u \leq m$, where m is the total number of eigenvalues of interest. The undamped natural frequencies f_u and the damping ratios ξ_u (in %) are finally determined as follows:

$$\lambda_{c,u} = \frac{\ln(\lambda_u)}{\Delta t}, \quad f_u = \frac{|\lambda_{c,u}|}{2\pi}, \quad \xi_u = -100\frac{\mathfrak{Re}(\lambda_{c,u})}{|\lambda_{c,u}|}, \tag{15}$$

where $\lambda_{c,u}$ are the continuous-time system poles, Δt is the sampling interval, $|\cdot|$ denotes the complex modulus and $\mathfrak{Re}(\lambda_{c,u})$ is the real part of $\lambda_{c,u}$. The real part of the eigenvectors ψ_u instead leads to the experimental mode shapes ϕ_u:

$$\phi_u = \mathfrak{Re}(C\psi_u). \tag{16}$$

FDD. The theory of the FDD technique is based on the formula of the input and output PSD relationship for a stochastic process [36]:

$$\mathbf{G}_{yy}(\omega) = \mathbf{H}(\omega)\,\mathbf{G}_{xx}(\omega)\,\mathbf{H}(\omega)^{\mathrm{H}}, \tag{17}$$

where $\mathbf{G}_{xx}(\omega)$, $\mathbf{G}_{yy}(\omega)$ are input and output PSD matrices, respectively. $\mathbf{H}(\omega)$ is the Frequency Response Function matrix, which can be expressed as a partial-fractional form over poles and residuals, as already shown in Eq. (6). The $^{\mathrm{H}}$ operator denotes the conjugate transpose. The first step involves estimating the PSD matrix [24], commonly achieved in practical applications using the *Welch* method [9, 36]. The estimation of the SD matrix in pyOMA2 is facilitated by wrapper functions that utilize the SciPy [2] `signal.csd()` function, as introduced in Sec. 2.1. The estimate of the output PSD $\hat{\mathbf{G}}_{yy}(\omega)$, which is known at discrete

frequencies $\omega = \omega_k$, is then decomposed by the Singular Value Decomposition (SVD) of the matrix:

$$\hat{\mathbf{G}}_{yy}(\omega_k) = \mathbf{U}_k \mathbf{S}_k \mathbf{V}_k^{\mathrm{H}} = \mathbf{U}_k \mathbf{S}_k \mathbf{U}_k^{\mathrm{H}}, \tag{18}$$

where $\mathbf{U}_k$ and $\mathbf{V}_k$ are the unitary matrices containing the left and right singular vectors, and $\mathbf{S}_k$ is the matrix of singular values. For a Hermitian and positive-definite matrix such as the PSD matrix $\mathbf{V}_k = \mathbf{U}_k$, a comparison between Eq. (17) and Eq. (18) suggests a relationship between the singular vectors and the mode shapes. When only one mode dominates at frequency ω_k, this relationship becomes one-to-one, as the PSD matrix approximates a rank-one matrix. Furthermore, the singular values are associated with the modal responses and can be used to define the spectra of equivalent Single Degree of Freedom (SDOF) systems. Assuming that only one mode is dominant at frequency ω_k, the PSD matrix approximates a matrix of rank one, and in such cases the first singular vector $\mathbf{u}_1$ is an estimate of the mode shape $\hat{\phi} = \mathbf{u}_1$.

EFDD. In the enhanced version of this method, the EFDD [25], the comparison of the mode-shape estimation at frequency lines around the peak leads to the identification of the singular values whose singular vectors have a correlation higher than a user-defined threshold, the so-called Modal Assurance Criterion (MAC) Rejection Level. Such singular values define the equivalent SDOF PSD function and this is then used to obtain the modal damping ratio and to obtain estimates of the natural frequency independent of the frequency resolution of the spectra [31, 32].

Poly-reference LSCF method. The poly-reference LSCF method (Polymax) is an extension of the LSCF algorithm introduced in Sec. 3.1. In contrast to the common-denominator model adopted by the LSCF estimator the poly-reference LSCF uses a right matrix fraction description [22, 21, 30], which implies that in addition to the pole information (i.e. frequency and damping), also modal participation factors are estimated. Once the poles and participation factors have been determined, the LSFD estimator can be used to directly estimate the mode shapes avoiding the SVD required when using the LSCF estimator. The fundamental difference between the EMA and OMA approaches lies in the fact that for EMA the FRFs are used while in OMA the output PSD matrix.

pyOMA2 package organization. The package is structured into three class levels: 1. setup classes: specifying a data array and sampling frequency for a single-setup scenario or a list of data arrays and their respective sampling frequencies for a multi-setup scenario. 2. Algorithm classes. 3. Support classes: auxiliary components for the first two levels (*e.g.*: saving geometric data, animating mode shapes). Fig. 5 shows the schematic of the module structure and the inheritance relationships between the classes.

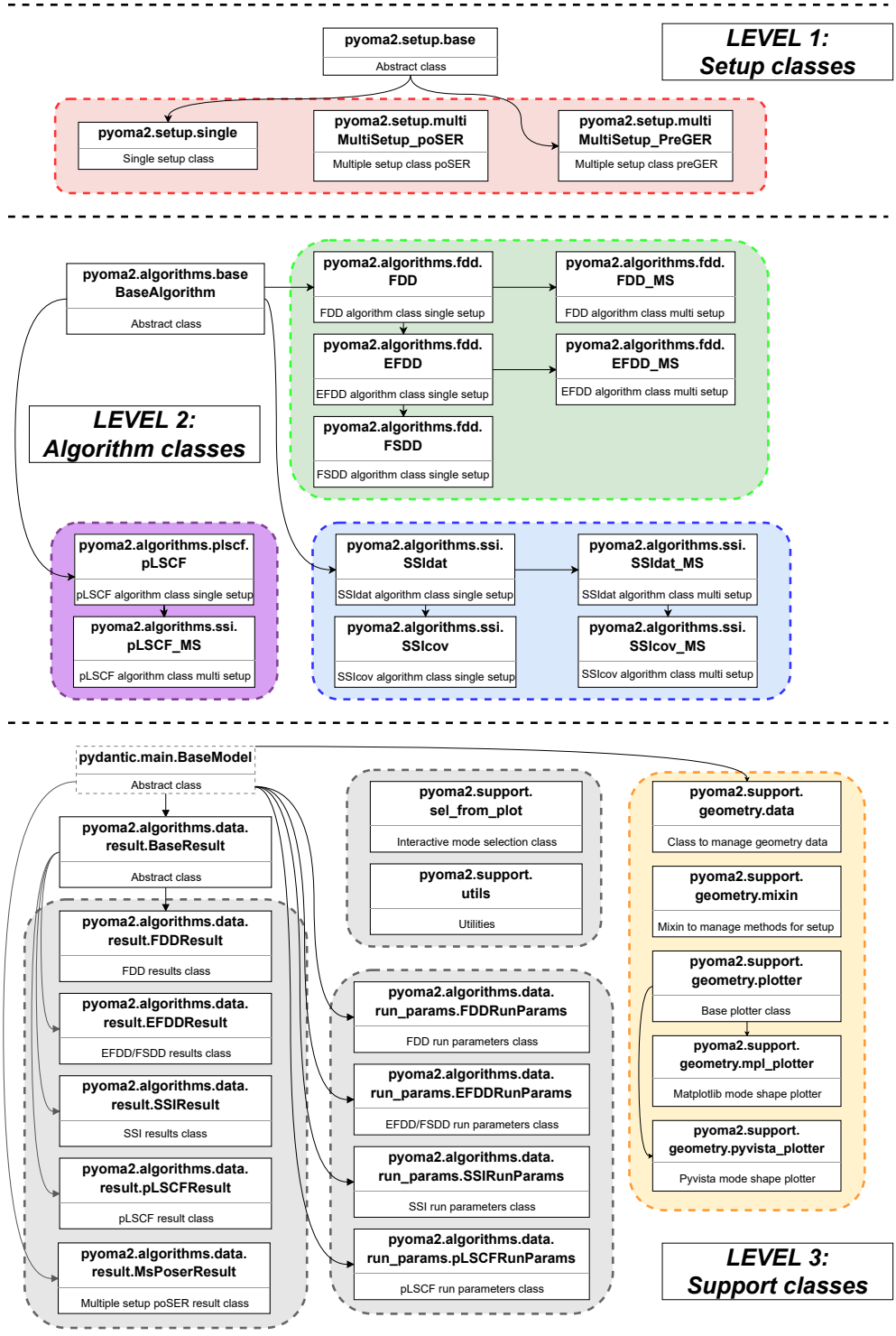


Figure 5: Schematic structure of the pyOMA2 with inheritance between classes.

Example. Here the use of the module for a single setup scenario is demonstrated: The process starts with importing the necessary modules.

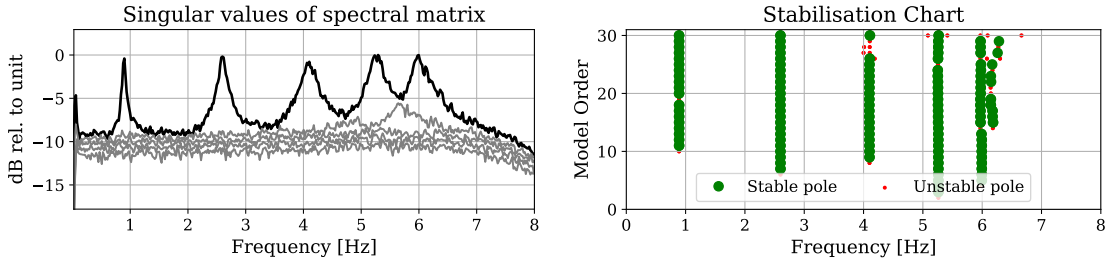
```
from pyoma2.setup import SingleSetup
from pyoma2.algorithms import SSIcov, FDD

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

Once the dataset has been imported and assigned to a variable, the next step is to create an instance of the `SingleSetup` class with the dataset and sampling frequency as parameters. This instance provides access to methods for evaluating the quality of the dataset and for performing preprocessing operations such as `plot_ch_info()` and `filter_data()`.

```
fdd = FDD(name="FDD", nxseg=1024)                    # Initialise the algorithms
ssicov = SSIcov(name="SSIcov", br=30, ordmax=30) # Initialise the algorithms
ss.add_algorithms(fdd, ssicov)                       # Add algorithms to the single
                                                  setup class
ss.run_all()                                         # Run all or run by name
```

The users then instantiate the algorithm classes of their choice, such as `FDD` and `SSIcov`. These algorithms are added to the setup class using the `add_algorithms()` method. The algorithms can either be executed individually with the method `run_by_name()` or collectively with `run_all()`. After the algorithms have been executed, methods such as `plot_CMIF()` and `plot_stab()` are used to visualize the results.



(a) (b)

Figure 6: (a) Plot of the singular values of the PSD matrix (b) Stabilisation chart for increasing model order.

These diagrams can be accessed interactively via the `mpe_from_plot()` method. The method allows users to extract the desired modes directly from the plots.

```
fig, ax = fdd.plot_CMIF(freqlim=(0,8))
fig1, ax1 = ssicov.plot_stab(freqlim=(0,8), hide_poles=False)
ss.mpe_from_plot("FDD")       # Select modes to extract from plots
ss.mpe_from_plot("SSIcov")    # Select modes to extract from plots
fdd.result.Fn                 # access results
```

```
ssicov.result.Fn                # access results
```

Alternatively, the modal properties can also be extracted manually using the `mpe()` method. Once the modes have been extracted, the results can be accessed via the `Result` class of the corresponding algorithms. Further information can be found in the documentation `https://pyoma.readthedocs.io/en/main/`.

3.3. KOMA

The Python package KOMA [37] offers much of the same functionality as pyOMA2; however, it is easier to use and less complete. In contrast to the object structure of pyOMA2, the most important calculations are performed in a procedural way. This feature could speed up the integration into a custom setup, but would require the use of other scientific Python libraries such as *scipy* for the required processing and more custom code. So far, only single analyzes are supported, so multi-setup analyzes would usually require custom morphing of mode shapes from different analyzes. The source code of KOMA is available on GitHub (`https://github.com/knutankv/koma`) and is released under MIT license.

The OMA algorithms SSI-cov and FDD, which have already been described in the Sec. 3.2, are supported. Peak picking in FDD and pole selection in SSI can be performed by interactive plots based on `matplotlib` , while visualization of the mode shape (animation and plotting) relies on the Python package `pyvista`. Automatic OMA is implemented as described in [38]; it is based on the clustering algorithm HDBSCAN (Hierarchical Density-Based Clustering Algorithm with Noise) introduced in [39] and its Python package `hdbscan`.

Assuming that a data matrix `data` with a sampling rate `fs` is given, the workflow of the SSI-cov analysis would typically look as follows:

```
from koma.oma import covssi, find_stable_poles

# Defining SSI parameters
i = 50  # block rows
orders_input = np.arange(2, 60+2, 2)

# Defining stabilization parameters
stabcrit = {'freq':0.05, 'damping': 0.2, 'mac': 0.2}
s = 2   # number of consequtive stable poles

# Running SSI-cov and filtering of poles based on stabcrit
lambd, phi, orders = covssi(data, fs, i, orders_input)
lambd, phi, orders, __ = find_stable_poles(lambd, phi, orders, s,
                                           stabcrit=stabcrit,
                                           indicator='freq')
```

Then the modes can be selected from the poles, which are represented as eigenvalues in `lambd` and eigenvectors in `phi`, either by defining and executing an interactive stabilization plot:

```
from koma.plot import StabPlotter
stab_plotter = StabPlotter(lambd, orders, phi=phi, freq_unit='hz')
fig = stab_plotter.get_fig()
```

or by automatic selection:

```
from koma.clustering import PoleClusterer, group_clusters
pole_clusterer = PoleClusterer(lambd, phi, orders,
                               min_cluster_size=15, min_samples=15)

pole_clusterer.cluster()
outputs = pole_clusterer.postprocess()
xi, omega_n, phi, orders, ixs, probs = group_clusters(*outputs)
```

For more information and examples of usage, see the documentation at `https://knutankv.github.io/koma/`.

4. High-speed Camera Based Experimental Modal Analysis

The identification of modal parameters of vibrating structures using optical methods such as high-speed cameras has undergone significant development in the last decade [40, 41]. The identification process is based on displacements in captured digital images, and successful displacement identification usually requires the application of (speckle) patterns on the surface of the vibrating object. Pattern matching [42] is used in most image-based displacement measurement applications. However, for vibration measurements, where the motion is usually far below the pixel range, the simplified optical flow [43] approach is usually faster and offers the potential for higher accuracy [44].

For coordinates (x, y), the optical flow method identifies the displacement Δx, Δy from the changes in light intensity between two time steps $I(x, y, t)$ and $I(x, y, t + \Delta t)$ [41]:

$$I'_{\mathrm{x}} \Delta x + I'_{\mathrm{y}} \Delta y = I(x, y, t) - I(x, y, t + \Delta t)\,, \tag{19}$$

where $I'_{\mathrm{x,y}}$ is a spatial gradient in the x and y directions. Two important assumptions are made: 1) the light intensity and the observed pattern must be constant during the movement, and 2) there is a linear relationship between the change in pixel intensity and the displacement, which usually limits the applicability of the method to sub-pixel displacements.

In general, due to the aperture problem, the identification of displacements cannot be based on a single pixel and surrounding pixels are needed [45]. Complex image scenes may require smoothing to linearize the spatial gradient [43]. When certain patterns are applied to the measured surface that produce a high spatial

gradient only in the direction of motion, the optical flow Eq. (19) can be simplified for use with only one pixel [41, 46]:

$$q(x, y, t) = \frac{I_0(x, y) - I(x, y, t + \Delta t)}{I_0'}, \tag{20}$$

where $I_0(x, y)$ is the reference image and I_0' is the spatial gradient in the direction of motion.

The concept of pattern matching was first developed by Lucas and Kanade [42]. It is a method for identifying local displacement in digital video based on minimizing the difference in the relatively-small Region of Interest (ROI) between two consecutive images. Under the same assumptions as for optical flow, the following relationship can be established between the pixels within the ROI for the time step Δt to identify the displacement [47]:

$$\begin{bmatrix} \Delta x \\ \Delta y \end{bmatrix} = - \begin{bmatrix} \sum I_{\mathrm{x}}' I_{\mathrm{x}}' & \sum I_{\mathrm{x}}' I_{\mathrm{y}}' \\ \sum I_{\mathrm{x}}' I_{\mathrm{y}}' & \sum I_{\mathrm{y}}' I_{\mathrm{y}}' \end{bmatrix}^{-1} \begin{Bmatrix} \sum I_{\mathrm{x}}' \\ \sum I_{\mathrm{y}}' \end{Bmatrix} \big(I_0(x, y) - I(x, y, t + \Delta t) \big), \tag{21}$$

where the summations are performed over all pixels in the ROI. This method is further expanded and improved in [48]; it serves as the basis for Digital Image Correlation (DIC), which is the most commonly-used method for identifying displacements [49].

The development of new methods and the improvement of current methods for measuring vibrations from video recordings is still an important research topic. It is attractive that optical methods using high-speed cameras allow non-contact, full-field displacement measurements with relative ease [40]; they are used in many areas of structural dynamics and more recently in vibroacoustics [50], where they have been successfully used to reconstruct the sound radiation field [51]. Some examples of full-field measurements [52] when the displacements are below the resolution of the camera sensor [53] are: the identification of elastic properties of soft tissues [54], crack detection using digital image correlation with the support of deep learning [55], and the measurement of long-span bridges [56]. Both methods, simplified optical flow and DIC, continue to be actively developed and improved. Optical flow-based EMA can be significantly improved with the hybrid method [44] where the global parameters (natural frequencies and damping) are identified by a high dynamic range sensor and the mode shapes are identified by the camera [47]. Recent advances in the optical flow method have been made in modal identification via the development of an adaptive spatial filtering algorithm [57] and in the identification of nonlinearities in the response of civil structures due to dynamic loading during catastrophic events [58]. A noise-resistant optical flow method for vibration measurement is developed by [59] based on the combination of multiple intensity signals within an ROI. A deep learning-assisted optical flow with Bayesian analysis for real-time displacement identification is developed

by [60]. The interpolation of the subpixel gray values in DIC was adjusted in [61] using the skin patterns to identify displacements. The interpolation scheme was also investigated by [62] to improve the DIC results. The influence of image-compression algorithms on the quality of speckle patterns for displacement identification was investigated by [63]. Recent advances in displacement identification include the development of Directional Digital Image Correlation (D-DIC), which solves the aperture problem of conventional DIC by assuming localized, unidirectional motion, allowing for more trackable localization and improved modal identification for structures without speckle patterns[64]. Such lively research activity is the main motive for open-source software packages, which are intended to serve as a platform for other researchers. Video processing methods can be difficult to implement, and making the basic methods available as open-source software libraries facilitates research into the methods and provides a starting point for researchers to learn the methods from scratch or to have a verifiable tool that can serve as a benchmark for their particular closed solutions [40].

4.1. pyIDI

Python Image Displacement Identification (pyIDI) is an open-source package developed by the structural dynamics community. The package was developed with the goal of providing an open-source displacement identification from high-speed video recordings, primarily aimed at researchers working on computer vision methods in structural dynamics. The source code is available on GitHub (`https://github.com/ladisk/pyidi`) and published under MIT license. Two main methods for identifying displacements are included, one is the simplified optical flow [41] and the second is based on the Lucas-Kanade [42] pattern-matching algorithm. The package is modular so that other researchers can contribute to it. Basic usage requires the creation of the instance in which the video file is deployed:

```
from pyidi import VideoReader
video = VideoReader(input_file='video.ext')
```

where `ext` can be one of the various supported media formats, *e.g.* `.cih`, `.cihx`, `.png`, `.avi` or `numpy.ndarray`. The next step is to select the method of displacement identification. This is done by importing the method class and creating an instance:

```
from pyidi import LucasKanade, SimplifiedOpticalFlow
idi = LucasKanade(video)
```

The next step is to select the points of interest for tracking, which can be done manually using an array of (y,x) pixel coordinates:

```
points = np.array([ [1, 2],
                    [1, 5],
                    [2, 10] ])
```

where the first column indicates indices along `axis 0` and the second column indices along `axis 1`. The points are then passed to the created method object:

```
idi.set_points(points)
```

The created method object (`idi`) can be configured:

```
idi.configure(*args, **kwargs)
```

The available arguments are documented in the `configure` method. To identify displacements, the `get_displacements` method is called:

```
displacements = idi.get_displacements()
```

The displacements identified at the selected points are returned as `numpy.ndarray` objects with the shape $(n_{points}, n_{images}, 2)$, where the last axis of the array contains the displacements of the selected points in the y (`axis=0`) and x (`axis=1`) directions.

4.2. speckle_pattern

In order for the methods defined by the Eq. (20) and (21) to work correctly, the surface of the observed object must have special characteristics. Depending on the identification method, either strong spatial gradients in the direction of motion are required or the object must have high-contrast features that are unique on small subsets. A simple way to achieve this is to print a specially-designed pattern on self-adhesive paper and affix it on the observed surface. To design a special pattern and customize the shape and quality of the pattern, the open-source software package `speckle-pattern` can be used. Its source code is available on GitHub (`https://github.com/ladisk/speckle_pattern`) and published under MIT license.

The simplified optical-flow method requires patterns that have a high spatial gradient in the direction of movement. This is usually a zebra pattern, where the lines are perpendicular to the direction of movement. To create such a pattern, the function `generate_lines` can be used:

```
from speckle_pattern import generate_lines

image_height = 50           # mm
image_width = 100           # mm
line_width = 5              # mm
dpi = 200                   # print resolution
orientation = 'horizontal'  # horizontal or vertical
```

```
lines_image = generate_lines(image_height, image_width, dpi,
    orientation=orientation, line_width=line_width,
    path='lines.jpg')
```

If the movement is to be measured in two directions, then a high gradient in two perpendicular directions is required. The checkerboard pattern is well suited for this purpose. Based on the previous example, the following code can be used to generate an image of a checkerboard pattern:

```
from speckle_pattern import generate_checkerboard

checkerboard_image = generate_checkerboard(image_height,
    image_width, dpi, line_width=line_width,
    path='checkerboard.jpg')
```

The methods based on pattern matching track a small subset (ROI) of the object. It is important that this small subset is unique on the object in order to be distinguishable for the algorithm. Usually a speckle pattern with random order, size and density is used for this purpose. Such patterns can be generated with the following code:

```
from speckle_pattern import generate_and_save

speckle_diameter = 3    # mm
speckle_image = generate_and_save(image_height, image_width,
    dpi, speckle_diameter, path='speckle.jpg')
```

Further adjustments to the speckles are possible; this is achieved with additional arguments of the function `generate_and_save()`, which are displayed here with their default values:

```
size_randomness = 0.5       # speckle size variety
position_randomness = 0.5   # speckle position variety
speckle_blur = 1            # Gaussian smoothing kernel size
grid_step = 1.2             # approximate grid step,
                            # in terms of speckle diameter 'D'
```

Two speckle-pattern images generated with different parameters are shown in Fig. 7.

4.3. pyMRAW

Commercially-available high-speed cameras often store video data in proprietary file formats, which must first be decoded accordingly in order to obtain the pixel intensities. When using Python's standard numerical tools, the pixel intensities are expected to be stored in array format, usually with three dimensions: Number of images, image height, image width. However, the data recorded by the high-speed camera can amount to hundreds of gigabytes. This data should be accessed effectively by reading it from the storage device directly whenever possible, rather than storing it entirely in the computer's working memory (Random Access Memory (RAM)).

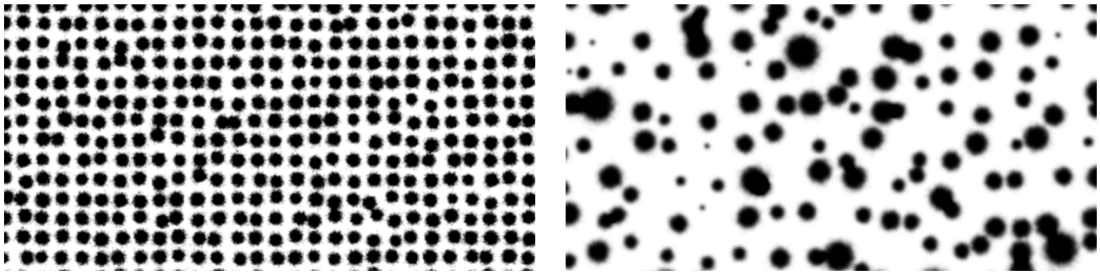


Figure 7: Two examples of custom speckle-pattern images, generated using `speckle-pattern` package.

Photron is one of the leading manufacturers of high-speed and ultra-high-speed cameras, which are frequently used in research and engineering applications. The `mraw` file format is used to store raw video data recorded with Photron high-speed cameras. `mraw` files are delivered together with a `cih` or a `cihx` metadata files that contain the video-recording information such as frame size, bit depth, frame rate, etc. The `mraw` file itself stores the recorded image data in a binary format. The pyMRAW package, with source code available on GitHub (`https://github.com/ladisk/pyMRAW`) under the permissive MIT license, was written to efficiently deal with `mraw` image data. PyMRAW uses Numpy's memory mapping functionality (`numpy.memmap`) to create a memory map to the video data stored in the `mraw` binary file, so that large amounts of image data do not need to be loaded into the computer's memory. To get the video data from the `mraw` file, the user provides the `load_video()` function with the path to the `cih/cihx` metadata file, stored next to the `mraw` file:

```
import pyMRAW
video, info = pyMRAW.load_video('data/beam.cihx')
```

For example, in the case of 16-bit image data, the object `video` is a memory-mapped array of data type `uint16` and shape $(n_{images}, \texttt{height}, \texttt{width})$, and `info` is a dictionary of metadata read from the file `cihx` that can be easily displayed and manipulated using standard Python dictionary operations.

The memory-mapped `video` array behaves like an ordinary `numpy.ndarray` object and is accepted as input in all workflows where a Numpy array is accepted. For example, its frames can be displayed using Python's `matplotlib` plotting library:

```
import matplotlib.pyplot as plt

first_image = video[0]
plt.figure()
plt.imshow(first_image, cmap='gray')
plt.show()
```

which produces the image shown in Fig. 8.

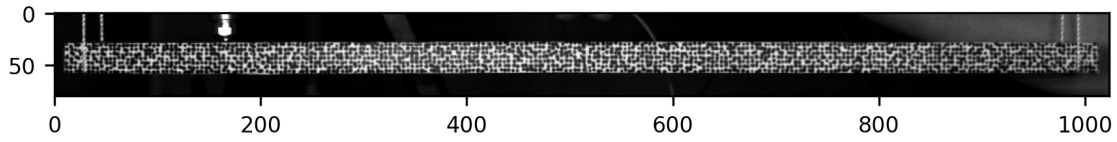


Figure 8: An image read using the pyMRAW package.

PyMRAW also includes the basic functionality to save the custom image data in the `mraw` format using `pyMRAW.save_mraw()`:

```
random_images = np.random.randint(low=0, high=255,
        size=(10, 64, 64), dtype=int)

info_dict = {'Record Rate(fps)': 25,
             'Total Frame': random_images.shape[0],
             'Image Width': random_images.shape[2],
             'Image Height': random_images.shape[1],
             'Comment Text': 'Randomly generated images.',
             }

mraw_file, cih_file = pyMRAW.save_mraw(
    images=random_images,
    save_path='random.mraw',
    bit_depth=8,
    info_dict=info_dict
    )
```

The expected image format is an appropriately shaped, (`n`$_{\texttt{images}}$`, height, width`), integer array. The argument `info_dict` is a dictionary of metadata that one wants to assign to your video. It uses default values for the most important metadata entries required to display the `mraw` video with Photron's PFV software if they are not provided via `info_dict`.

4.4. OpenCV-Python

The open-source Computer Vision Library is an open-source library containing more than 2500 algorithms for various operations on digital images, as well as some common machine-learning techniques for image processing. It was developed to support commercial products dealing with computer vision and the code is released under the Apache 2 license. The library was originally developed in C++ and OpenCV-Python is a Python wrapper for the original implementation. The library's algorithms are intended for real-time use, which means that the algorithms are optimized for fast execution. Common image-processing actions, such as edge detection, image filtering, affine transformation, etc., can be very useful for structural dynamics

applications. Feature detection and matching methods, implemented in OpenCV, can be used for tasks such as extrinsic camera parameter determination and 3D geometry triangulation. ArUco marker detection is another useful feature when implementing algorithms for precise positioning based on digital images with OpenCV [65]. The library also implements several efficient object tracking and optical flow detection algorithms that can be useful in certain motion-detection workflows.
The OpenCV-Python package source code is available on Github (`https://github.com/opencv/opencv-python`) under the MIT license.

4.5. Scikit Image

This is an open-source image processing library developed by the Python community. The library is released for the Python programming language under the BSD open-source License and is well suited for use in research and teaching. Due to permissive licensing and robust implementation of many standard image processing operations, it is also suitable for industrial applications [66]. Compared to OpenCV-Python, Scikit Image is less suitable for real-time machine vision applications, but offers a more Pythonic approach to image processing algorithms, many of which can be useful for structural dynamics applications.
The Scikit Image package is developed on Github (`https://github.com/scikit-image/scikit-image`) and uses the BSD-3-Clause license.

4.6. Digital Image Correlation Engine (DICe)

DICe is a cross-platform tool developed by Sandia National Laboratory for the complete identification of displacements and strains [67]. It is written in C/C++ and can be used freely under the specific license conditions. It is based on the Lucas-Kanade [42] pattern-matching method, with many improvements to the original method implemented [48, 49]. Some advanced features that have been implemented are: arbitrary shape of ROI, convolution-based interpolation functions, two optimization approaches for pattern matching: gradient-based and simplex-based, etc. The tool can be used as a standalone program (via GUI or Command Line Interface (CLI)) or as a plug-in in larger applications.
The DICe package is developed on Github (`https://github.com/dicengine/dice`).

5. Vibration fatigue

Material fatigue is the process by which a crack develops and grows via repeated loading of the material. Above a certain crack length, the remaining cross-section can no longer withstand the loads and the entire

specimen fails. When planning and assessing structural fatigue, there are typically three different approaches to determine the state of material fatigue *i. e.* stress-life, strain-life, and fracture mechanics.

The focus here is on the stress-life approach as it covers the high-cycle fatigue which is common in vibration fatigue. The stress-life method is a phenomenological approach that builds on the concepts of fatigue resistance and damage.

Fatigue strength is the resistance of the structure to fatigue cracks. It is usually determined by cyclic loading of a specimen with a constant amplitude and recording the number of cycles to failure. The specimens can be very different, e.g. they can be a smooth, notch-free piece of material or a full-size structure. In addition, the specimens may be exposed to different environments, (*e.g.* air at room temperature or salt water) which has inherent stress concentrations and a corrosive environment. Fatigue strength therefore describes the resistance against constant amplitude loading of a specific component and environment, which has the advantage that the nominal stresses can be used to evaluate fatigue-life. Experiments show that the Basquin equation [68] describes the fatigue strength for the whole domain or as a piecewise function for disjoint parts of the domain [69]:

$$N(\Delta\sigma) = \begin{cases} C_1 \Delta\sigma^{-b_1} & \text{for } \Delta\sigma > \Delta\sigma_c, \textit{and} \\ C_2 \Delta\sigma^{-b_2} & \text{elsewhere,} \end{cases} \tag{22}$$

where C_i, b_i are parameters for fatigue strength determined from experiments, $N(\Delta\sigma)$ is the number of cycles to failure in the stress range $\Delta\sigma$ and $\Delta\sigma_c$ is a limit stress range which splits the domain.

In general, the dynamics stress of the material in a real structure is not a harmonic load with a constant amplitude. It is therefore necessary to extract load ranges from the variable-amplitude loading and then combine these load ranges with the fatigue-life model for constant-amplitude loading to determine fatigue damage of the structure.

The varying-amplitude stress signal is cycle-counted. Several cycle-counting algorithms have been proposed to extract stress ranges, but the *de-facto* standard is the rainflow cycle counting algorithm [70, 71].

In addition, the actual stress level of a stress range $\Delta\sigma = |\sigma_1 - \sigma_2|$ has a great influence on the extent of fatigue damage it causes in the material. A pure compressive stress range ($\sigma_2 < 0$) does not open the fatigue crack and therefore cannot propagate the crack, while a stress range with a maximum tensile stress close to the ultimate strength of the material will propagate a crack more than a similar stress range with a lower maximum stress. The mean stress level $\sigma_m = \frac{1}{2}(\sigma_1 + \sigma_2)$ describes the stress level of a stress range and is commonly used in a stress correction method to convert a stress range into an equivalent stress range

with zero mean value [72]. The equivalent stress range with zero mean value by the Smith-Watson-Topper stress-correction is given as :

$$\Delta\sigma_t = \sqrt{\Delta\sigma^2 + 2\Delta\sigma\Delta\sigma_m}. \tag{23}$$

The total damage D caused by the stress load with varying amplitude is finally estimated by an accumulation rule. Again, there are several theories of damage accumulation [73, 74, 75], but one that is commonly used and works quite well is the linear damage-accumulation rule proposed by Palmgren [76] and Miner [77]:

$$D = \sum_{i=1}^{k} \frac{n_i}{N(\Delta\sigma_{t,i})}, \tag{24}$$

where D is the cumulative damage, n_i is the number of cycles at stress level i, $N(\Delta\sigma_{t,i})$ is the total number of cycles-to-failure at the stress range $\Delta\sigma_{t,i}$ and k is the total number of different stress levels.

When the frequency range of the time-varying loads on a structure is lower than the structure's natural frequency, these loads are considered quasi-static. In quasi-static condition, the stress in the material is directly proportional to the external load. However, when the frequency range of the applied load broadens and overlaps with the structure's natural frequencies [78], the resulting stress is influenced by both the external load and the structure's dynamic response. In this type of damage accumulation, known as vibration fatigue [18], the direct proportionality between the external excitation and the stress load in the fatigue zone no longer applies.

Unlike fatigue under quasi-static loading with constant or variable amplitude – where the stress-time history is a known input for cycle-counting algorithms [70] and is thus considered deterministic – the stress loads causing vibration fatigue can be either deterministic or random, depending on the type of dynamic excitation. When the excitation is deterministic, such as with sinusoidal dwell or sweep, the resulting stress load is also deterministic [79]. However, in real-world scenarios, dynamic excitation is typically random [80], making the associated stress loads random as well, best characterized by statistical estimators, which are briefly summarized next [36]. Assuming the excitation at the j-th degree-of-freedom of the dynamic system is represented by a random variable $x_j(t)$ (force, displacement, velocity or acceleration), and given that its PSD $G_{xx,j}(\omega)$ is known (2), the response σ_k at the k-th stress degree of freedom can also be expressed as a PSD [18]:

$$G_{\sigma\sigma,k}(\omega) = |H_{\sigma x,jk}(\omega)|^2 \, G_{xx,j}(\omega), \tag{25}$$

where $H_{\sigma x,jk}(\omega)$ represents the FRF of the dynamic structure from the excitation signal x_j at the j-th degree of freedom to the response signal σ_k at the k-th stress degree of freedom (similar would apply for strain-based fatigue life estimation). A general definition of the dynamic structure's FRF for viscous damping is

provided by the summation term (6). Alternatively, for hysteretic damping, a specific form of the stress frequency response function is expressed as:

$$H_{\sigma x,jk}(\omega) = \sum_{r=1}^{N_m} \frac{\sigma A_{r,jk}}{\omega_r^2 - \omega^2 + \mathrm{j}\,\eta_r\,\omega_r^2}, \tag{26}$$

and describes the effect of the structure's modal properties on its stress response.

By applying the linear damage-accumulation rule [76, 77] in combination with log-log linear Basquin equation (22), the damage intensity is:

$$d = \nu_p C^{-1} \int_0^\infty \sigma^b p_a(\sigma) d\sigma, \tag{27}$$

where ν_p and $p_a(\sigma)$ represent the expected frequency of peaks and the probability density function of the load-cycle amplitude, respectively. Since the stress load is characterized by its power spectral density $G_{\sigma\sigma}(\omega)$, converting it to the time domain to obtain ν_p and $p_a(\sigma)$ would be inefficient. Consequently, various spectral cycle-counting methods have been developed; for a comprehensive review and comparison of over 20 such methods see [75]. The primary advantage of spectral methods lies in their efficiency and ease of implementation within the dynamic stress-response of structures. However, most of these methods do not directly account for the mean values of individual stress cycles.

Contemporary challenges in vibration-fatigue research originate from the increasing complexity of real-world loading conditions, particularly due to non-stationary and multiaxial loads. Non-stationarity of dynamic loading is addressed by Zorman *et al.* [81] using a short-time approach, while Trapp and Wolfsteiner [82] adopt non-statonarity matrix to decompose non-stationary processes into quasi-stationary Gaussian segments. Alternatively, Zhang *et al.* [83] apply a deep neural network to handle non-stationary loading. For recreating the non-stationary and non-Gaussian vibration load in a laboratory environment, Ren *et al.* present a novel control methodology in [84]. Addressing the multi-modal response of the dynamic structure, Sui and Zhang [85] proposed a fatigue response spectrum method, and later enhanced it in [86] by combining single moment and Projection-by-Projection approaches. For applying the fatigue spectrum to multiaxial loads, Aimé *et al.* [87] established the fatigue-damage multi-spectrum method of load-signal evaluation. Additionally, Pei *et al.* [88] investigated non-proportional multiaxial loading, and Proner *et al.* [89] experimentally demonstrated the significance of multiaxiality in vibration fatigue.

A further level of vibration-fatigue complexity arises from multiphase interactions, particularly in cases involving fluid-structure interactions [90] or mixed-phase materials [91]. Studies have explored vibration fatigue of structures such as acoustic black holes [92], ultrasonic-assisted laser-shock peened specimens [93] and directionally solidified superalloys [94]. The structural properties of polymer and composite materials

present further challenges. Researchers have examined the effects of 2.5D composite weaving [95] and functionally-graded coating [96] on the vibration fatigue life of specimens, as well as the vibration fatigue life of carbon-fiber reinforced composites [97] and 3D-printed Polylactic Acid (PLA) structures [98, 99].

5.1. fatpack

`fatpack` is an open source project for fatigue analysis in the time domain in Python. It is developed on Github (`https://github.com/gunnstein/fatpack`) and uses the Internet Systems Consortium (ISC) license. The package includes modules for cycle counting, stress correction, accelerated fatigue testing and fatigue endurance. An overview of the functions in `fatpack` can be found in Tab. 1.

Table 1: Overview of modules and functionality in fatpack

Module	Description
`rainflow`	Cycle counting by the 4-point rainflow cycle counting algorithm. Different levels of granularity are possible in cycle counting, from the definition of load levels, to the separation of cycles and residuals, to the complete extraction of ranges and corresponding averages by a single function. This granularity also allows the user to determine how residuals are treated, e.g. by counting them as full or half cycles, or to create case-specific objects such as to-from or range-mean cycle counting matrices.
`endurance`	Fatigue endurance curves can be generated for different subdivisions of the range (linear, bilinear or trilinear curves) with the Basquin relation. Damage accumulation with Miner's sum is also implemented in this module.
`stresscorrection`	Various methods for correcting mean and compressive stresses are implemented in this module.
`racetrack`	Implementation of the nonlinear race track amplitude filter, which removes low-amplitude cycles from a signal without changing the order of the remaining cycles. Often used to accelerate fatigue tests with varying amplitude and to obtain sequence effects.

The following code snippet shows the entire process of calculating fatigue damage according to the stress-life approach and the core functionality of all modules in `fatpack` with the exception of the `racetrack` module.

```
import numpy as np
import fatpack


# Assume that `y` is the data series, we generate one here
y = np.random.normal(0., 30., size=10000) * 10 + 15

# Extract the stress ranges and corresponding means by rainflow counting
S, Sm = fatpack.find_rainflow_ranges(y, return_means=True)

# Remove purely compressive stress ranges from cycle count
```

```
Smax = Sm + S / 2.
mask = (Smax > 0.)
S, Sm = S[mask], Sm[mask]

# Apply mean stress correction by Smith-Watson-Topper and find the
# equivalent stress range at zero mean.
St = fatpack.find_swt_equivalent_stress(S, Sm)

# Determine the fatigue damage, using a trilinear fatigue curve
# with detail category Sc, Miner's linear damage summation rule.
Sc = 90.0
curve = fatpack.TriLinearEnduranceCurve(Sc)
fatigue_damage = curve.find_miner_sum(St)
```

Note that the docstring of most functions and classes contains a section with examples. The main repository also contains detailed examples in notebooks.

5.2. FLife

`FLife` is an open-source Python package for fatigue-life estimation in the frequency domain and is developed on Github (`https://github.com/ladisk/FLife`) with a MIT license. The development of the `FLife` package began with the need to standardize and simplify the calculation of vibration fatigue and to create an accurate and transparent benchmark tool for existing and future spectral methods. Because the `FLife` code has been thoroughly tested for accuracy, it provides the analyst with a reliable and accurate estimate of fatigue-life. The package supports more than 20 different spectral methods, see Fig. 9. The theoretical background and comparison of the methods was done in a review article [75].
The 20+ spectral methods are divided into 4 subgroups (Fig 9):

- Narrowband correction factor: methods are based on narrowband approximation and account for broadband process with correction factor,
- Rainflow Counting (RFC) Probability Density Function (PDF) approximation: methods are based on approximation of the probability density function of Rainflow counting,
- Combined fatigue damage (cycle damage combination): methods are based on splitting of PSD of broadband process into N narrowband approximations and accounting the formation of distinct categories of cycles,
- Combined fatigue damage (narrowband damage combination): methods are based on splitting the PSD of broadband process into N narrowband approximations and summing narrowband damages by a suitable damage combination rule.

The `FLife` package is based on the stress-load PSD $G_{\sigma\sigma}(\omega)$ (Eq. (25)) in the fatigue zone. With known material fatigue parameters, `FLife` provides the estimation of the structure's fatigue life. Stress PSD can be created either in the form of a `Numpy` array for closed-form analysis or via a simple and intuitive graphical user interface. Alternatively, `FLife` can also process vibration-induced stress-loads in the form of time series, relying on the `FatPack` or `rainflow` package.

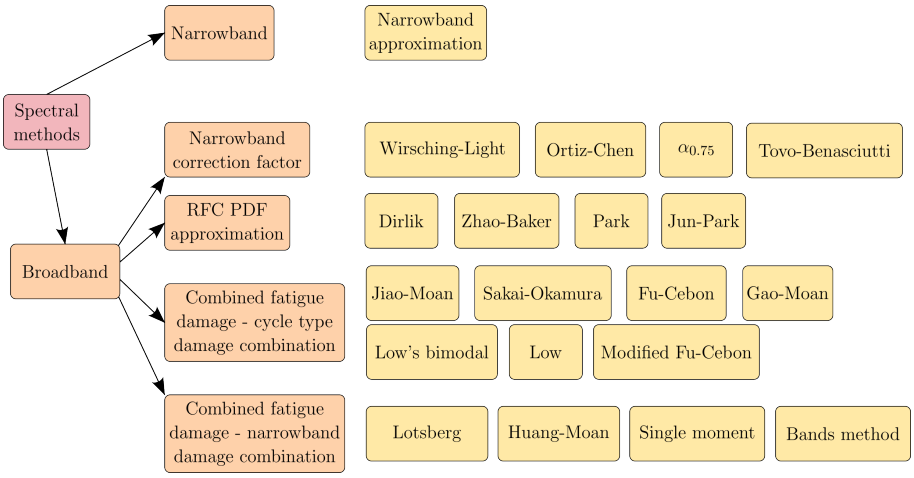


Figure 9: Structure of methods supported by FLife.

The main functionality of `FLife` is provided by the object `FLife.SpectralData`, which contains data required for fatigue-life estimation: spectral moments of PSD, spectral band estimators and other parameters. `SpectralData` is instantiated with input parameters:

- `input = 'GUI'` - PSD is provided by user via GUI (graphically and tabulary), see Fig. 10,
- `input = (PSD, freq)` - tuple of PSD and frequency vectors is provided,
- `input = (x, dt)` - tuple of time history and sampling period is provided.

The following code listing shows a short example of using `FLife` to determine the expected vibration fatigue life with three different counting methods. More detailed examples can be found in the `FLife` source code repository [100].

```
import FLife, numpy as np

C = 1.8e+22  # S-N curve intercept [MPa**k]
k = 7.3 # S-N curve inverse slope [/]
```

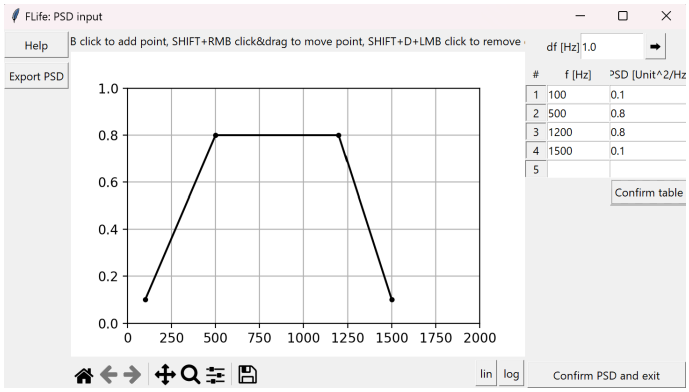


Figure 10: PSD definition using GUI.

```
freq = np.array((100, 500, 1200, 1500))
PSD = np.array((0.1, 0.8, 0.8, 0.1,))

# Instantiation of SpectralData object
sd = FLife.SpectralData(input=(PSD, freq))

# Instantiation of counting-method object -spectral and rainflow-
# on sd object
dirlik = FLife.Dirlik(sd)
tb = FLife.TovoBenasciutti(sd)
rf = FLife.Rainflow(sd, T=1, fs=1e5)

# Obtaining the estimated vibration fatigue-life
print(f'            Dirlik: {dirlik.get_life(C = C, k=k):4.0f} s')
print(f'Tovo Benasciutti 2: {tb.get_life(C = C, k=k,
                                          method="method 3"):4.0f} s')
print(f'          Rainflow: {rf.get_life(C = C, k=k):4.0f} s')
```

5.3. py-fatigue

`py-fatigue` is an open source Python package developed on Github (`https://github.com/OWI-Lab/py_fatigue`) under a GNU GPLv3 license. `py-fatigue` is used for deterministic stress-life fatigue analysis [101]. The tool is particularly useful for analyzing long-term strain signals collected by Structural Health Monitoring (SHM) systems. `py-fatigue` seamlessly processes time series observations into cycle-count matrices and stores them in an efficient JavaScript object notation (JSON) data structure that exploits their typical sparsity. By processing and storing the data in 10-minute intervals, the results of even multi-year measurements can be quickly combined without having to process the entire time series [102, 103]. Both time-series processing and fatigue analysis take advantage of the just-in-time compiler of the `numba` package to achieve better performance.

New instances of `CycleCount` can be created either with method `from_timeseries()` (array-like time series of samples) or with method `from_rainflow()` (JSON object in `py-fatigue`-friendly data structure). `CycleCount.from_timeseries()` applies the three-point rainflow counting algorithm [104] to the time series passed as input. The method accepts bin widths and lower-bound attributes for binning of stress range and mean stresses. The larger the bin widths, the coarser the degree of approximation of the cycle count matrix obtained to the original cycle count.

```
import py_fatigue as pf
import py_fatigue.testing as pft

# Generate syntetic time-series
t = pft.get_sampled_time(duration=10000, fs=10)
s = pft.get_random_data(t=t, min_=-30, range_=180,
                        random_type="weibull", a=2., seed=42)

# CycleCount definition from time-series
cycle_count = pf.CycleCount.from_timeseries(data=s, time=t,
                                            mean_bin_width=3.,
                                            range_bin_width=3.)
```

The `cycle_count` can be stored as a JSON cycle-count matrix using the `as_dict()` method. This method preserves the accuracy of the most damaging events in the time series via two user-defined attributes: `max_consecutive_zeros` to store a cycle as it is when the previous n samples were empty, and `damage_tolerance_for_binnin` to store a cycle as it is when the relative difference of Palmgren-Miner damages over a hypothetical SN curve with single slope equal to `damage_exponent` is more than `damage_tolerance_for_binning`.

```
cc_matrix = cycle_count.as_dict(max_consecutive_zeros=8,
                                damage_tolerance_for_binning=0.1,
                                damage_exponent=4)
```

Finally, `as_dict()` also stores the time series of half cycles (or remaining cycles), which represent fatigue cycles whose period is longer than the original duration of the time series. This is advantageous for obtaining low-frequency cycles [105] if several time series are to be concatenated.

In the offshore-wind industry, for example, the time series of SHM data is typically stored every 10 minutes (600s), but some highly-damaging fatigue cycles can have frequencies much lower than 1/600s $\sim$ 0.0017 Hz, and can be related to several naturally slow factors such as tidal cycles, wind gusts or even seasonal changes. When merging these 10-minute `cycle_count` objects, Low-Frequency (LF) cycles remain uncounted as residuals [105]. In `py-fatigue`, these residuals can be resolved to include uncounted LF cycles in the concatenated `cycle_count` object. This approach eliminates the need to apply rainflow counting to the entire stress signal, improving efficiency.

For this reason, `cc_matrix` can be imported back into `CycleCount` using the `from_rainflow()` method, and multiple `CycleCount` instances can be combined into a cluster using the `sum()` method to perform long-term fatigue calculations.

```
# Suppose we have a list of cycle-count matrices
cc_matrices = [cc_matrix_1, ..., cc_matrix_n]

# Cluster them together
cycle_count_list = [CycleCount.from_rainflow(m) for m in cc_matrices]
cycle_count_sum = sum(cycle_count_list)

# Retrieve the effect of low frequency fatigue
cycle_count_sum_lf = cycle_count_sum.resolve_residuals()
```

Cumulative number of full stress cycles obtained by concatenating `CycleCount` instances of 10-minute intervals for a bending-moment signal (see Fig. 11a)) are shown as full cycles without resolved residuals in Figure 11b). Full cycles with resolved residuals (see Fig. 11b)) show that high-damaging fatigue cycles with low frequencies are recovered by resolving the residuals of concatenated `cycle_count` objects. This illustration shows the effect of residual cycles on a 6-hour signal but this can readily be extended to several years [105].

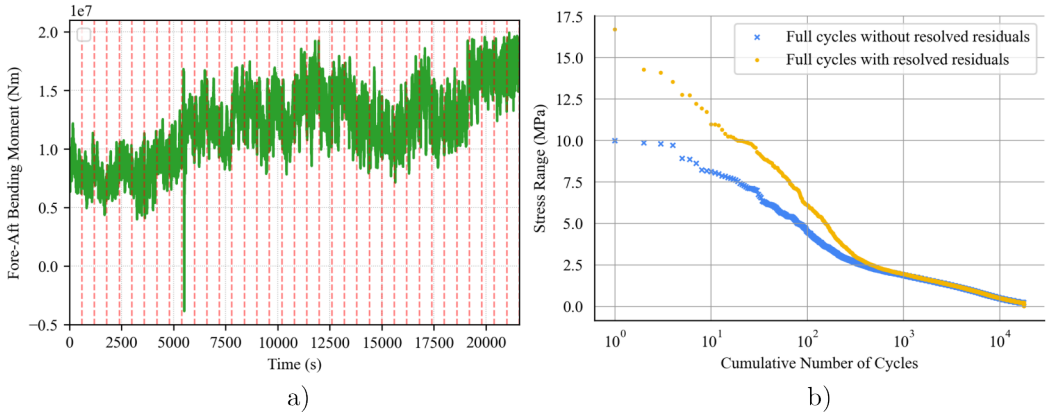


Figure 11: Wind turbine loading, a) 6-hour bending moment signal, taken from a wind turbine, with 10-minute intervals marked as red lines, b) the effect of recovered low frequency cycles.

Figure 11b) can be easily generated using built-in plot functions for `CycleCount` histograms.

```
# Generate histograms
fig, ax = plt.subplots()
ax.set_xscale('log')

cycle_count_sum.plot_histogram(fig = fig, plot_type='counts-range-cumsum', ls="",
    marker="x", label = 'Full cycles without resolved residuals')
```

```
cycle_count_sum_lf.plot_histogram(fig = fig, plot_type='counts-range-cumsum', ls="",
    marker=".", label = 'Full cycles with resolved residuals')

ax.legend()
```

Finally, stress values are usually measured at accessible locations and therefore require scaling with Stress Concentration Factor (SCF) to obtain the stress history at a specific location of the structure. Multiplying a `cycle_count` instance by a scalar is equivalent to applying an SCF to the cycle-count matrix stored in `cycle_count.stress_concentration_factor`.

Stress ranges can be corrected for the effects of mean stress using the method `cycle_count.mean_stress_correction`, which implements the correction models DNVGL [106], Goodman, Walker and Smith-Watson-Topper [107, 108].

Once the data preparation is completed, the fatigue analysis can be performed with the modules `damage.stress_life` or `damage.crack_growth`. Both methods require stress ranges and material properties that can be defined using the modules `material.sn_curve` and `material.crack_growth_curve`. In addition, the crack-growth analysis requires a crack geometry, which is defined with the module `geometry` (see [109] for more details).

For the stress-life analysis, the linear damage-accumulation rule Palmgren-Miner [77] is available, which enables the calculation of damage-equivalent stress ranges and bending moments. Non-linear damage models such as the Pavlou, Manson-Halford, Si-Jian and Leve methods [110] are also supported. For crack-growth analysis, the Paris law [111] is implemented, which allows estimations of cycles to failure for different crack geometries, including cracks in an infinite plate and both external and internal surface cracks in a cylinder. Users can manually define other geometries by specifying the crack growth rates as functions of the stress intensity factor.

6. Rotordynamics

Rotordynamic analysis has its origins in the 19th century with Rankine, de Laval and Föppl, followed by Stodola, Jeffcott and others [112]. The theory developed considerably in the 20th century with the introduction of calculation methods and modern computers [113, 114, 115, 116]. Vibration analysis of rotating machinery is still very important in the 21st century. For devices such as motors [117], pumps [118], turbines [119] and compressors [120], the amplitude of the vibrations must be limited, otherwise various errors may occur [121]. Dynamic analysis is useful both in the machine-design phase and in monitoring, diagnosis and prognosis [122]. Understanding and modeling the behavior of equipment under different operating condi-

tions is crucial for optimizing performance, preventing failures and ensuring reliability and efficiency. In this context, the development of computer models that enable the simulation of these mechanical systems and support both research and decision-making is essential. Nowadays, computer models for rotating machines are combined with machine-learning tools and improve their evaluation capacity [123, 124].

Various tools are available for rotordynamic analysis, from commercial software (such as Ansys Mechanical and COMSOL Multiphysics) to independent tools (e.g. ROTORINSA; XLTRC software). However, many of these options often require the purchase of licenses and may limit users to a graphical interface, making it difficult to perform complex and automated analysis. In addition, the lack of open and collaborative development is a significant drawback of these tools, preventing the user community from contributing improvements and new features, limiting the potential for innovation and scientific progress.

This section introduces Rotordynamic Open Source Software (ROSS) [125], an open-source library written in Python for the dynamic analysis of rotors. ROSS represents a significant effort to provide an accessible and flexible tool for the engineering community. Its code is hosted entirely in a remote GitHub repository, and the community is encouraged to actively participate in its development. This collaborative approach not only promotes transparency and accessibility of the software, but also enables continuous development driven by user contributions and feedback.

6.1. Rotordynamic model

In modeling, the rotor shaft (e.g. Timoshenko beam theory) is discretized using the finite element method, resulting in a system of second-order ordinary differential equations in which axial, lateral and torsional degrees of freedom are taken into account. Typically, the system is linearized around an equilibrium configuration with linear bearing and seal coefficients, and the disks are modeled as rigid bodies. One of the most important properties of rotor systems is that their modal parameters depend on the nominal speed of the rotor Ω. The governing equations have the form:

$$\mathbf{M}\ddot{\mathbf{x}}(t) + (\mathbf{C} + \Omega\mathbf{G})\dot{\mathbf{x}}(t) + \mathbf{K}\mathbf{x}(t) = \mathbf{f}(t)\,, \tag{28}$$

where $\mathbf{x}(t) \in \mathbb{R}^n$ represent the generalized displacements and $\mathbf{f}(t) \in \mathbb{R}^n$ the generalized forces (e.g. unbalance, misalignment). The mass, damping, gyroscopic and stiffness matrices of the system are $\mathbf{M}$, $\mathbf{C}$, $\mathbf{G}$ and $\mathbf{K}$, with dimension $n \times n$. In the frequency domain, the equations become:

$$(-\Omega^2\mathbf{M} + \mathrm{j}\,\Omega(\mathbf{C} + \Omega\mathbf{G}) + \mathbf{K})\hat{\mathbf{x}}(\Omega) = \hat{\mathbf{f}}(\Omega)\,, \tag{29}$$

where $j = \sqrt{-1}$. There are numerous analyzes that can be performed in ROSS, e.g. time and frequency responses, Campbell diagrams, complex modal analysis, and stochastic dynamics (with uncertain parameters).

Based on GitHub data, we have observed changes in engagement with ROSS's repository and documentation over the past few years. In 2020, the repository averaged 4.4 daily visitors, which increased to 12.6 in 2024. The documentation access also grew, rising from an average of 1.0 daily visitors in 2020 to 16.8 in 2023. The documentation has been accessed by users from various countries. Access statistics show the following country-wise breakdown of visitors for the last 12 months: Brazil (996), USA (778), China (522), Germany (444), India (287), France (239), Japan (193).

In [120], the first version of ROSS (not yet called ROSS) was used to analyze the effects of uncertainties in damping coefficients on rotor behavior. ROSS was applied to enhance a rotordynamic finite- element model using neural networks in [126]. Different reduced-order models were evaluated to speed up computations considering a stochastic model in [127]. Gyroscopic disk metastructures were proposed for broadband rotor vibration attenuation in [128]. Other examples of the application of ROSS in scientific investigations include: evaluating hydrodynamic bearing models [129], combining convolutional network for fault diagnosis [130], and data-driven Dirichlet sampling on manifolds to augment training sampling [124].

Rotating machine dynamic analysis is still an important field of research. Recently, a variety of investigations have been published in refereed journals. For instance, related to misalignment and nonlinear contact [131], rub-impact [132], absorbers [133], flywheels [134], couplings [135], active bearings [136], stability analysis [137], and uncertainty analysis [138].

6.2. ROSS

The source code of ROSS (Rotordynamic open-source Software) [125] is available on GitHub (`https://github.com/petrobras/ross`). It is released under the Apache 2.0 license. To use this package, you need to insert the base components, such as shaft elements, bearing elements and disks, in a list-like format. If the shaft elements are not numbered, the class defines a number for each element, corresponding to the position of the element in the list provided to the rotor constructor. ROSS is organized in different classes (shaft, disk, bearing, etc.) that are assembled to create a rotor (or multi-rotor). Then, different analyzes (methods) can be run and results post-processed as shown in Fig. 12. An example on how to build a model in ROSS is given in Fig. 13(a).

Modeling complex machines, such as a centrifugal compressor, is also possible using ROSS, see Fig. 13(b). The shaft elements are shown in gray, the disks are red and the bearings are shown as springs and dampers. After formulating the model, it is possible to plot the rotor geometry, run simulations and get results in the form of graphs. ROSS can perform different analyses, such as static analysis, critical speed map, mode

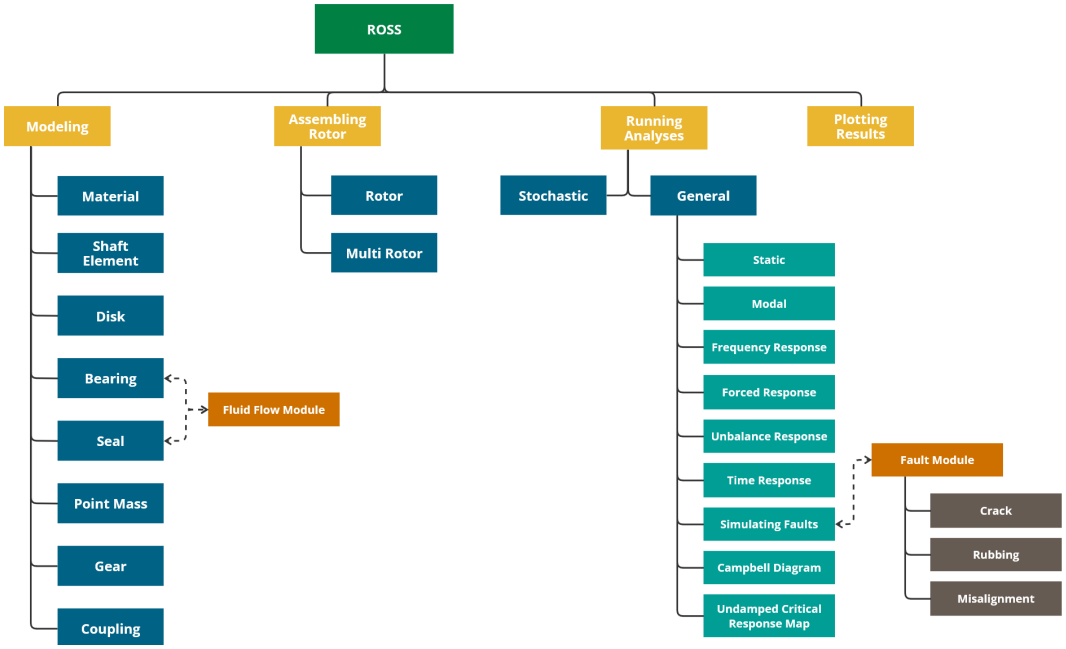


Figure 12: ROSS is organized in different classes (shaft, disk, bearing, etc.) that are assembled to create a rotor (or multi-rotor). Then, different analyzes (methods) can be run and results post-processed.

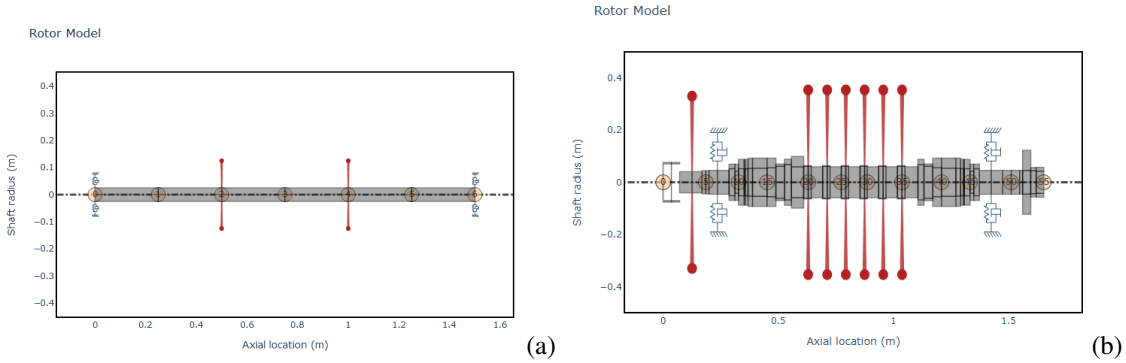


Figure 13: Example of a simple rotor model (a) and of a centrifugal compressor (b) modeled with ROSS.

shape, frequency response, and time response. The user can run the function `campbell.plot(harmonics=[0.5, 1])`, to determine the Campbell diagram presented in Fig. 14, generated for the compressor given by Fig. 13(b), the backward and forward critical speeds can be observed in the obtained Campbell diagram.

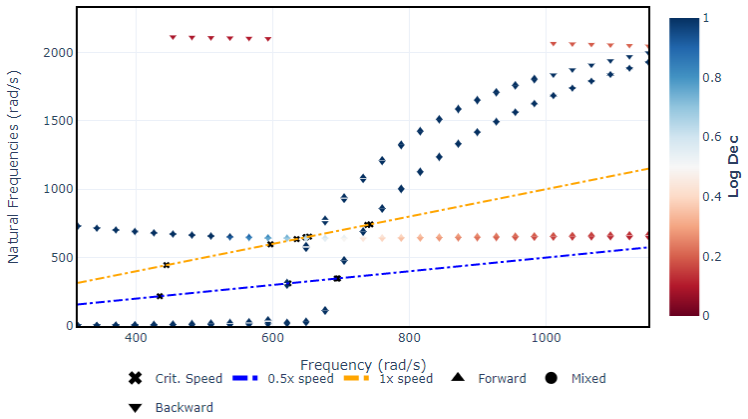


Figure 14: Campbell Diagram.

With this library, it is possible to simulate not only the rotor, but also the measurement configuration with the help of probe variables. These variables are a list of tuples with the node, the location where the reaction is to be observed and information about the orientation. Fig. 15 shows the Bode and Nyquist diagram, generated by the following code:

```
# Unbalance 1
n1 = 29 # node
m1 = 0.003 # magnitude
p1 = 0 # phase
# Unbalance 2
n2 = 33 # node
m2 = 0.002 # magnitude
p2 = 0 # phase

frequency_range = np.linspace(315, 1150, 101)
results2 = rotor3.run_unbalance_response(
[n1, n2], [m1, m2], [p1, p2], frequency_range)

probe1 = (15, 45) # node 15, orientation 45 degree
probe2 = (35, 45) # node 35, orientation 45 degree

results2.run_unbalance_response.plot(probe=[probe1, probe2], robe_units="degrees")
```

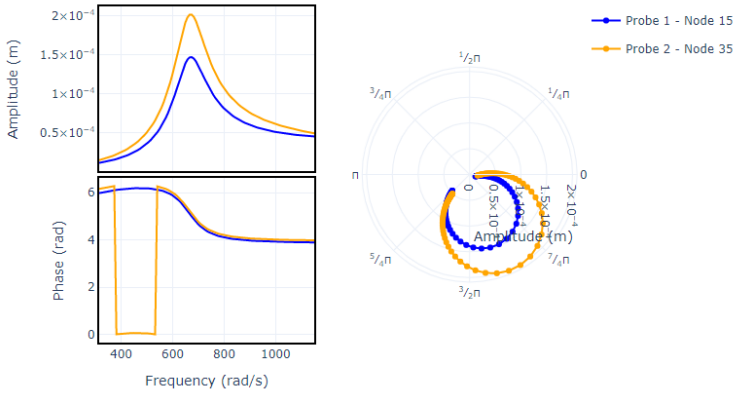


Figure 15: Unbalance analysis results with Bode and Nyquist diagram.

This library can also be used to evaluate the effects of seals on the dynamic behavior of rotors and allows the extraction of the coefficients (mass, stiffness and damping) with uncertainties [120]. In addition, ROSS has made significant progress recently. While it used to support both 4-Degree-Of-Freedom (DOF) and 6-DOF models, the tool is now moving to using only the 6-DOF model. New features have also been implemented, including a generic coupling element and a new class that enables the modeling of gear systems (see Fig. 16). The generic coupling element facilitates the simulation of different types of connections between rotors. The detailed modeling of gear systems enables the evaluation of the effect of the gear on the torsional responses of multiple rotor systems. These additions increase the ability of ROSS to perform detailed and robust analysis. The 6-DOF model, now equipped with these features, provides users with a more efficient and accurate tool for simulating complex mechanical systems, ensuring better predictions of system performance and reliability. The addition of a new class in ROSS dedicated to the simulation of seals (e.g. labyrinth seals, honeycomb seals and hole pattern seals) is also planned.

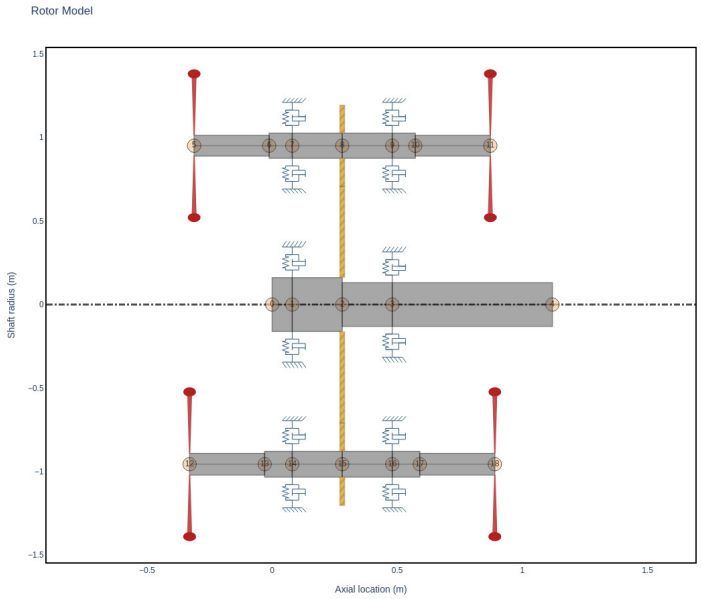


Figure 16: Modeling of gear systems.

As it is an open-source library coded in the Python language, it facilitates the integration of codes with other programs without depending on commercial software. In addition, the ROSS package has detailed documentation and a series of examples in Jupyter Notebooks with instructions for use.

The ROSS package enables numerical analysis of rotating machinery under various operating conditions involving components such as shafts, bearings and seals. This capability enables the simulation of critical rotor-dynamics issues, including development and industrial applications, and supports fault diagnosis. Future work on the ROSS library will focus on the integration of learning algorithms for digital twin technologies and fault-detection applications.

The section is concluded with a code that shows how to construct and plot a rotor using ROSS (many other examples can be found on `https://github.com/petrobras/ross`):

```
# Rotor example

# Importing ross and numpy
#(ross must be installed '!pip install ross-rotordynamics')
import ross as rs
import numpy as np

# Beginning shaft, disks, and bearings
shaft_elements = []
disk_elements = []
```

```
bearing_seal_elements = []

# Choosing the material
steel = rs.Material.load_material("Steel")

# Creating the shaft with 6 finite elements
for i in range(6):
    shaft_elements.append(rs.ShaftElement(
      L=0.25, material=steel, n=i, idl=0, odl=0.05))

# Creating two discs
disk_elements.append(rs.DiskElement.from_geometry(
  n=2, material=steel, width=0.07, i_d=0.05, o_d=0.28))
disk_elements.append(rs.DiskElement.from_geometry(
  n=4, material=steel,  width=0.07, i_d=0.05, o_d=0.35))

# Creating two bearings
bearing_seal_elements.append(rs.BearingElement(
  n=0, kxx=1e6, kyy=1e6, cxx=0, cyy=0))
bearing_seal_elements.append(rs.BearingElement(
  n=6, kxx=1e6, kyy=1e6, cxx=0, cyy=0))

# Creating the rotor with all the components (shaft, disks, and bearings)
rotor1 = rs.Rotor(shaft_elements=shaft_elements,
                  bearing_elements=bearing_seal_elements,
                  disk_elements=disk_elements,)

# Plotting rotor
rotor1.plot_rotor()
```

7. Substructuring and Transfer Path Analysis

Dynamic Substructuring (DS) and Transfer Path Analysis (TPA) are engineering concepts where dynamic aspects of complete products are characterized on the basis of individually-examined components or substructures. This approach can reduce the complexity of the overall problem and provide insight into how to optimize or troubleshoot the individual component. One of the ways to model the substructures is to use FRFs, where substructures can be fully modeled by an experimental approach [139]. The approach known as Frequency-Based Substructuring (FBS) also enables the construction of hybrid models in which experimentally-characterized and numerically-modeled (not yet produced) parts can be combined. TPA is a reliable and effective tool for determining the critical paths for the transmission of sound and vibration from the actively vibrating components of a product to the connected passive substructures. In cases where vibrating mechanisms are too complex to model or measure directly, TPA characterises a source with a set of forces that mimic the operational responses [140]. The combination of DS and TPA facilitates Virtual Acoustic Prototyping (VAP) and enables the simulation of acoustic responses in yet-to-be-developed products, allowing design and performance to be optimized early in the development process [141, 142]. In

addition, response models of the interconnecting joints can also be isolated and later parameterized with FBS [143]. In [144], an FBS framework was proposed to develop a machine-learning-based approach to estimate the mass and stiffness parameters of the joints. Similarly, in [145], FBS is used to generate the training-set samples in the form of hybrid models of the structure of interest for machine-learning-based joint health monitoring. In the context of experimental FBS, the main challenge for a successful substructure-coupling implementation remains the modelling of the common interface between the substructures. Significant effort has been performed by the researchers to find a suitable and reliable interface model by means of Virtual Point Transformation (VPT) [146], extended to flexible interfaces in [147, 148], or Singular Vector Transformation (SVT) [149]. In [150, 151], authors provide a practical and reliable methodology for the quantification and propagation of the random measurement uncertainty propagation in FBS.

Lagrange Multiplier Frequency-based Substructuring. The Lagrange Multiplier - Frequency Based Substructuring (LM-FBS) determines the admittance of the assembled system from the admittances of the individual subsystems with a set of interface forces as unknown variables. A short recap of the LM-FBS is summarized in the following according to [139]. Consider two substructures A and B connected at the interface DOFs $(\star)_2^{\mathrm{A}}$ and $(\star)_2^{\mathrm{B}}$ as depicted in Fig. 17. With admittances of the individual subsystems ($\mathbf{Y}^{\mathrm{A}}$ and $\mathbf{Y}^{\mathrm{B}}$) partitioned in internal ($(\star)_1^{\mathrm{A}}$ and $(\star)_3^{\mathrm{B}}$) and interface DOFs, the governing equation of motion for the uncoupled system can be written as[1]:

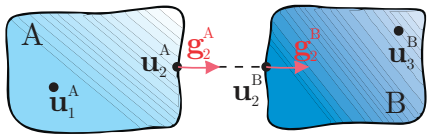


Figure 17: Substructures A and B to be coupled at the common interface.

$$\mathbf{u} = \mathbf{Y}^{\mathrm{A|B}}\left(\mathbf{f}+\mathbf{g}\right),$$

$$\mathbf{u} = \begin{bmatrix} \mathbf{u}_1^{\mathrm{A}} \\ \mathbf{u}_2^{\mathrm{A}} \\ \mathbf{u}_2^{\mathrm{B}} \\ \mathbf{u}_3^{\mathrm{B}} \end{bmatrix}, \quad \mathbf{Y}^{\mathrm{A|B}} = \begin{bmatrix} \mathbf{Y}_{11}^{\mathrm{A}} & \mathbf{Y}_{12}^{\mathrm{A}} & \mathbf{0} & \mathbf{0} \\ \mathbf{Y}_{21}^{\mathrm{A}} & \mathbf{Y}_{22}^{\mathrm{A}} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{Y}_{22}^{\mathrm{B}} & \mathbf{Y}_{23}^{\mathrm{B}} \\ \mathbf{0} & \mathbf{0} & \mathbf{Y}_{32}^{\mathrm{B}} & \mathbf{Y}_{33}^{\mathrm{B}} \end{bmatrix}, \quad \mathbf{f} = \begin{bmatrix} \mathbf{f}_1^{\mathrm{A}} \\ \mathbf{f}_2^{\mathrm{A}} \\ \mathbf{f}_2^{\mathrm{B}} \\ \mathbf{f}_3^{\mathrm{B}} \end{bmatrix}, \quad \mathbf{g} = \begin{bmatrix} \mathbf{0} \\ \mathbf{g}_2^{\mathrm{A}} \\ \mathbf{g}_2^{\mathrm{B}} \\ \mathbf{0} \end{bmatrix}. \tag{30}$$

[1] An explicit dependence on frequency is omitted for readability.

The vector $\mathbf{u}$ represents the displacements to the external force vector $\mathbf{f}$, and $\mathbf{g}$ is the vector of interface forces that hold the substructures together and therefore only appear at interface DOFs.
The compatibility of the displacements at the common boundary is reformulated in the general formulation:

$$\mathbf{B}\,\mathbf{u} = \mathbf{0} \quad \text{where} \quad \mathbf{B} = \begin{bmatrix} \mathbf{0} & -\mathbf{I} & \mathbf{I} & \mathbf{0} \end{bmatrix}. \tag{31}$$

The equilibrium condition is enforced by replacing the interface forces with a set of unknown Lagrange multiplier vectors $\boldsymbol{\lambda}$:

$$\mathbf{g} = -\mathbf{B}^{\mathrm{T}}\boldsymbol{\lambda}. \tag{32}$$

By eliminating $\boldsymbol{\lambda}$ from the set of Eqs. (30 - 32) one obtains:

$$\mathbf{u} = \underbrace{\left[\mathbf{I} - \mathbf{Y}^{\mathrm{A|B}}\mathbf{B}^{\mathrm{T}}\left(\mathbf{B}\,\mathbf{Y}^{\mathrm{A|B}}\mathbf{B}^{\mathrm{T}}\right)^{-1}\mathbf{B}\right]\mathbf{Y}^{\mathrm{A|B}}}_{\mathbf{Y}^{\mathrm{AB}}}\mathbf{f}, \tag{33}$$

where $\mathbf{Y}^{\mathrm{AB}}$ is the admittance of the assembled system.

In- situ Tranfer Path Analysis. Consider an assembly of substructures A and B. Substructure A is an active component with the operational excitation $\mathbf{f}_1$. In contrast, no excitation is active on the passive substructure B. The responses on B are therefore a consequence of $\mathbf{f}_1$ and are observed near the interface at the indicator DOFs ($\mathbf{u}_4$).
Source excitations $\mathbf{f}_1$ are often not measurable in practice; therefore, in-situ TPA takes a different approach to describe the operational excitations. A set of equivalent forces $\mathbf{f}_2^{\mathrm{eq}}$ is introduced, which act on the DOFs of the interface. When the source is deactivated, $\mathbf{f}_2^{\mathrm{eq}}$ yields the same responses on the passive side as $\mathbf{f}_1$. The simultaneous application of the operational forces $\mathbf{f}_1$ and the equivalent forces $\mathbf{f}_2^{\mathrm{eq}}$, which act in the opposite direction, should therefore eliminate any response on the passive side:

$$\mathbf{0} = \underbrace{\mathbf{Y}_{41}^{\mathrm{AB}}\mathbf{f}_1}_{\mathbf{u}_4} + \mathbf{Y}_{42}^{\mathrm{AB}}\left(-\mathbf{f}_2^{\mathrm{eq}}\right). \tag{34}$$

Expressing the equivalent forces $\mathbf{f}_2^{\mathrm{eq}}$ yields:

$$\mathbf{f}_2^{\mathrm{eq}} = \left(\mathbf{Y}_{42}^{\mathrm{AB}}\right)^{+}\mathbf{u}_4. \tag{35}$$

7.1. pyFBS

An open-source Python package pyFBS [152] aims to make the advanced methods of FBS and TPA more accessible to the engineering community. The source code of pyFBS is available on GitLab (`https://`

gitlab.com/pyFBS/pyFBS) and published under MIT license. With pyFBS, users can perform all aspects of FBS and TPA, starting with virtual design of experiments. The package enables 3D visualization of measurement setups and supports interactive positioning of inputs and outputs:

```
import pyFBS
view3D = pyFBS.view3D()
```

Within the 3D display, the user can add geometric objects either with the PyVista Python package [153] or with *.stl* objects. 3D objects representing inputs (impacts or shaker inputs) and outputs (channel responses) can be displayed together with the corresponding labels (see Fig. 18):

```
view3D.add_stl(path_to_stl)
view3D.show_imp(dataframe_impact)
view3D.label_imp(dataframe_impact)
```

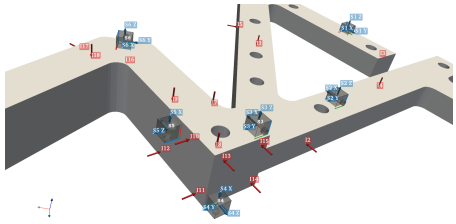


Figure 18: 3D viewer.

The measurement setup can be documented by accessing the pose of the input/output objects. The pyFBS package enables user-friendly FRF synthesis based on the mass and stiffness matrices imported from the Finite Element Method (FEM) software. Currently, only data import from Ansys is supported [154]:

```
MK = pyFBS.MK_model(rst_file, full_file)
```

Imported mode shapes can be animated in combination with the 3D display. FRFs can be synthesized using either the full harmonic method or the mode-superposition method for predefined input/output DOFs:

```
MK.FRF_synth(dataframe_channel, dataframe_impact)
```

Using synthesized or measured FRFs, state-of-the-art techniques for successful coupling or decoupling of substructures can be easily implemented with pyFBS [155]. To model the interface between the substructures, the virtual-point transformation [146, 156] and the singular-vector transformation [149] are implemented in a user-friendly way:

```
vpt = pyFBS.VPT(dataframe_channel, dataframe_impact,
    dataframe_vp_outputs, dataframe_vp_inputs)
vpt.apply_VPT(MK.freq, MK.FRF)
```

Once the collocated FRFs have been obtained for each substructure considered in the coupling, the user must code manually. First, the admittance for the uncoupled system ($\mathbf{Y}^{A|B}$) and the corresponding Boolean matrix **B** should be defined. The admittance of the full assembly is then obtained as follows:

```
Y_AB = Y_A_B - Y_A_B @ B.T @ np.linalg.inv(B @ Y_A_B @ B.T)
    @ B @ Y_A_B
```

The estimation of the equivalent set of forces can be carried out in a similar way. VPT is first applied so that the equivalent forces at the virtual interface DOFs can be estimated:

```
Y_42 = Y_4f @ vpt.Tf
```

Finally, the inverse problem for estimating the equivalent amount of force should be manually coded as:

```
f_2_eq = np.linalg.pinv(Y_42) @ u4
```

The package also offers the possibility to use DS-based expansion methods to obtain consistent response models of the substructures, i.e. mixing of two equivalent models of the same substructure into their hybrid model. One of the models provides the dynamic properties (overlay model, usually experimental) and the second model provides a dense set of DOFs (parent model, usually numerical). In pyFBS, the System Equivalent Model Mixing (SEMM) [157, 158] and Modal System Equivalent Model Mixing (M-SEMM) [159] are implemented in a user-friendly way that adopts the input/output dataframe notation:

```
Y_AB_SEMM = pyFBS.SEMM(MK.FRF, Y_exp,
                    df_chn_num = dataframe_channel_parent,
                    df_imp_num = dataframe_impact_parent,
                    df_chn_exp = dataframe_channel_overlay,
                    df_imp_exp = dataframe_impactt_overlay,
                    SEMM_type='extended')
```

For better readability of the article, only the most basic functions of pyFBS are presented in the form of Python code. For additional information on the full capabilities of the package, the interested reader is referred to `https://pyfbs.readthedocs.io/en/latest/examples/examples.html`, where all methods implemented in pyFBS are also theoretically elaborated as well as showcased on real datasets. As decided by the authors of the package, certain aspects of the coding are intentionally left to the user to implement manually in order to improve the user's understanding of the underlying methods. For users unfamiliar with the subject, the full procedure is always accessible and explained on the package's website in the corresponding examples.

8. Machine Learning in Structural Dynamics

Modeling the dynamic behavior of structures is often a major challenge. Traditional approaches, which have been followed for many years, have provided engineers with great predictive power to design safe and durable structures. In recent years, however, the need for more complex materials and more complex structures has emerged. This need stems from several newly-imposed constraints on the design of structures. Arguably one of the most important of these constraints is the reduction of waste and the sustainability of materials and overall structures. The reduction in material volume and the introduction of sustainable but complex materials (which is possible due to recent advances in material manufacturing), leads to nonlinearities and complex structural behavior. As a result, traditional physics-based approaches to modeling structures may be insufficient to achieve the desired prediction accuracy.

As in many other fields, this problem is often tackled in structural dynamics using data-driven approaches and machine learning in particular [160]. By following the data-driven path to modeling, one can skip the mathematical formulation of the physics of a particular problem. Such an approach can be very practical when modeling structural quantities whose relationship may be complicated enough to formalize an accurate model. The same principle applies in cases where the mathematical formulation is difficult or in case of inherent uncertainties in the modeled structure. In all these cases, machine learning is a powerful tool for modeling that establishes a direct link between the observations of a phenomenon and a model to approximate the true underlying relationship of the phenomenon. An important factor that has contributed to the development of machine-learning approaches to structural dynamics is the explosion of methods developed solely from a machine-learning perspective. Such methods, especially deep-learning algorithms [161], aim to contribute to the field of artificial intelligence, which the authors argue is a superset of machine learning. Nonetheless, powerful algorithms have been developed in recent years that have applications in structural dynamics, as explained later in this article.

The various structural dynamic problems that machine-learning tools deal with can be categorized according to the three main learning problems described in [162]. The first is classification, i.e. learning how to group vectors into two or more classes. The second is regression, i.e. learning the mapping of vectors from an input space $\mathcal{X}$ to an output space $\mathcal{Y}$. The third problem is that of density estimation, which involves approximating the underlying (probability) density function of some data. This task is sometimes referred to as generative modeling, as the density functions can be used to generate samples or realizations.

In recent years, a large number of open-source code packages for machine learning have been developed. The Python modules tend to be the most widely used, providing researchers with powerful modeling and

inference capabilities. Structural dynamics is, of course, a discipline where researchers have extensively used machine learning packages to develop modeling algorithms. However, there are no widely used machine learning packages specifically targeted at structural dynamics. For more general dynamic modeling, open-source modules have been developed and are available. Such modules are `Deeptime` [163], which can be used for dimensionality reduction of data and Markov state modeling of time series. A submodule of the aforementioned module is *PySINDy* [164, 165], which is an implementation of the work presented in [166] and can be used to find equations in dynamic systems. Another example of a machine-learning module dedicated to dynamic modeling is that of Physics-Informed Neural Networks (PINNs) [167], which allows the combination of data and physical equations to improve the performance of machine-learning models.

Modules such as those mentioned above offer powerful tools for modeling dynamic systems. For structural dynamics, however, the various problems are often treated as individual problems that require special attention. Examples can be found in the literature where the work or approaches are motivated or supported by existing packages, such as [168], where an equation-discovery method is presented that is able to deal with high levels of data sparsity and noise often present in structural-dynamics applications. Similar modifications of PINNs can be found in [169, 170], where the PINN formulation is adapted to the appropriate context.

The special attention required for individual applications of structural dynamics and the unavailability of a field-specific machine-learning module have led researchers to develop methods that utilize widely used machine-learning packages. A commonly-used package with a wide range of machine-learning tools is `sklearn` [171], whose sources are available on GitHub (`https://github.com/scikit-learn/scikit-learn`) and which has been released under the BSD 3-clause license. The special package enables the use of simple machine-learning models that are not case-specific. Although simple machine-learning models are often sufficient for structural-dynamics applications, the need for more powerful models has recently led to the adaptation of neural-network models. For the use of neural networks, the two most commonly used are `PyTorch` [172] and `Tensorflow` [173]. The source code of PyTorch is available on GitHub (`https://github.com/pytorch/pytorch`) and is published under the BSD 3-clause license. Tensorflow's source is also available on GitHub (`https://github.com/tensorflow/tensorflow`) and is released under the Apache-2.0 license. Both packages offer features that have contributed to the recent explosion of research activity in machine learning. One important capability is automatic differentiation [174], which allows researchers to define the models to fit their applications, combine different model types, and still compute the necessary derivatives in an automated way to train the models on data. Another class

of machine-learning methods that has been used extensively in structural dynamics are Gaussian processes (GPs) [175]. Similar to neural networks, there are packages that allow the definition of GP models, e.g. `GPy` [176], `GPflow` [177, 178], which is built with `Tensorflow`, and `GPyTorch` [179], which was created with PyTorch.

In the absence of a specific Python module for structural dynamics, this section attempts to present how models from commonly-used Python modules can be used for the purposes of applications of the field. Machine learning in structural dynamics can be found in various areas such as structure identification, input identification, SHM, model predictive control, and surrogate modeling. Machine learning serves as a tool to solve the above problems, which can be categorized into the three types of learning algorithms (regression, classification and density estimation) and can be handled by a range of machine learning algorithms, from simpler to more complicated ones.

8.1. Regression

In regression, the first category of learning algorithms, structural identification is the most common problem encountered in structural dynamics. Structural identification is the process of creating a model that can model the dynamic behavior of a system in time [180]. Such techniques are also used for equation finding in system identification [181], input estimation [182, 183], and surrogate modeling [184]. Other current applications of regression include structural health monitoring tasks such as the localization of damage in continuous systems [185].

The usual equation of motion for a structural system is as follows:

$$\mathbf{M}\ddot{\mathbf{y}} + g(\dot{\mathbf{y}}, \mathbf{y}) = \mathbf{F}, \tag{36}$$

where $\ddot{\mathbf{y}}$, $\dot{\mathbf{y}}$, $\mathbf{y}$ are the acceleration, velocity and displacement vectors of the degrees of freedom of the system, $\mathbf{M}$ is the mass matrix, $\mathbf{F}$ is the vector of the external forces acting on the system and g is a function that describes the internal forces as a function of the velocities and displacements of the system. It is clear that the equation cannot be solved analytically if the function g is not known. The aim of such tasks is to define a model f that describes the state transition from time t to time $t+1$. The model can be written as:

$$\mathbf{y}_{t+1} = f(\mathbf{y}_{t-m:t}), \tag{37}$$

where $\mathbf{y}_t$ is the vector of displacements at time t, $\mathbf{y}_{t-m:t}$ are the vectors of displacements at times $t-m$, $t-m+1$..., $t-1$ and m is the number of previous steps that the model f takes into account to predict

the displacements at time $t+1$, which is often referred to as delay. One can therefore use a number of machine-learning methods to define the model f.

The first and probably simplest method is to perform linear regression. To do this, one must first have time series data $\{y\}_{0:t}$ available and construct a matrix $\mathbf{A}$ given by:

$$\mathbf{A} = \begin{bmatrix} y_0^1 & y_0^2 & \cdots & y_0^N & y_1^1 & \cdots & y_m^N \\ y_1^1 & y_1^2 & \cdots & y_1^N & y_1^2 & \cdots & y_{m+1}^N \\ \vdots & \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ y_{t-m}^1 & y_{t-m}^2 & \cdots & y_{t-m}^N & y_{t-m}^2 & \cdots & y_{t-1}^N \end{bmatrix}, \tag{38}$$

where y_t^i is the displacement of the i^{th} degree of freedom of the structure and at time t. Similarly, the $\mathbf{B}$ matrix is created as follows:

$$\mathbf{B} = \begin{bmatrix} y_{m+1}^1 & y_{m+1}^2 & \cdots & y_{m+1}^N \\ y_{m+2}^1 & y_{m+2}^2 & \cdots & y_{m+2}^N \\ \vdots & \vdots & \ddots & \vdots \\ y_t^1 & y_t^2 & \cdots & y_t^N \end{bmatrix}. \tag{39}$$

Based on the two matrices, one may use `scikit-learn` to define a linear model as follows:

```
from sklearn.linear_model import LinearRegression
lin_model = LinearRegression()     # Define the model
lin_model.fit(A, B)                # Fit the model to the data
```

Considering that one wants to make predictions for the time step $t+1$ of the time series, a new matrix $\mathbf{C}$ can be defined as follows:

$$\mathbf{C} = \begin{bmatrix} y_{t-m+1}^1 & y_{t-m+1}^2 & \cdots & y_{t-m+1}^N & y_t^1 & \cdots & y_t^N \end{bmatrix} \tag{40}$$

and using the following code, the model can be used to make the prediction:

```
preds = lin_model(C)
```

The specific algorithm is a very simple way to define a one-step-ahead model. By making predictions for time $t+1$ and then rolling the matrix $\mathbf{C}$ to contain the prediction at time $t+1$, the model can be used again to make predictions for time $t+2$ and so on. However, the specific model is linear. Linearity limits the predictive power of the model in several ways. Therefore, a more complex and powerful model can be used. Conveniently, the matrices $\mathbf{A}$ and $\mathbf{B}$ do not need to be changed to be used by another model. More specifically, `PyTorch` can be used to define a neural network that performs the predictions for one step in advance. In this case, it is a feedforward neural network and the code to define this network is provided:

```
import torch
# n_dof: the number of degrees-of-freedom
# m: the number of lag timesteps; the lag
# n_hidden: the size of the hidden layer
model = torch.nn.Sequential(
torch.nn.Linear(n_dof * m, n_hidden),
torch.nn.ReLU(),
torch.nn.Linear(n_hidden, n_dof))
```

The code snippet above defines a simple neural network with an input layer, a hidden layer and an output layer. To train the network, the following code can be used:

```
# Loss function, mean-squared error in this case
loss_fn = torch.nn.MSELoss()

# Define the optimisation algorithm
optimiser = torch.otpim.Adam(model.parameters(), lr=0.001)

# Training loop
n_epochs = 20
for epoch in range(n_epochs):
    y_pred = model(A)
    loss = loss_fn(y_pred, B)
    optimizer.zero_grad()
    loss.backward()
    optimizer.step()
```

In the code snippet above, the loss function is defined and, since it is a regression problem, selected as the mean-square-loss function. The Adam optimizer [186] with a Learning Rate (LR) of 0.001 is chosen as the optimization algorithm and the training is performed for 20 epochs. The above code is a very simple and straightforward definition and training of a neural-network model. Mini-batching and other techniques, such as validation, are not used, but their addition is relatively simple and depends on the application one is dealing with. The advantage of `PyTorch` and `Tensorflow` is that the definition and training of neural network models can be easily modified and that users are free to define their own algorithms for each part of the training procedure.

Exactly the same model defined above can also be used for surrogate modeling. By slightly modifying the model, it could be used for various regression applications of structural dynamics. One can change the variables that define the size of the input, hidden and output layers according to the needs of a regression problem and train the new model on the appropriate data.

8.2. Classification

Classification models are very useful in the field of SHM. Since the early years of research in the field [187], statistical classifiers have been used to distinguish between different damaged or undamaged states

of structures, and to this day, classification models are used to solve current damage-detection problems [188]. Classification algorithms such as those presented in this article have also been used extensively in damage localization to identify the location of damage, reducing the problem to a classification problem [189]. In recent years, attention has shifted to transfer learning approaches and the field of Population-Based Structural Health Monitoring (PBSHM) [190, 191, 192, 193, 194] has been developed with the aim of using data from extensively-monitored structures to perform inference in the case of data-scarce structures. In these works, machine learning is extensively used to create classification models that work for different structures and classify the structures according to their similarity. The basic models used are similar to those discussed here, but the transfer requires special handling depending on the application.

To start creating and training a machine-learning model, a matrix of damage-sensitive features is needed. The damage-sensitive features must be correctly selected with technical intuition and experience. Such a matrix can be defined as:

$$\mathbf{D} = [\mathbf{x}^1, \mathbf{x}^2 ..., \mathbf{x}^K], \tag{41}$$

where $\mathbf{x}^i$ is the i^{th} vector of damage-sensitive features used for classification. The targets are then defined. In the case of binary classification, the targets are placed in a vector given by:

$$\mathbf{T} = [y^1, y^2 ..., y^k]^T, \tag{42}$$

where $y^i \in \{0, 1\}$ is the i^{th} label. For the case of binary classification, `sklearn` implementations of classification algorithms can be used to distinguish between two classes – most often these classes are the undamaged and the damaged state of a structure. Examples of models that can be used to classify according to the damage-sensitive data in D of Eq. (41) are logistic regression and Support Vector Machines (SVM) [195] and the corresponding implementation with `sklearn`:

```
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC

# Fit the models to the data D and labels T
log_reg_model = LogisticRegression()
log_reg_model.fit(D, T)

svm_model = SVC()
svm_model.fit(D, T)

# Predict the states of a structure according to testing data D_test
T_test_log_reg = log_reg_model.predict(D_test)
T_test_svm = svm_model.predict(D_test)
```

In the case of multiple classes, the targets are given in a matrix form as:

$$\mathbf{T} = [\mathbf{y}^1, \mathbf{y}^2 ..., \mathbf{y}^k]^T, \tag{43}$$

where $\mathbf{y}^i$ is the i^{th} target vector with a one-hot encoding of the class of the i^{th} sample; e.g. for the sample i, which belongs to the second of three classes, the vector $\mathbf{y}^i$ would be $[0,1,0]$.

Similar to before, a simple model can be defined with `scikit-learn`, such as Support Vector Machines, Gaussian mixture models, k-nearest neighbors, etc. A commonly-used algorithm in the case of multiple classes is the decision tree. The convenience of the algorithm results from its interpretability. The implementation with `sklearn` is similar to the case of binary classification and is given by:

```
from sklearn.tree import DecisionTreeClassifier

# Define the model and fit it to the data D and the damage-class labels T
dec_tree_model = DecisionTreeClassifier()
dec_tree_model.fit(D, T)

# Predict the states of a structure according to testing data D_test
T_test = dec_tree_model.predict(D_test)
```

Although such approaches are efficient, more powerful approaches, such as neural networks can also be used. In this case, the code to create a multi-class classification model using PyTorch is:

```
import torch
# n_feature: the number of damage-sensitive features
# n_hidden: the size of the hidden layer
# n_classes: the number of classes
model = torch.nn.Sequential(
torch.nn.Linear(n_feature, n_hidden),
torch.nn.ReLU(),
torch.nn.Linear(n_hidden, n_classes),
torch.nn.Softmax())
```

The model is a simple model with input, hidden and output layers. The network is trained in the same way as before, but a different loss function must be used for classification. In cases of multi-class classification, the cross-entropy loss is usually used, which is defined as follows:

```
loss = nn.CrossEntropyLoss()
```

8.3. Density estimation

The third problem of learning, as defined in [162], is to define the PDF of some observed data and possibly define a generative model that can be used to generate data from the underlying distribution. Such models also provide functionalities that are useful for structural dynamics. Similar to damage classification, density-estimation models are used to determine the probability of outliers in the detection of damage. In recent

work, such density-estimation models are combined with active-learning approaches to address the problem of data scarcity and the high cost of labeling data in the domain [196]. Generative models such as Variational AutoEncoders (VAE) [197] have been used to assess wind-turbine degradation based on Supervisory Control and Data Acquisition (SCADA) data [198] as well as part of a reduced-order model [199, 200]. In other cases, variations of Generative Adversarial Networks (GANs) [201] are used as a nonlinear transformation for nonlinear modal analysis [202][2]. Similarly, normalizing flows were used for the definition of nonlinear normal modes [203].

This task has long been treated with approaches ranging from the very simple to the more complicated. Simple approaches include the assumption that the data originate from a particular distribution (e.g. Bernoulli, Gaussian, log-normal, etc.) and the use of the observed data to empirically calculate the parameters of the distributions. `SciPy` [2] provides tools to fit such distributions to available data. Among the more complicated and powerful approaches is Kernel Density Estimation (KDE) [204], which can be used to define the probability density function of data with multimodal underlying distributions. Again, `SciPy` provides an implementation of the algorithm, and given a matrix of features like the one in Eq. (41), the KDE of the data can be defined as follows:

```
import numpy as np
import scipy
# D: number of samples X number of features numpy array
# D_t: numpy array with newly observed data
kernel = scipy.stats.gaussian_kde(D)

# Calculate the PDF of newly-observed data D_t
PDF_new = kernel.pdf(D_t)

# Sample n_new data points
new_samples = kernel.resample(n_new)
```

Approaches such as those mentioned above are relatively easy to implement, and deliver satisfactory results with low-dimensional data. However, as the dimensionality of the data increases, such methods become computationally intensive and in some cases are no longer as effective as with low-dimensional data. In recent years, solutions to the problem of sampling from high-dimensional underlying distributions have been the target of extensive research. The result has been a variety of methods for artificially generated data, especially for images.

One of the first methods to generate realistic-looking images from an underlying distribution is that of GANs.

[2]The code can be found in `https://github.com/GiorgTsial/GAN_NL_modal_analysis`

The GAN model defines a competition between two neural networks, the generator and the discriminator, whose aim is to use random noise to generate real-looking data or to distinguish between real data and artificially generated data. While GANs can be used to generate samples from the underlying distribution (or manifold) of the data, a major drawback is that they cannot be used to assign a probability density to the samples. Several variations of the original algorithm can be found in the literature, such as the conditional GAN (cGAN) [205], which includes a control variable in the input of the generator and discriminator and allows sampling from conditional probability distributions, and the Wasserstein GAN (WGAN) [206], which improves the stability of the training of the models and introduces a more informative loss function. Open-source resources are available for many of the numerous variations of GAN (often referred to as vanilla GAN). A repository with implementations of several of these variants using `PyTorch` can be found here `https://github.com/eriklindernoren/PyTorch-GAN`.

Close to the time GANs were introduced as a generative model, other models with similar capabilities were also introduced. An alternative to generative modeling is that of the VAE. The VAE uses the neural network architecture of an AutoEncoder (AE), which projects the data into a latent space with a smaller dimension than the data dimension and then maps the latent vectors back into the original space. The traditional AE approach places no restrictions on the structure of the latent space, whereas the VAE restricts the latent variables to independent Gaussian distributions. With a VAE, one can draw samples of variables from the latent space and generate samples from the underlying distribution of the data. As in the case of GANs, different variants have been presented for VAEs. A comprehensive but not exhaustive list of implementations in PyTorch can be found here `https://github.com/AntixK/PyTorch-VAE`.

Another deep-learning approach to generative modeling is offered by Normalizing Flows (NFs) [207, 208]. This specific algorithm provides a capability that GANs and VAEs do not offer, namely the estimation of the probability density at the points of the data space. NFs use a series of invertible transformations to map points from a Gaussian distribution to samples of the underlying distribution of the data. One can therefore take points from the Gaussian distribution and transform them into artificial samples of the data distribution or use the inverse transformations to calculate the probability density of samples of the data space. A comprehensive implementation of NFs in pytorch can be found in [209] and the code is available in `https://github.com/VincentStimper/normalizing-flows`.

9. Vibration Control

Many engineered components and assemblies are exposed to dynamic environments during their lifetime. For example, a satellite with sensitive instruments can be exposed to strong acceleration, vibration and shock loads during the rocket launch and the subsequent stage separations. However, testing in the true environment can be quite expensive. For this reason, engineers need the ability to simulate these environments in the laboratory.

Recently, researchers have begun to move away from traditional single-axis vibration testing towards testing with multiple shakers with simultaneous control to multiple sensors throughout the test object [210, 211, 212]. While the theoretical basis for performing this type of test has been known for decades (e.g. [213]), the hardware capacity required to perform these tests in real time has only recently become available.

With the flexibility to control a given test with multiple sensors from multiple excitation devices, the design space for such a test is incredibly large. For this reason, the topic of MIMO vibration tests is currently being researched intensively. Starting with the question of where the excitation sources are [214, 215] or control sensors [216, 217, 218] should be placed, through to the derivation of the actual MIMO test specification from flight data [219, 220, 221] and the application of limits and tolerances [222]. Even the control strategy itself can be investigated [223, 224], including adaptation to nonlinear structures [225] or non-stationary signals [226], or to combine several environments simultaneously [227]. While the accuracy of vibration control is an important consideration; with multiple actuators there are often limitations on the size and power available to perform a particular test. Therefore, there is considerable interest in reducing the energy required to perform such tests [228, 229] or to try to approximate the fidelity of MIMO testing with large single-axis shaker tables [230].

Vibration Control Theory. The general purpose of a vibration controller is to develop inputs to a system (typically voltage signals to a vibration actuator) that cause the system to respond in certain degrees of freedom. Depending on the type of control, the control equations can be set up in different ways.

In the case of random vibration, for example, the responses and inputs are represented as a PSD. In the case of a single control channel and a single shaker, a single PSD is used. For MIMO control, however, we will consider PSD matrices that include not only the vibration level at different frequencies for each control channel, but also the relationships between the channels, often represented as coherence and phase. PSDs are usually calculated from time signals using common techniques such as Welch's technique [9], which is the approach used by SciPy's `signal.csd` function as discussed in Section 2.

The general equation for this type of vibration problem with n_x control channels and n_v excitation signals is:

$$\mathbf{G}_{xx}(\omega) = \mathbf{H}(\omega)\,\mathbf{G}_{vv}(\omega)\,\mathbf{H}(\omega)^{\mathrm{H}}, \tag{44}$$

where $\mathbf{G}_{xx}$ is the $n_x \times n_x$ PSD matrix of the responses at the control channels, $\mathbf{G}_{vv}$ is the $n_v \times n_v$ PSD matrix of the excitation signals and $\mathbf{H}$ is the $n_x \times n_v$ transfer function matrix representing the dynamics of the system. The operator $^{\mathrm{H}}$ stands for the complex conjugate transpose of a matrix. This equation is evaluated on each frequency line ω. In general, the number of control responses n_x can be greater than, equal to or less than the number of excitation signals n_v, although in any case there are effects on vibration control [215, 231].

The goal of a random vibration controller is then to determine the correct input matrix $\mathbf{G}_{vv}$ to generate the desired response $\mathbf{G}_{xx}$. One such approach is to simply take the pseudo inverse (†) of the transfer function matrix at each frequency line:

$$\mathbf{G}_{vv}(\omega) = \mathbf{H}(\omega)^{\dagger}\mathbf{G}_{xx}(\omega)\mathbf{H}(\omega)^{\dagger^{\mathrm{H}}}. \tag{45}$$

More sophisticated strategies for controlling vibrations could implement regularization during inversion, weight control channels or frequency lines based on controllability or multiple coherence, modify the signals to meet channel response limits, update the signals based on previous control errors, etc., and will generally be a function f of the matrix of the system transfer function, the desired responses, and any other metric or user input that might be desirable to use:

$$\mathbf{G}_{vv} = f\left(\mathbf{G}_{xx}, \mathbf{H}, \ldots\right). \tag{46}$$

Once the PSD of the excitation is calculated, the controller must compute time histories that do justice to the levels and relationships between the various excitation signals on each frequency line; this can be achieved using techniques such as those found in [232, 233].

Several realizations of time histories are generated from the excitation PSD and passed on to the amplifiers of the shakers, and the responses to these signals are measured. The response PSD matrices are calculated from the response time signals. These response PSD matrices can be compared with the desired response PSD to generate various error metrics. Closed-loop vibration controllers can update the next iteration of the output signals based on the error between the desired response and the achieved response. Measured time data during control can also be used to recalculate the system transfer functions if, for example, the structure is nonlinear and the matrix of the transfer function changes with the test level.

9.1. Rattlesnake Vibration Controller

One difficulty of researching MIMO vibration control is the lack of an open platform on which to do the research. Commercial vibration control software is typically proprietary, so the ability for an independent research group to implement new ideas in existing software is limited. Many researchers end up writing their own vibration controller, but this is very inefficient; a control strategy that would conceptually be less than 20 lines of computer code might evolve into more than 1000 lines of code by the time an entire controller is written. Therefore it would be desirable if an open-source vibration control platform existed, allowing researchers to focus more on the interesting portions of the problem such as implementing new control strategies.

To fill this gap, the Rattlesnake Vibration Controller was developed. Rattlesnake is an open-source MIMO combined-environment controller [234] written in the Python programming language. The source code is available on GitHub (`https://github.com/sandialabs/rattlesnake-vibration-controller`) and is released under the General Public License (GPL)-3.0 license. Rattlesnake attempts to make it easier to experiment with new control strategies without having to significantly change the controller itself.

Rattlesnake contains abstracted interfaces to data acquisition hardware so that it can be operated with various data-acquisition systems. Currently implemented hardware interfaces include NI-DAQmx from NI, LAN-XI OpenAPI from HBK and Quattro from DataPhysics. Rattlesnake can also accept a set of state-space matrices or a SDynPy `System` object (see Sec. 10) to provide a "virtual" test article; Rattlesnake integrates these equations of motion to simulate vibration control, which is useful for developing new control strategies without having to worry about damaging expensive test equipment if something goes wrong.

Rattlesnake also abstracts the different vibration environments, resulting in each environment having a standard interface and allowing Rattlesnake to run many different environments. Currently implemented are MIMO Random Vibration Control, MIMO Transient Vibration Control, Time History Generation and Modal Testing. Due to the abstracted interface, it is trivial for Rattlesnake to run these environments simultaneously to provide a combined environment interface.

Rattlesnake includes a full graphical user interface that allows users to set up tests. Fig. 19 shows two screenshots from a random vibration test.

To facilitate the exploration of vibration research, users can load custom control laws in both the MIMO Random and MIMO Transient environments. These are either Python functions, generators or classes that must accept certain arguments and return the new output. Below is an example of a custom control law that can be loaded into Rattlesnake to implement the pseudo-inversion control described in Eq. (45).

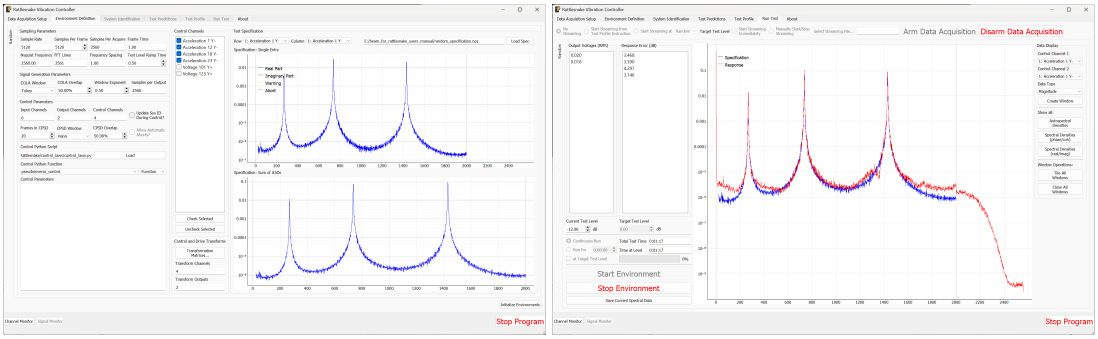


Figure 19: Screenshots of the Random Vibration environment definition (left) and running a Random Vibration test (right).

```python
import numpy as np

def pseudoinverse_control(
        specification, # Specifications
        warning_levels, # Warning levels
        abort_levels, # Abort Levels
        transfer_function,  # Transfer Functions
        noise_response_cpsd,  # Noise levels and correlation
        noise_reference_cpsd, # from the system identification
        sysid_response_cpsd,  # Response levels and correlation
        sysid_reference_cpsd, # from the system identification
        multiple_coherence, # Coherence from the system identification
        frames, # Number of frames in the PSD and FRF matrices
        total_frames, # Total frames that could be in the PSD and FRF matrices
        extra_parameters = '', # Extra parameters for the control law
        last_response_cpsd = None, # Last Control Response for Error Correction
        last_output_cpsd = None, # Last Control Excitation for Drive-based control
        ):
    # Invert the transfer function using the pseudoinverse
    tf_pinv = np.linalg.pinv(transfer_function)
    # Return the least squares solution for the new output PSD
    return tf_pinv@specification@tf_pinv.conjugate().transpose(0,2,1)
```

The modal-testing environment, while not technically a vibration-control environment, was added to allow engineers to perform dynamic characterization without changing equipment. The modal-testing environment can perform hammer impacts as well as shaker excitation. In addition, Rattlesnake's modal package can be combined with other environments by utilizing Rattlesnake's features for combined environments. For example, the user can control the response amplitudes on a set of sensors to better characterize nonlinearities in the test object during modal testing.

Although Rattlesnake was designed as research software, its capability has been demonstrated for large channel-count tests. Rattlesnake has successfully controlled the specified response of 80 channels with 24 shaker drive signals over 4000 frequency lines while streaming 250 channels of data to disk. The data stored

by Rattlesnake contains all the metadata that defines the test, so the user does not have to worry about recording which test parameters were used to perform each test. These data files can be loaded directly into the software to repeat a test.

10. SDynPy, A Structural Dynamics Framework for Python

A major challenge when performing structural dynamics calculations in Python is the lack of a consistent framework and objects to represent the data types frequently encountered in structural dynamics. In practice, NumPy `ndarrays` are often used to store data. However, this approach is prone to indexing errors, as practitioners may inadvertently select the wrong subset of rows or columns for a particular calculation. In addition, frequently-desired operations such as plotting animated modes or large data sets are not implemented. The SDynPy package was developed to overcome these challenges [235]. The SDynPy source code is available on GitHub (`https://github.com/sandialabs/sdynpy`) and is released under the GPL 3.0 license.

SDynPy defines a convenient framework for performing structural dynamics analysis. It introduces a set of objects to represent data types commonly used in structural dynamics, including:

- `CoordinateArray` – Representation of degrees of freedom defined by a node ID and the direction of the local coordinate system, e.g. `101X+`, to support bookkeeping or the visualization of measurement positions and directions. Many SDynPy objects can be directly indexed with `CoordinateArray` objects, which simplifies bookkeeping and automatically handles sign reversals, e.g. if the degree of freedom `101X-` is defined but `101X+` is requested.
- `NDDataArray` – Representation of common data types from tests or finite-element analyzes. Subclasses of this class represent specific data types, e.g. time histories (`TimeHistoryArray`), frequency response functions (`TransferFunctionArray`), and others
- `ShapeArray` – Representation of mode shapes or deflection shapes
- `Geometry` – Representation of test or finite-element geometry, consisting of nodes (`NodeArray`), coordinate systems (`CoordinateSystemArray`), tracelines (`TracelineArray`) and elements (`ElementArray`)
- `System` – Representation of mass, stiffness and damping matrices defining a dynamic system and allows "reduced" systems in which a transformation between the internal state and the physical state

is defined (e.g. a modal model or a constrained substructure model)

SDynPy array objects are typically implemented using subclasses of NumPy's `ndarray` and are therefore able to utilize features of the `ndarray` type, including arbitrary dimensionality, broadcasting, and many of the NumPy functions that operate on `ndarrays` such as `intersect1d`, `concatenate`, or `unique`.
SDynPy reflects the Python philosophy of "Batteries Included"[3] and therefore includes a wide range of structural dynamics and related tools. This decision was made to facilitate its use by structural dynamicists who are not well versed in programming concepts and probably have no interest in managing a large number of Python modules. Some of these features include: readers for common file formats (including the Universal File Format and Rattlesnake described in Sec. 9.1), simple finite elements, multi-reference mode fitters, camera geometry and calibration functions, substructuring functions and force reconstruction functions.
Perhaps SDynPy's most useful contribution to the open-source structural dynamics community is its graphical user interfaces and data visualization tools, examples of which are shown in Fig. 20. These include interactive signal processing, mode fitting, and geometry and shape visualization tools.
To summarize, SDynPy can significantly reduce the tediousness of structural dynamics analyzes, as complex operations can often be performed in just a few lines of code. This allows users to focus more on structural dynamics and less on bookkeeping and plotting operations.

11. Acoustically Induced Vibration of Pipeline Systems

Gas pipeline systems in oil and gas refineries often operate under high static and dynamic pressures, with reciprocating compressors creating the necessary conditions. Gas pulsation, a common source of excitation, can significantly affect the behavior of pipelines due to the interaction between their structural dynamics and the acoustic response of the transported gas, a phenomenon known as Acoustically-Induced Vibration (AIV) [236]. When acoustic and structural natural frequencies coincide, vibrations can amplify dramatically [237], especially in reciprocating compressors where harmonic energy is concentrated below 100 Hz [238]. To mitigate these effects, numerical techniques and standards such as API 618 [239] guide the design and analysis of safe and reliable pipeline systems that ensure adequate structural and acoustic performance [240]. These analyzes compare predicted values to allowable limits to prevent fatigue failures in accordance with API 618 and Energy Institute guidelines [241]. FEM and Computational Fluid Dynamics (CFD) are often used for fluid-induced noise and vibration [238, 242]. Although FEM and CFD are excellent for modeling

[3] see `https://docs.python.org/3/tutorial/stdlib.html#batteries-included`

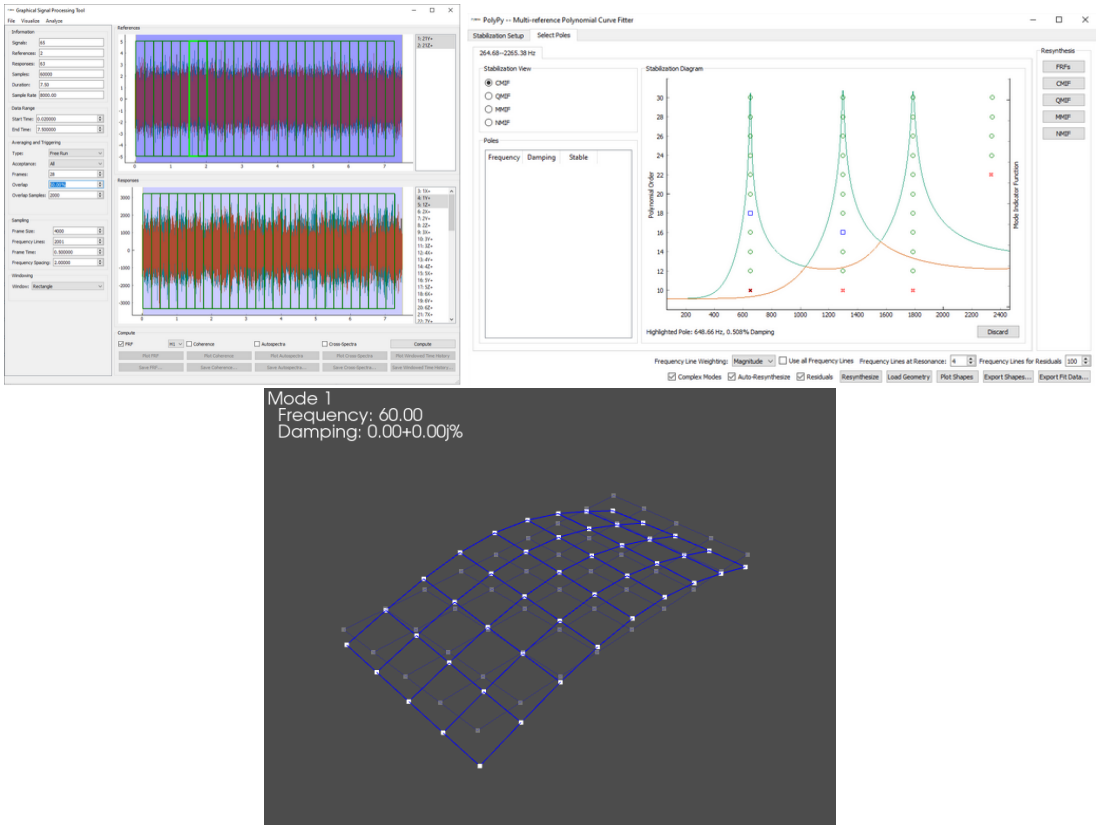


Figure 20: Example Graphical User Interfaces in SDynPy

complex geometries and local effects, their computational costs can be prohibitive for large piping systems [238]. For low-frequency plane-wave scenarios, 1D models offer a computationally efficient alternative that quickly solves global pipeline behavior and guides more detailed FEM or CFD analyzes for critical regions [236]. A classical 1D approach, the Transfer-Matrix Method (TMM), has proven to be a reliable tool for acoustic analysis over the last 40 years [236, 243]. In addition, low-frequency analysis allows the use of beam theory to model the global vibration behavior of structures, which is implemented by the FEM [244]. The integration and adaptation of TMM with FEM can provide an effective approach for the analysis of AIV in piping systems.

TMM is an analytical method for solving linear acoustic problems in chain-like systems such as mufflers, ducts and related components [245], based on the direct solution of the acoustic wave equation considering plane waves. It remains a useful tool, as demonstrated in works like Bravo and Maury [246], who suc-

[h]

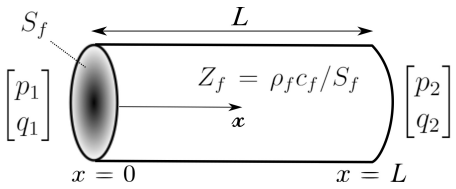


Figure 21: Uniform tube, 1D acoustic element, with length L (adapted from [236]).

cessfully applied this method in the analysis of silencers in low-speed ducted flows. Tuozzo et al. [236] investigated the mathematical properties and applications of this method for the acoustic behavior of piping systems, focusing on the study of low-frequency AIV in situations where plane waves can be assumed. Tuozzo et al. [236] and Mareze et al. [247] have found, however, that for large and complex piping systems that are still within the range of the plane-wave assumption, a simple adaptation of this method, called the Mobility-Matrix Method (MMM), provides a more suitable non-mixed numerical approach, especially for simplifying the coupling of different acoustic systems. For a given acoustic plane wave element (see Fig. 11), the MMM organizes the nodal pressures p_1 and p_2 as well as the volume velocities q_1 and q_2 from the TMM into separate vectors and transforms the transfer matrix into a mobility matrix that depends on the fluid properties such as sound velocity c_f, density ρ_f and cross-sectional area S_f. The mobility matrix of the element can be written for a given angular frequency ω [236] as:

$$\begin{bmatrix} -\mathrm{j}\cot(kx)/Z_f & \mathrm{j}/Z_f \sin(kx) \\ \mathrm{j}/Z_f \sin(kx) & -\mathrm{j}\cot(kx)/Z_f \end{bmatrix} \begin{bmatrix} p_1 \\ p_2 \end{bmatrix} = \begin{bmatrix} q_1 \\ q_2 \end{bmatrix}, \tag{47}$$

where $k = \omega/c_f$ is the acoustic wave number, $Z_f = \rho_f c_f / S_f$ and j is the complex unit.

This strategy leads to a two-noded element with one degree-of-freedom per node (pressure), which can be assembled with other elements in a similar way to the FEM. This approach is also known as the Finite Element Transfer-Matrix Method (FETM). The mobility matrix in Eq. (47) can be further refined by using dissipation models such as the Low-Reduced Frequency (LRF) model [248] and models for dissipation at low Mach numbers by Howe [249] and Peters et al. [250] can be used.

In addition, AIV analysis involves the study of the structural response of pipes excited by internal acoustic pulsations. Assuming that the pipes are made of linear-elastic isotropic materials whose length is much larger than their diameter and whose deflection is small compared to the thickness, this allows the use of Timoshenko beam theory with FEM as performed by Wu et al. [244]. Since AIV focuses on the analysis of

the global, low-frequency, structural behavior of the pipeline modeled as a beam system, a weak coupling between the structural and acoustic fields can be assumed [251]. In this case, the axial stress σ_1, the radial stress σ_r and the circumferential stress, σ_c on the structure pipe wall (which are analytically derived from the pipe dimensions and the internal pressure) are converted into axial forces [252]. In this case, for a given structural beam element, the equivalent axial forces acting at both ends can be determined after the acoustic analysis, and the additional element load vector $\mathbf{f}_p^e$ to be added to the vector of external structural forces is given by (in element coordinates):

$$\mathbf{f}_p^e = \left[-F_x, 0, 0, 0, 0, 0, F_x, 0, 0, 0, 0, 0\right]', \tag{48}$$

where, assuming ν as the Poisson coefficient of material,

$$F_x = S_f \left[\left(\sigma_1 - \nu(\sigma_r + \sigma_c) \right) \right]. \tag{49}$$

This "weak" (one-way) coupling is assumed on the basis of the following hypotheses: The fluid is a gas; the acoustic plane wave propagates axially (1-D acoustics); the pipe is thin-walled and linearly elastic; the radial inertia of the pipe wall is neglected for low frequencies; and the velocity of the acoustic wave c_f in the gas is affected by the mechanical compliance of the pipe wall, given by:

$$c_f = \left(\frac{\rho_f}{K_f} \left(1 + \frac{D_i K_f}{Et} \right) \right)^{-1/2}, \tag{50}$$

where K_f is the bulk modulus of the fluid, E is the modulus of elasticity of the pipe material, D_i is the internal diameter and t is the wall thickness of the pipe.

11.1. OpenPulse

OpenPulse [253], an open-source Python software, numerically models low-frequency AIV and allows the user to define the pipeline route and geometry, material properties, sections and loads (measured or theoretical). It first performs a time-harmonic acoustic response analysis of the 1D acoustic domain with FETM and then applies the resulting pressure field as internal distributed loads to the structural piping system modeled with FEM and Timoshenko 3D beam theory. *OpenPulse* has a GUI so that it can be used similarly to well-known commercial finite-element software. It has been implemented with PyQt5 [254] using VTK [255] to render the graphical output. Gmsh [256] is used as a mesh processor and facilitates the assignment

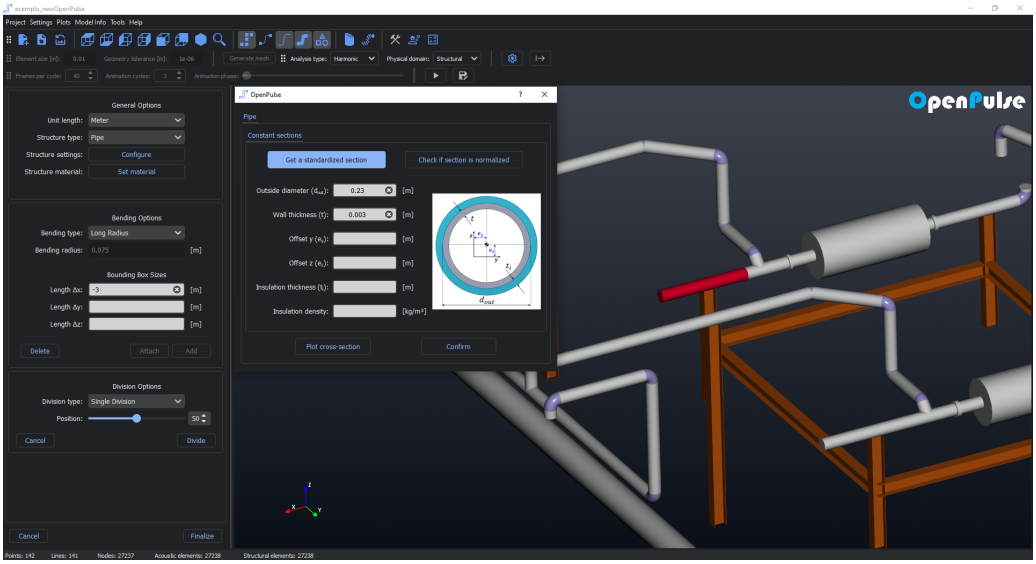


Figure 22: Path, geometry, sections and properties definition in *OpenPulse*.

of nodes and elements to properties and boundary conditions. Linear systems of equations and algebraic operations are solved with NumPy [257], SciPy [2] and PyPardiso [258]. For the visualization of the data, Matplotlib [259] is used to create 2D diagrams, while the data interfaces are based on Pandas [260] and H5py [261] to manage and store structured datasets. Finally, dependency management and environment configuration are streamlined with Poetry [262]. The source code of *OpenPulse* is available on GitHub (`https://github.com/open-pulse/OpenPulse`) and released under the GPL v3 license.

In *OpenPulse*, each section is defined sequentially based on isometric drawings, with properties assigned accordingly, as shown in Fig. 22. Different sections, materials and structural components (such as support beams, expansion chambers and valves) can be configured, allowing the design of complex systems. Each structural element is modeled with a customized beam-element formulation to accurately represent the required structural behavior.

Various types of acoustic and structural boundary conditions can be applied to the model, including sound pressure, volume velocity, acoustic impedance, prescribed displacements for support conditions, external structural forces, self-weight and elastic supports. External tables can be imported to define excitations or properties that vary with frequency.

If the user has installed the REFPROP database [263] on his computer, the gas properties can be determined by directly specifying the working pressure and temperature in *OpenPulse*. The resulting time-harmonic

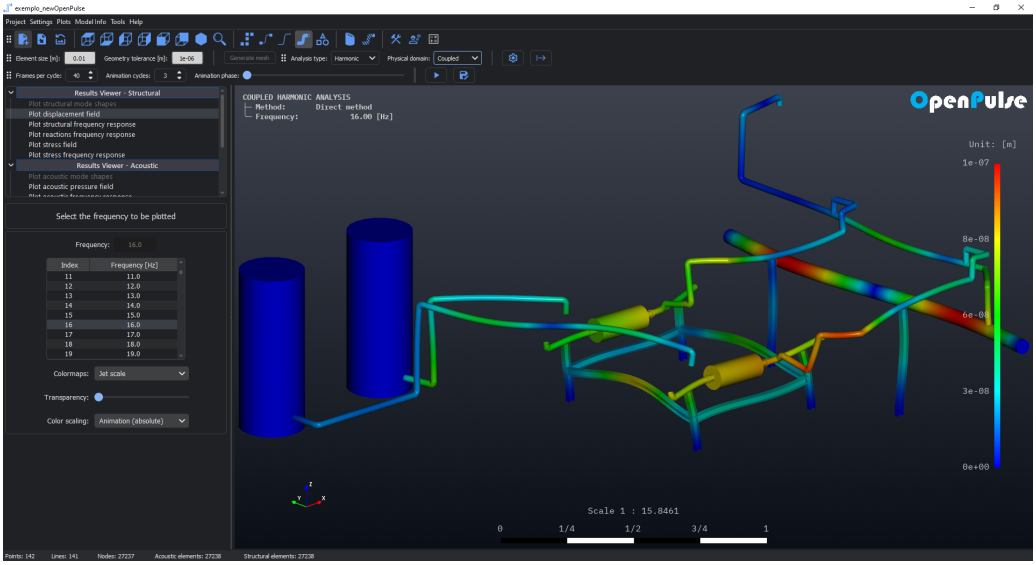


Figure 23: Displacement results for a given frequency, obtained after weak coupling with the acoustic problem.

pressure distribution is calculated for all pipes in the model and can be visualized with a representative color map. The problem of structural harmonics is then addressed by using the acoustic results to calculate the equivalent internal loads in the structural FEM model. The time-harmonic displacements of the entire system, including the non-pipe structures, can then be analyzed, as shown in Fig. 23. In addition to visualizing the global behavior, the user can plot local frequency responses for both pressure and displacement. This allows for more detailed analysis to identify potential matches between acoustic and structural resonances or to investigate specific forced response frequencies. *OpenPulse* identifies when the plane-wave assumption is not met and also indicates when the system may be nonlinear and urges caution when evaluating responses.

12. SDyPy: Structural Dynamics Python

While Scipy [2] collects and organizes most of the essential numerical algorithms, there is not yet a package that would do the same for the algorithms and methods in the field of structural dynamics. The widely used and new state-of-the-art methods are often implemented in independent packages that offer little interoperability and no common standards to enforce rigorous testing, standardized documentation, or streamlined distribution. In an effort to defragment open-source efforts in the field of structural dynamics, the SDyPy package was introduced.

The goal of SDyPy is to organize and distribute the existing packages and to provide native SDyPy functions.

The inclusion of an existing package in the SDyPy framework should therefore not affect the independence of the package beyond the will of the original authors. With the variable independence of the packages in mind, the multi-level integration framework was introduced. The multi-level integration structure presented in the SDyPy Enhancement Proposal (SEP) number 1 defines four levels of integration:

- **4th level**: External packages that deal with structural dynamics and are listed as related projects. These packages are developed completely independently and are listed as reference and potential dependencies for 3rd, 2nd and 1st level packages.
- **3rd level**: Tests and documentation correspond to the SDyPy template. The package has an MIT or similar license that allows free use for any purpose (including commercial). Availability in the Python Package Index and installation via the command *pip*.
- **2nd level**: All elements of the 3rd level. The package is a namespace package of SDyPy. Also corresponds to the SDyPy nomenclature and has the same license.
- **1st level**: All elements from level 2. Developed in a repository under the SDyPy organization.

The four levels of integration ensure that the package can be associated with SDyPy regardless of the desired independence of the authors.
If SEP1 defines the most important functions and guidelines for SDyPy, then SEP2 defines nomenclature guidelines, SEP3 defines the SDyPy namespace packages, SEP4 presents the SDyPy roadmap and SEP5 defines the unified time-series data format.
SEP4, the roadmap for the content of the SDyPy packages, defines several 1st-level packages that cover the most important parts of the field of structural dynamics. The currently implemented 1st level packages include:

- `SDyPy.EMA`: Experimental modal analysis.
- `SDyPy.FRF`: Frequency Response Function estimation.
- `SDyPy.excitation`: Generation of the excitation signals.
- `SDyPy.io`: Input/output methods (currently LVM, UFF and MRAW).

The 1st and 2nd level packages (namespace packages) are used seamlessly with the SDyPy package, as they can be accessed directly via the SDyPy namespace. The existing 1st level modules in SDyPy are presented in the following.

SDyPy.EMA submodule. SDyPy EMA is the first 1st-level package introduced in the SDyPy framework. Its origins lie in the package pyEMA [264]. After the launch of the SDyPy initiative, it was considered a good candidate for full integration into the SDyPy framework. The following example shows how it is used:

```
import sdypy as sd
ema_object = sd.EMA.Model(freq, frf_array)
ema_object.get_poles()     # Calculates the poles for stabilization
ema_object.select_poles() # Opens the interface for selecting the stable poles
```

To keep the code clear, the `SDyPy` should be imported as `sd`, similar to `numpy`'s `np`. The module `EMA` is called from the SDyPy namespace, from which the module `Model` can be called and used.

SDyPy.FRF submodule. In structural dynamics, the estimation of the frequency response function is a frequently used feature. The SDyPy FRF module is a 1st level package which serves as a channel between the long-available pyFRF [265] package and the SDyPy framework. The pyFRF package is a 3rd level package and is not itself a namespace package to allow its direct use. To avoid the active maintenance of two identical packages (pyFRF and SDyPy FRF), the SDyPy FRF imports the pyFRF and makes it available in the SDyPy framework. How to use the SDyPy FRF:

```
frf_object = sd.FRF.FRF(sampling_frequency, exc=excitation, resp=respones)
frf = frf_object.get_FRF(type="default", form="receptance") # Get the estimated FRF
```

The integration of pyFRF int SDyPy is a good example of how to integrate packages that do not want to adhere to the requirements of 1st/2nd level integration.

SDyPy.excitation submodule. The generation of excitation signals requires the handling of numerous details which, if overlooked, can lead to inappropriate excitation of the structure. The goal of SDyPy.excitation is to simplify signal generation and make it reliable. Similar to SDyPy.FRF, SDyPy.excitation channels the functionality of another package, in this case pyExSi [266]. As a consequence of pyExSi, the excitation module provides the generation of impulse signals, uniform random, normal random, pseudo-random, stationary Gaussian, stationary non-Gaussian, non-stationary non-Gaussian, burst random and sine sweep signals. To be used as part of the SDyPy framework:

```
signal = sd.excitation.pseudo_random(N) # signal of length N
```

SDyPy.io submodule. In addition to the processing of signals in the area of structural dynamics, an important aspect is the loading and saving of these signals. There are several packages that can process different formats. `SDyPy.io` aims to make these input/output packages conveniently available. Currently, SDyPy io

(similar to the FRF and excitation modules) channels the functionality of the pyUFF, lvm_read and pyMRAW packages. The module is further subdivided according to the file format used. An example of reading the Universal File Format (UFF):

```
uff_object = sd.io.uff.UFF(filename)
uff_object.read_sets() # Read the sets contained within the UFF file
```

Similarly, the LabVIEW Measurement Files (LVM) can be read:

```
lvm_data = sd.io.lvm.read(filename)
```

The raw data from the Photron high-speed cameras are saved in the MRAW format. The MRAW can be read with:

```
data, info = sd.io.mraw.load_video(filename)
```

13. Conclusions

In the last 10 years, Python-based open-source development has experienced exponential growth. Projects such as Numpy, SciPy, Pandas, TensorFlow, PyTorch, and scikit-learn have become indispensable tools for big-data and machine-learning research. The open-source nature of these tools enables fast development, collaboration and accessibility, democratizing these fields and fostering innovation. Several scientific fields have benefited from open-source research, including signal processing and mechanical-systems modeling. The readership of journals such as MSSP has been able to build on these general scientific packages and also develop several specific packages; however, the potential for the development of a general package for structural dynamics remains.

This article gives an overview of selected Python projects in the field of signal processing and mechanical systems, such as: Vibration Fatigue, Vibration Control, Substructuring, Experimental Modal Analysis. Most of the presented packages have a permissive open-source license and a broad user base. For the long-term survival of an open-source package, a broad developer base and the participation of several institutions are essential. For this reason, Sec. 12 presents a proposal for the development of a general Python package to support structural dynamics research involving multiple institutions. The SDyPy package proposes a package template that makes it relatively easy to integrate other packages. In this way, established packages can be integrated into a general framework in a relatively simple way. The integration follows a roadmap agreed by the community. In addition, the community can suggest improvements via the SDyPy Enhancement Proposals (SEP).

Open-source-based research is developed, discussed, managed, and governed openly. Openness and free use are essential for the progress of science. In this way, a researcher approaching a scientific topic can progress faster and with a smaller redundancy of effort. The same applies to an experienced researcher introducing a new method: With the underlying open-source packages, such a researcher can focus on the groundbreaking scientific contribution and reduce the effort required to implement the known methods.

Acknowledgments

The authors acknowledge partial financial support from the Slovenian Research Agency (research core funding No. P2-0263 and research project L2-60140).

Author Ivan Tomac acknowledges partial financial support from the Croatian Science Foundation (HRZZ-IP-2022-10-8856) and European Union's Horizon 2020 research and innovation programme under Marie Skłodowska-Curie Grant Agreement No. 101027829.

The authors, Dag Pasquale Pasca and Angelo Aloisio, express their gratitude to their respective institutions, Treteknisk and the University of L'Aquila, for their continued support. They also thank Diego Federico Margoni, Marco Martino Rosso, Stefanos Sotiropoulos, Giuseppe Carlo Marano, and Rocco Alaggio for their contributions to the development of the *pyOMA* project.

The authors express their gratitude to Diego M. Tuozzo, Jose L. Souza, Lucas Kulakauskas, Danilo Espindola, Vitor Slongo, Rodrigo Schwartz, Gildean Almeida, Claudio Mendona, Andr Brando and Eden Suzart for their valuable contributions during the development of *OpenPulse*.

The authors express their gratitude to Javier Del Fresno Zarza, Marie Brns, Tim vrta and Jure Korbar for their valuable contributions during the development of *pyFBS*.

Authors Thiago G. Ritto, Raphael Timbó, Aldemir A. Cavallini Jr, Vinicius T. Costa and Jéssica G. F. Santos acknowledge partial financial support from Petrobras (Petrleo Brasileiro) and Brazillian funding agencies: CNPQ, CAPES, FAPERJ, and FAPEMIG.

Sandia National Laboratories is a multi-mission laboratory managed and operated by National Technology & Engineering Solutions of Sandia, LLC (NTESS), a wholly owned subsidiary of Honeywell International Inc., for the U.S. Department of Energys National Nuclear Security Administration (DOE/NNSA) under contract DE-NA0003525. This written work is authored by an employee of NTESS. The employee, not NTESS, owns the right, title and interest in and to the written work and is responsible for its contents. Any subjective views or opinions that might be expressed in the written work do not necessarily represent the views of the U.S. Government. The publisher acknowledges that the U.S. Government retains a non-exclusive, paid-up, irrevocable, world-wide license to publish or reproduce the published form of this written work or allow

others to do so, for U.S. Government purposes. The DOE will provide public access to results of federally sponsored research in accordance with the DOE Public Access Plan.

References

[1] Matthew Rocklin. Seven stages of open software, blog at https://www.coiled.io/blog/stages-of-openness, 2020.

[2] Pauli Virtanen, Ralf Gommers, Travis E. Oliphant, Matt Haberland, Tyler Reddy, David Cournapeau, Evgeni Burovski, Pearu Peterson, Warren Weckesser, Jonathan Bright, Stéfan J. van der Walt, Matthew Brett, Joshua Wilson, K. Jarrod Millman, Nikolay Mayorov, Andrew R. J. Nelson, Eric Jones, Robert Kern, Eric Larson, C J Carey, İlhan Polat, Yu Feng, Eric W. Moore, Jake VanderPlas, Denis Laxalde, Josef Perktold, Robert Cimrman, Ian Henriksen, E. A. Quintero, Charles R. Harris, Anne M. Archibald, Antônio H. Ribeiro, Fabian Pedregosa, Paul van Mulbregt, Aditya Vijaykumar, Alessandro Pietro Bardelli, Alex Rothberg, Andreas Hilboll, Andreas Kloeckner, Anthony Scopatz, Antony Lee, Ariel Rokem, C. Nathan Woods, Chad Fulton, Charles Masson, Christian Häggström, Clark Fitzgerald, David A. Nicholson, David R. Hagen, Dmitrii V. Pasechnik, Emanuele Olivetti, Eric Martin, Eric Wieser, Fabrice Silva, Felix Lenders, Florian Wilhelm, G. Young, Gavin A. Price, Gert-Ludwig Ingold, Gregory E. Allen, Gregory R. Lee, Hervé Audren, Irvin Probst, Jörg P. Dietrich, Jacob Silterra, James T Webber, Janko Slavič, Joel Nothman, Johannes Buchner, Johannes Kulick, Johannes L. Schönberger, José Vinícius de Miranda Cardoso, Joscha Reimer, Joseph Harrington, Juan Luis Cano Rodríguez, Juan Nunez-Iglesias, Justin Kuczynski, Kevin Tritz, Martin Thoma, Matthew Newville, Matthias Kümmerer, Maximilian Bolingbroke, Michael Tartre, Mikhail Pak, Nathaniel J. Smith, Nikolai Nowaczyk, Nikolay Shebanov, Oleksandr Pavlyk, Per A. Brodtkorb, Perry Lee, Robert T. McGibbon, Roman Feldbauer, Sam Lewis, Sam Tygier, Scott Sievert, Sebastiano Vigna, Stefan Peterson, Surhud More, Tadeusz Pudlik, Takuya Oshima, Thomas J. Pingel, Thomas P. Robitaille, Thomas Spura, Thouis R. Jones, Tim Cera, Tim Leslie, Tiziano Zito, Tom Krauss, Utkarsh Upadhyay, Yaroslav O. Halchenko, and Yoshiki Vázquez-Baeza. Scipy 1.0: fundamental algorithms for scientific computing in python. *Nature Methods*, 17(3):261–272, 3 2020.

[3] Aaron Meurer, Christopher P. Smith, Mateusz Paprocki, Ondřej Čertík, Sergey B. Kirpichev, Matthew Rocklin, Amit Kumar, Sergiu Ivanov, Jason K. Moore, Sartaj Singh, Thilina Rathnayake, Sean Vig, Brian E. Granger, Richard P. Muller, Francesco Bonazzi, Harsh Gupta, Shivam Vats, Fredrik Johans-

son, Fabian Pedregosa, Matthew J. Curry, Andy R. Terrel, Štěpán Roučka, Ashutosh Saboo, Isuru Fernando, Sumith Kulal, Robert Cimrman, and Anthony Scopatz. Sympy: symbolic computing in python. *PeerJ Computer Science*, 3:e103, Jan 2017.

[4] Guido Van Rossum, Fred L Drake, et al. *Python reference manual*, volume 111. Centrum voor Wiskunde en Informatica Amsterdam, 1995.

[5] Charles R. Harris, K. Jarrod Millman, Stéfan J. van der Walt, Ralf Gommers, Pauli Virtanen, David Cournapeau, Eric Wieser, Julian Taylor, Sebastian Berg, Nathaniel J. Smith, Robert Kern, Matti Picus, Stephan Hoyer, Marten H. van Kerkwijk, Matthew Brett, Allan Haldane, Jaime Fernández del Río, Mark Wiebe, Pearu Peterson, Pierre Gérard-Marchant, Kevin Sheppard, Tyler Reddy, Warren Weckesser, Hameer Abbasi, Christoph Gohlke, and Travis E. Oliphant. Array programming with NumPy. *Nature*, 585(7825):357–362, sep 2020.

[6] J. D. Hunter. Matplotlib: A 2d graphics environment. *Computing in Science & Engineering*, 9(3):90–95, 2007.

[7] Joseph Fourier. *Théorie analytique de la chaleur*. F. Didot, Paris, 1822.

[8] James W. Cooley and John W. Tukey. An algorithm for the machine calculation of complex fourier series. *Mathematics of Computation*, 19(90):297–301, 1965.

[9] P. Welch. The use of fast fourier transform for the estimation of power spectra: A method based on time averaging over short, modified periodograms. *IEEE Transactions on Audio and Electroacoustics*, 15(2):70–73, jun 1967.

[10] T. Kailath. A view of three decades of linear filtering theory. *IEEE Transactions on Information Theory*, 20(2):146–181, mar 1974.

[11] Hongbing Wang, Shiqian Chen, and Wanming Zhai. Variational generalized nonlinear mode decomposition: Algorithm and applications. *Mechanical Systems and Signal Processing*, 206:110913, jan 2024.

[12] Meng Rao, Xingkai Yang, Zheng Cao, Ming J. Zuo, and Fulei Chu. Normalizing vibration signals with a novel piecewise power fitting method for intelligent fault detection of rotating machinery. *Mechanical Systems and Signal Processing*, 221:111751, dec 2024.

[13] Miao He and David He. A new hybrid deep signal processing approach for bearing fault diagnosis using vibration signals. *Neurocomputing*, 396:542–555, jul 2020.

[14] Ian Dias Viotti, Ronny Francis Ribeiro, and Guilherme Ferreira Gomes. Damage identification in sandwich structures using convolutional neural networks. *Mechanical Systems and Signal Processing*, 220:111649, nov 2024.

[15] Haiming Yi, Lei Hou, Yuhong Jin, Nasser A. Saeed, Ali Kandil, and Hao Duan. Time series diffusion method: A denoising diffusion probabilistic model for vibration signal generation. *Mechanical Systems and Signal Processing*, 216:111481, jul 2024.

[16] The SciPy community. Scipy roadmap. `https://docs.scipy.org/doc/scipy/dev/roadmap.html`, 2024. Accessed: 2024-11-04.

[17] Kihong Shin and Joseph K. Hammond. *Fundamentals of Signal Processing for Sound and Vibration Engineers 1st Edition*. John Wiley & Sons Ltd, Chichester, England, 2008.

[18] Janko Slavič, Matjaž Mršnik, Martin Česnik, Jaka Javh, and Miha Boltežar. *Vibration Fatigue by Spectral Methods*. Elsevier, 2021.

[19] Zhi-Fang Fu and Jimin He. *Modal analysis*. Elsevier, 2001.

[20] P Guillaume, L. Hermans, and H. Van der Auwerer. Maximum likelihood identification of modal parameters from operational data. *Proceedings of the 17th International Modal Analysis Conference (IMAC17)*, pages 1887–1893, 1999.

[21] Bart Cauberghe. *Applied frequency-domain system identification in the field of experimental and operational modal analysis*. PhD thesis, Vrije Universiteit Brussel, 2004.

[22] Patrick Guillaume, S Vanlanduit, Herman Van Der Auweraer, Bart Peeters, Peter Verboven, and Steve Vanlanduit. A poly-reference implementation of the least-squares complex frequency-domain estimator. Technical report, 2003.

[23] 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.

[24] Rune Brincker, Lingmi Zhang, and Palle Andersen. Modal identification of output-only systems using frequency domain decomposition. *Smart materials and structures*, 10(3):441, 2001.

[25] Rune Brincker, Carlos E Ventura, and Palle Andersen. 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*, pages 698–703. Society for Experimental Mechanics, 2001.

[26] Lingmi Zhang, Tong Wang, and Yukio Tamura. A frequency–spatial domain decomposition (fsdd) method for operational modal analysis. *Mechanical systems and signal processing*, 24(5):1227–1239, 2010.

[27] Peter Van Overschee. *Subspace identification for linear systems: Theory—Implementation—Applications*. Springer Science & Business Media, 2012.

[28] Bart Peeters and Guido De Roeck. Reference-based stochastic subspace identification for output-only modal analysis. *Mechanical systems and signal processing*, 13(6):855–878, 1999.

[29] Michael Döhler. *Subspace-based system identification and fault detection: Algorithms for large systems and application to structural vibration analysis*. PhD thesis, Université Rennes 1, 2011.

[30] Bart Peeters, Herman Van der Auweraer, Patrick Guillaume, and Jan Leuridan. The polymax frequency-domain method: a new standard for modal parameter estimation? *Shock and Vibration*, 11(3-4):395–409, 2004.

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

[32] Carlo Rainieri and Giovanni Fabbrocino. Operational modal analysis of civil engineering structures. *Springer, New York*, 142:143, 2014.

[33] Michael Döhler and Laurent Mevel. Efficient multi-order uncertainty computation for stochastic subspace identification. *Mechanical Systems and Signal Processing*, 38(2):346–366, 2013.

[34] Sandro DR Amador and Rune Brincker. Robust multi-dataset identification with frequency domain decomposition. *Journal of Sound and Vibration*, 508:116207, 2021.

[35] Albert Benveniste and Laurent Mevel. Nonstationary consistency of subspace methods. *IEEE Transactions on Automatic Control*, 52(6):974–984, 2007.

[36] Julius S. Bendat and Allan G. Piersol. *Random Data*. Wiley, 1 2010.

[37] Knut Andreas Kvåle. knutankv/koma: v1.3.2, 10 2024.

[38] K A 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*, pages 3041–3048. CRC Press, 2021.

[39] Ricardo J G B Campello, Davoud Moulavi, Arthur Zimek, and Jörg Sander. Hierarchical density estimates for data clustering, visualization, and outlier detection. *ACM Trans. Knowl. Discov. Data*, 10:5:1–5:51, 2015.

[40] Javad Baqersad, Peyman Poozesh, Christopher Niezrecki, and Peter Avitabile. Photogrammetry and optical methods in structural dynamics – a review. *Mechanical Systems and Signal Processing*, 86:17–34, 2017.

[41] Jaka Javh, Janko Slavič, and Miha Boltežar. The subpixel resolution of optical-flow-based modal analysis. *Mechanical Systems and Signal Processing*, 88:89–99, 5 2017.

[42] Bruce D Lucas and Takeo Kanade. An iterative image registration technique with an application to stereo vision. In *Proceedings of the 7th International Joint Conference on Artificial Intelligence - Volume 2*, IJCAI'81, pages 674–679, San Francisco, CA, USA, 1981. Morgan Kaufmann Publishers Inc.

[43] Berthold K.P. Horn and Brian G. Schunck. Determining optical flow. *Artificial Intelligence*, 17(1-3):185–203, 8 1981.

[44] Jaka Javh, Janko Slavič, and Miha Boltežar. High frequency modal identification on noisy high-speed camera data. *Mechanical Systems and Signal Processing*, 98:344–351, 2018.

[45] Ennio Mingolla, James T. Todd, and J. Farley Norman. The perception of globally coherent motion. *Vision Research*, 32(6):1015–1031, 6 1992.

[46] I. Tomac, J. Slavič, and D. Gorjup. Single-pixel optical-flow-based experimental modal analysis. *Mechanical Systems and Signal Processing*, 202:110686, 11 2023.

[47] D. Gorjup, J. Slavič, and Miha Boltežar. Frequency domain triangulation for full-field 3d operating-deflection-shape identification. *Mechanical Systems and Signal Processing*, 133:106287, 11 2019.

[48] Simon Baker and Iain Matthews. Lucas-kanade 20 years on: A unifying framework. *International Journal of Computer Vision*, 56(3):221–255, 2 2004.

[49] Bing Pan. Digital image correlation for surface deformation measurement: historical developments, recent advances and future goals. *Measurement Science and Technology*, 29(8):082001, 8 2018.

[50] Patrick O'Donoughue, François Gautier, Erwan Meteyer, Thomas Durand-Texte, Mathieu Secail-Geraud, Félix Foucart, Olivier Robin, Alain Berry, Manuel Melon, Charles Pézerat, Adrien Pelat, and Pascal Picart. Comparison of three full-field optical measurement techniques applied to vibration analysis. *Scientific Reports 2023 13:1*, 13(1):1–22, 2 2023.

[51] P. Gardonio, G. Guernieri, E. Turco, L. Dal Bo, R. Rinaldo, and A. Fusiello. Reconstruction of the sound radiation field from flexural vibration measurements with multiple cameras. *Mechanical Systems and Signal Processing*, 195:110289, 7 2023.

[52] Lele Luan, Yang Liu, and Hao Sun. Extracting high-precision full-field displacement from videos via pixel matching and optical flow. *Journal of Sound and Vibration*, page 117904, 2023.

[53] Shanwu Li, Faraz Abed Azad, and Yongchao Yang. Super-sensitivity full-field measurement of structural vibration with an adaptive incoherent optical method. *Mechanical Systems and Signal Processing*, 202:110666, 11 2023.

[54] Seyed Rasoul Atashipour and Javad Baqersad. Noninvasive identification of directionally-dependent elastic properties of soft tissues using full-field optical data. *Journal of the Mechanical Behavior of Biomedical Materials*, 151:106266, 3 2024.

[55] Shanshan Yu, Jian Zhang, Chengpeng Zhu, Zeyang Sun, and Shuai Dong. Full-field deformation measurement and cracks detection in speckle scene using the deep learning-aided digital image correlation method. *Mechanical Systems and Signal Processing*, 209:111131, 3 2024.

[56] Yuchao Wang, Weihua Hu, Jun Teng, and Yong Xia. Full-field displacement measurement of long-span bridges using one camera and robust self-adaptive complex pyramid. *Mechanical Systems and Signal Processing*, 215:111451, 6 2024.

[57] Miaoshuo Li, Guojin Feng, Rongfeng Deng, Feng Gao, Fengshou Gu, and Andrew D. Ball. Structural vibration mode identification from high-speed camera footages using an adaptive spatial filtering approach. *Mechanical Systems and Signal Processing*, 166:108422, 3 2022.

[58] Sifan Wang and Mayuko Nishio. Anomaly detection in structural dynamic systems via nonlinearity occurrence analysis using video data. *Mechanical Systems and Signal Processing*, 216:111506, 7 2024.

[59] Mingguang Shan, Xuefen Xiong, Jianfeng Wang, Mengmeng Dang, Xueqian Zhou, Luyi Liang, Zhi Zhong, Bin Liu, Lei Liu, and Lei Yu. A noise-robust vibration signal extraction method utilizing intensity optical flow. *Measurement*, 235:114889, 8 2024.

[60] Haifeng Wen, Ruikun Dong, and Peize Dong. Structural displacement measurement using deep optical flow and uncertainty analysis. *Optics and Lasers in Engineering*, 181:108364, 10 2024.

[61] Xuanshi Cheng, Shibin Wang, Huixin Wei, Linan Li, Zongze Huo, Chuanwei Li, and Zhiyong Wang. Digital image correlation by natural textures on biological skin. *Optics and Lasers in Engineering*, 165:107547, 6 2023.

[62] Paolo Neri. Augmented-resolution digital image correlation algorithm for vibration measurements. *Measurement*, 231:114565, 5 2024.

[63] Yusheng Wang, Zhixiang Huang, Pengfei Zhu, Rui Zhu, Tianci Hu, Dahai Zhang, and Dong Jiang. Effects of compressed speckle image on digital image correlation for vibration measurement. *Measurement*, 217:113041, 8 2023.

[64] Thijs Masmeijer, Ed Habtour, Klemen Zaletelj, and Janko Slavi. Directional dic method with automatic feature selection. *Mechanical Systems and Signal Processing*, 224:112080, 2 2025.

[65] Joe Minichino and Joseph Howse. *Learning OpenCV 3 computer vision with python*. Packt Publishing Ltd, 3 edition, 2 2020.

[66] Stéfan van der Walt, Johannes L Schönberger, Juan Nunez-Iglesias, François Boulogne, Joshua D Warner, Neil Yager, Emmanuelle Gouillart, Tony Yu, and the scikit-image contributors. scikit-image: image processing in python. *PeerJ*, 2:e453, 8 2014.

[67] Dan Z. Turner. Digital image correlation engine (dice) reference manual. *Sandia report, Sand2015-10606 O*, 2015.

[68] O. H. Basquin. The exponential law of endurance tests. In *Proceedings of the thirteenth annual meeting*, pages 625–630. American Society for Testing Materials, 1910.

[69] Erwin Haibach. Modifizierte Lineare Schadensakkumulationshypothese zur Bercksichtigung des Dauerfestigkeitsabsalls mit Fortschreitender Schdigung. Technical Mitteilungen 50/70, Fraunhofer, Darmstadt, 1970.

[70] Astm e1049-85(2023): Standard practices for cycle counting in fatigue analysis, 2023.

[71] ISO 12110-2:2013 metallic materials — Fatigue testing — Variable amplitude fatigue testing — Part 2: Cycle counting and related data reduction methods, 2013.

[72] N. E. Dowling, C. A. Calhoun, and A. Arcari. Mean stress effects in stress-life fatigue and the Walker equation. *Fatigue & Fracture of Engineering Materials & Structures*, 32(3):163–179, 2009.

[73] M. Aim, A. Banvillet, L. Khalij, E. Pagnacco, E. Chatelet, and R. Dufour. Experimental evaluation of multiaxial test-tailored specifications based on fatigue damage multi-spectra. *International Journal of Fatigue*, 191:108702, 2025.

[74] Wenliang Fan, Zhengliang Li, and Xiaoyang Yang. A spectral method for fatigue analysis based on nonlinear damage model. *International Journal of Fatigue*, 182:108188, 2024.

[75] Aleš Zorman, Janko Slavič, and Miha Boltežar. Vibration fatigue by spectral methods—a review with open-source support. *Mechanical Systems and Signal Processing*, 190, 2023.

[76] A. Palmgren. Die lebensdauer von kugellagern. *Zeitschrift des Vereins Deutscher Ingenieure*, 68:339–341, 1924.

[77] Milton A. Miner. Cumulative damage in fatigue. *Journal of Applied Mechanics*, 12(3):A159–A164, 9 1945.

[78] N.M.M. Maia and J.M.M. Silva. *Theoretical and Experimental Modal Analysis*. Research Studies Press, 1997.

[79] Christian Lalanne. *Sinusoidal Vibration*, volume 1. Wiley-ISTE, 3 edition, 2014.

[80] Christian Lalanne. *Random Vibration*, volume 3. Wiley-ISTE, 3 edition, 2014.

[81] Aleš Zorman, Janko Slavič, and Miha Boltežar. Short-time fatigue-life estimation for non-stationary processes considering structural dynamics. *International Journal of Fatigue*, 147:106178, 2021.

[82] Arvid Trapp and Peter Wolfsteiner. Fatigue assessment of non-stationary random loading in the frequency domain by a quasi-stationary gaussian approximation. *International Journal of Fatigue*, 148:106214, 7 2021.

[83] Chun Zhang, Ruoqing Wan, Junru He, Jian Yu, and Yinjie Zhao. Non-stationary vibration fatigue life prediction of automotive components based on long short-term memory network. *International Journal of Fatigue*, 187:108459, 10 2024.

[84] Siyu Ren, Huaihai Chen, and Ronghui Zheng. Modified time domain randomization technique for multi-shaker non-stationary non-gaussian random vibration control. *Mechanical Systems and Signal Processing*, 213:111311, 5 2024.

[85] Guohao Sui and Yahui Zhang. Response spectrum method for fatigue damage assessment of mechanical systems. *International Journal of Fatigue*, 166:107278, 2023.

[86] Guohao Sui, Xinyu Jin, Hongyu Cui, and Yahui Zhang. Improvement and test verification of the fatigue response spectrum method. *Mechanical Systems and Signal Processing*, 217:111519, 8 2024.

[87] M. Aimé, A. Banvillet, L. Khalij, E. Pagnacco, E. Chatelet, and R. Dufour. A framework proposal for new multiaxial fatigue damage and extreme response spectra in random vibrations frequency analysis. *Mechanical Systems and Signal Processing*, 213:111338, 5 2024.

[88] Xianjun Pei, Yuda Cao, Tang Gu, Mingjiang Xie, Pingsha Dong, Zhigang Wei, Jifa Mei, and Tairui Zhang. Generalizing multiaxial vibration fatigue criteria in the frequency domain: A data-driven approach. *International Journal of Fatigue*, 186:108390, 9 2024.

[89] Enrico Proner, Emiliano Mucchi, and Roberto Tovo. A relationship between fatigue damage estimation under multi-axis and single-axis random vibration. *Mechanical Systems and Signal Processing*, 215:111402, 6 2024.

[90] Yongbo Yang, Guohao Sui, and Yahui Zhang. Response spectrum method for fatigue damage assessment of aero-hydraulic pipeline systems. *Computers & Structures*, 287:107119, 10 2023.

[91] Jiang Lai, Shihao Yang, Lingling Lu, Tiancai Tan, and Lei Sun. Two-phase flow-induced vibration fatigue damage of tube bundles with clearance restriction. *Mechanical Systems and Signal Processing*, 166:108442, 3 2022.

[92] Weiqi Du, Shuxin Li, and Yuanxin Luo. A frequency domain method for vibration fatigue analysis of acoustic black hole structure. *International Journal of Fatigue*, 172:107605, 7 2023.

[93] Xiankai Meng, Xumin Leng, Chong Shan, Liucheng Zhou, Jianzhong Zhou, Shu Huang, and Jinzhong Lu. Vibration fatigue performance improvement in 2024-t351 aluminum alloy by ultrasonic-assisted laser shock peening. *International Journal of Fatigue*, 168:107471, 3 2023.

[94] Hao Lu, Yeda Lian, Jundong Wang, Zhixun Wen, Zhenwei Li, and Zhufeng Yue. Vibration fatigue assessment and crack propagation mechanism of directionally solidified superalloy with film cooling holes. *International Journal of Fatigue*, 187:108456, 10 2024.

[95] Yana Wang, Yu Gong, Qin Zhang, Yuhuai He, Jian Jiao, and Ning Hu. Vibration fatigue properties of laminated and 2.5d woven composites: A comparative study. *International Journal of Fatigue*, 168:107466, 3 2023.

[96] Cong Sun, Yao Yang, Hui Li, Hesong Xu, Feng Zhao, Zhuo Xu, Guowei Sun, Junxue Hou, and He Li. Prediction of vibration fatigue life of fiber reinforced composite thin plates with functionally graded coating under base random excitation. *Thin-Walled Structures*, 200:111891, 7 2024.

[97] Yueao Jian, Mudan Chen, Zixiang Sha, Deng'an Cai, Yue Jiang, Shuang Li, Guangming Zhou, and Xinwei Wang. High-cycle random vibration fatigue behavior of cfrp composite thin plates. *Engineering Failure Analysis*, 159:108089, 5 2024.

[98] Martin Česnik, Janko Slavič, and Miha Boltežar. Accelerated vibration-fatigue characterization for 3d-printed structures: Application to fused-filament-fabricated pla samples. *International Journal of Fatigue*, 171:107574, 6 2023.

[99] Martin Česnik and Janko Slavič. Frequency-dependent fatigue properties of additively manufactured PLA. *Polymers*, 16(15), 2024.

[100] LADISK. The FLife source code repository. `https://github.com/ladisk/FLife`, Dec 2022.

[101] Pietro D'Antuono, Wout Weijtjens, and Christof Devriendt. Py-fatigue, 2022. Accessed: 2024-09-12.

[102] Negin Sadeghi, Koen Robbelein, Pietro D'Antuono, Nymfa Noppe, Wout Weijtjens, and Christof Devriendt. Fatigue damage calculation of offshore wind turbines' long-term data considering the low-frequency fatigue dynamics. *Journal of Physics: Conference Series*, 2265(3):032063, may 2022.

[103] Gabriel Marsh, Colin Wignall, Philipp R Thies, Nigel Barltrop, Atilla Incecik, Vengatesan Venugopal, and Lars Johanning. Review and application of rainflow residue processing techniques for accurate fatigue damage estimation. *International Journal of Fatigue*, 82:757–765, 2016.

[104] C.H. McInnes and P.A. Meehan. Equivalence of four-point and three-point rainflow cycle counting algorithms. *International Journal of Fatigue*, 30(3):547–559, 2008.

[105] Negin Sadeghi, Pietro D'Antuono, Nymfa Noppe, Koen Robbelein, Wout Weijtjens, and Christof Devriendt. Quantifying the effect of low-frequency fatigue dynamics on offshore wind turbine foundations: a comparative study. *Wind Energy Science*, 8:1839–1852, 12 2023.

[106] Det Norske Veritas GL. Dnvgl-rp-c203: Fatigue design of offshore steel structures. *DNV GL: Oslo, Norway*, 2016.

[107] Gyoko Oh. Effective stress and fatigue life prediction with mean stress correction models on a ferritic stainless steel sheet. *International Journal of Fatigue*, 157:106707, 2022.

[108] KNua Smith. A stress-strain function for the fatigue of metals. *Journal of materials*, 5:767–778, 1970.

[109] OWI-Lab. py-fatigue-tutorials, 2024. Accessed: 2024-09-12.

[110] Kris Hectors and Wim De Waele. Cumulative damage and life prediction models for high-cycle fatigue of metals: A review. *Metals*, 11(2), 2021.

[111] Paul Paris and Fazil Erdogan. A critical analysis of crack propagation laws. 1963.

[112] Dimarogonas. A brief history of rotordynamics. In *Proceeding of the International Conference on Rotating Machine Dynamics*, Rotordynamics, Venice, Italy, April 1992. Springer-Verlag.

[113] J M Vance, F Y Zeidan, and B G Murphy. *Machinery Vibration and Rotordynamics*. Wiley, 2010.

[114] Michael I. Friswell, John E. T. Penny, Seamus D. Garvey, and Arthur W. Lees. *Dynamics of rotating machines*. Cambridge University Press, 2010.

[115] Y Ishida and T Yamamoto. *Linear and Nonlinear Rotordynamics: A Modern Treatment with Applications*. Wiley-VCH, second edition, 2013.

[116] R. Tiwari. *Rotor Systems: Analysis and identification*. CRC Press, 2017.

[117] Heesoo Kim, Janne Nerg, Tuhin Choudhury, and Jussi T. Sopanen. Rotordynamic simulation method of induction motors including the effects of unbalanced magnetic pull. *IEEE Access*, 8:21631 – 21643, 2020.

[118] Maurice F. White, Erik Torbergsen, and Victor A. Lumpkin. Rotordynamic analysis of a vertical pump with tilting-pad journal bearings. *Wear*, 207(1-2):128 – 136, 1997.

[119] Tobias S. Morais, Aldemir Ap. Cavalini, Gilberto P. Melo, and Valder Steffen. Input force identification in a francis hydro turbine unit model. *Mechanisms and Machine Science*, 63:309 – 323, 2019.

[120] Raphael Timbó and Thiago G. Ritto. Impact of damper seal coefficients uncertainties in rotor dynamics. *Journal of the Brazilian Society of Mechanical Sciences and Engineering*, 41(4), 2019.

[121] Subhasis Nandi, Toliyat, Hamid A., and Xiaodong Li. Condition monitoring and fault diagnosis of electrical motors - a review. *IEEE Transactions on Energy Conversion*, 20(4):719 – 729, 2005.

[122] A. Muszynska. *Rotordynamics*. Mechanical Engineering. Taylor & Francis, 2005.

[123] Olivier Janssens, Viktor Slavkovikj, Bram Vervisch, Kurt Stockman, Mia Loccufier, Steven Verstockt, Rik de Walle, and Sofie Van Hoecke. Convolutional neural network based fault detection for rotating machinery. *Journal of Sound and Vibration*, 377:331 – 345, 2016.

[124] Luan S. Prado and Thiago G. Ritto. Data driven dirichlet sampling on manifolds. *Journal of Computational Physics*, 444, 2021.

[125] Raphael Timbó, Rodrigo Martins, Gabriel Bachmann, Flavio Rangel, Júlia Mota, Juliana Valério, and Thiago G Ritto. Ross - rotordynamic open source software. *Journal of Open Source Software*, 5(48):2120, 2020.

[126] Rishith E. Meethal, Anoop Kodakkal, Mohamed Khalil, Aditya Ghantasala, Birgit Obst, Kai-Uwe Bletzinger, and Roland Wüchner. Finite element method-enhanced neural network for forward and inverse problems. *Advanced Modeling and Simulation in Engineering Sciences*, 10(1), 2023.

[127] Thiago G. Ritto, Guilherme N. Lacerda, Aldemir A. Cavallini, Raphael Timbó, and Leonardo V. Pereira. Reduced-order modeling in rotordynamics and its robustness to random matrix perturbation. *Journal of Vibration and Acoustics*, 146(1), 2024.

[128] André A T Brandão, Aline S de Paula, and Adriano T Fabro. Rainbow gyroscopic disk metastructures for broadband vibration attenuation in rotors. *Journal of Sound and Vibration*, 532, 2022.

[129] J.A. Mota, D.J.G. Maldonado, J.V. Valério, and T.G. Ritto. Modeling of hydrodynamic bearings with a novel boundary parameterization approach. *Journal of the Brazilian Society of Mechanical Sciences and Engineering*, 44(5), 2022.

[130] Seungjoo Lee, YoungSeok Kim, Hyun-Jun Choi, and Bongjun Ji. Multi-stage approach using convolutional triplet network and ensemble model for fault diagnosis in oil plant rotary machines. *Machines*, 11(11), 2023.

[131] Xingfu Ma, Zhinong Li, Jiawei Xiang, Changzheng Chen, and Fengchao Huang. Vibration characteristics of rotor system with coupling misalignment and disc-shaft nonlinear contact. *Mechanical Systems and Signal Processing*, 223, 2025.

[132] Zhimin Zhu, Yuqi Li, Chuanmei Wen, Zhong Luo, Yuanzhao Chen, and Bing Li. Rub-impact investigation of a bolted joint rotor-bearing system considering hysteresis behavior at mating interface. *Mechanical Systems and Signal Processing*, 224, 2025.

[133] Hulun Guo, Zhiwei Cao, Tianzhi Yang, and Li-Qun Chen. A variable-thickness-arch based nonlinear vibration absorber for rotor-bearing systems. *Mechanical Systems and Signal Processing*, 224, 2025.

[134] Reza Takarli, Ali Amini, Mohammadsadegh Khajueezadeh, Mohammad Shadnam Zarbil, Abolfazl Vahedi, Arash Kiyoumarsi, Hadi Tarzamni, and Jorma Kyyra. A comprehensive review on flywheel energy storage systems: Survey on electrical machines, power electronics converters, and control systems. *IEEE Access*, 11:81224 – 81255, 2023.

[135] Chao Zhang, Peng Cao, Rupeng Zhu, Weifang Chen, and Dan Wang. Dynamic modeling and analysis of the spline joint-flexible coupling-rotor system with misalignment. *Journal of Sound and Vibration*, 554, 2023.

[136] H.A.P. Silva and R. Nicoletti. Rotor vibration control using tilting-pad journal bearing with active pads — numerical and experimental results. *Journal of Sound and Vibration*, 546, 2023.

[137] Rajasekhara Reddy Mutra, J. Srinivas, D. Mallikarjuna Reddy, Muhamad Norhisham Abdul Rani, Mohd Azmi Yunus, and Zahrah Yahya. Dynamic and stability comparison analysis of the high-speed turbocharger rotor system with and without thrust bearing via machine learning schemes. *Journal of the Brazilian Society of Mechanical Sciences and Engineering*, 46(5), 2024.

[138] Chao Fu, Jean-Jacques Sinou, Weidong Zhu, Kuan Lu, and Yongfeng Yang. A state-of-the-art review on uncertainty analysis of rotor systems. *Mechanical Systems and Signal Processing*, 183, 2023.

[139] Dennis De Klerk, Daniel J Rixen, and Sven N Voormeeren. General framework for dynamic substructuring: history, review and classification of techniques. *AIAA journal*, 46(5):1169–1181, 2008.

[140] Maarten van der Seijs, Dennis De Klerk, and Daniel J Rixen. General framework for transfer path analysis: History, theory and classification of techniques. *Mechanical Systems and Signal Processing*, 68:217–244, 2016.

[141] Andy Moorhouse, Andy Elliott, and Joshua Meggitt. Virtual acoustic prototyping-a story of four decades.

[142] M Haeussler, D C Kobus, and D J Rixen. Parametric design optimization of e-compressor nvh using blocked forces and substructuring. *Mechanical Systems and Signal Processing*, 150:107217, 2021.

[143] Michael Kreutz, Francesco Trainotti, Verena Gimpl, and Daniel J Rixen. On the robust experimental multi-degree-of-freedom identification of bolted joints using frequency-based substructuring. *Mechanical Systems and Signal Processing*, 203:110626, 2023.

[144] Jure Korbar, Domen Ocepek, Gregor Čepon, and Miha Boltežar. Training artificial neural networks using substructuring techniques: Application to joint identification. *Mechanical Systems and Signal Processing*, 198:110426, 2023.

[145] Tim Vrtač, Domen Ocepek, Martin Česnik, Gregor Čepon, and Miha Boltežar. A hybrid modeling strategy for training data generation in machine learning-based structural health monitoring. *Mechanical Systems and Signal Processing*, 207:110937, 2024.

[146] Maarten V Van Der Seijs, Daniël D van den Bosch, Daniel J Rixen, and Dennis de Klerk. An improved methodology for the virtual point transformation of measured frequency response functions in dynamic substructuring. In *4th ECCOMAS thematic conference on computational methods in structural dynamics and earthquake engineering*, volume 4, 2013.

[147] EA Pasma, MV van der Seijs, SWB Klaassen, and MW van der Kooij. Frequency based substructuring with the virtual point transformation, flexible interface modes and a transmission simulator. In *Dynamics of Coupled Structures, Volume 4: Proceedings of the 36th IMAC, A Conference and Exposition on Structural Dynamics 2018*, pages 205–213. Springer, 2018.

[148] Jesús Ortega Almirón, Fabio Bianciardi, and Wim Desmet. Flexible interface models for force/displacement field reconstruction applications. *Journal of Sound and Vibration*, 534:117001, 2022.

[149] Francesco Trainotti, Tomaž Bregar, Steven W B Klaassen, and Daniel J Rixen. Experimental decoupling of substructures by singular vector transformation. *Mechanical Systems and Signal Processing*, 163:108092, 2022.

[150] JWR Meggitt and AT Moorhouse. A covariance based framework for the propagation of correlated uncertainty in frequency based dynamic sub-structuring. *Mechanical Systems and Signal Processing*, 136:106505, 2020.

[151] F Trainotti, M Haeussler, and DJ Rixen. A practical handling of measurement uncertainties in frequency based substructuring. *Mechanical Systems and Signal Processing*, 144:106846, 2020.

[152] Tomaž Bregar, Ahmed Mahmoudi, Miha Kodrič, Domen Ocepek, Francesco Trainotti, Miha Pogačar, Mert Göldeli, Gregor Čepon, Miha Boltežar, and Daniel Rixen. pyfbs: A python package for frequency based substructuring. *Journal of Open Source Software*, 7(69):3399, 2022.

[153] C Sullivan and Alexander Kaszynski. Pyvista: 3d plotting and mesh analysis through a streamlined interface for the visualization toolkit (vtk). *Journal of Open Source Software*, 4(37):1450, 2019.

[154] Alexander Kaszynski et al. Pyansys: Python interface to mapdl and associated binary and ascii files. *Zenodo. doi*, 10, 2020.

[155] Domen Ocepek, Francesco Trainotti, Gregor Čepon, Daniel J Rixen, and Miha Boltežar. On the experimental coupling with continuous interfaces using frequency based substructuring. *Mechanical Systems and Signal Processing*, 217:111517, 2024.

[156] Tomaž Bregar, Nikola Holeček, Gregor Čepon, Daniel J Rixen, and Miha Boltežar. Including directly measured rotations in the virtual point transformation. *Mechanical Systems and Signal Processing*, 141:106440, 2020.

[157] Steven W B Klaassen, Maarten V van der Seijs, and Dennis de Klerk. System equivalent model mixing. *Mechanical Systems and Signal Processing*, 105:90–112, 2018.

[158] Miha Kodrič, Gregor Čepon, and Miha Boltežar. Expansion of the dynamic strain field in 3d-printed structures using a hybrid modeling approach. *Measurement*, 206:112339, 2023.

[159] Miha Pogačar, Domen Ocepek, Francesco Trainotti, Gregor Čepon, and Miha Boltežar. System equivalent model mixing: A modal domain formulation. *Mechanical Systems and Signal Processing*, 177:109239, 2022.

[160] Christopher M Bishop and Nasser M Nasrabadi. *Pattern recognition and machine learning*, volume 4. Springer, 2006.

[161] I. Goodfellow. Deep learning, 2016.

[162] V.N. Vapnik, V. Vapnik, et al. Statistical learning theory. 1998.

[163] M. Hoffmann, M.K. Scherer, T. Hempel, A. Mardt, B. de Silva, B.E. Husic, S. Klus, H. Wu, J.N. Kutz, S. Brunton, and F. Noé. Deeptime: a python library for machine learning dynamical models from time series data. *Machine Learning: Science and Technology*, 2021.

[164] B. de Silva, K. Champion, M. Quade, J.-C. Loiseau, J. Kutz, and S. Brunton. PySINDy: A Python package for the sparse identification of nonlinear dynamical systems from data. *Journal of Open Source Software*, 5(49):2104, 2020.

[165] A.A. Kaptanoglu, B.M. de Silva, U. Fasel, K. Kaheman, A.J. Goldschmidt, J. Callaham, C.B. Delahunt, Z.G. Nicolaou, K. Champion, J.-C. Loiseau, J.N. Kutz, and S.L. Brunton. PySINDy: A comprehensive python package for robust sparse system identification. *Journal of Open Source Software*, 7(69):3994, 2022.

[166] S.L. Brunton, J.L. Proctor, and J.N. Kutz. Discovering governing equations from data by sparse identification of nonlinear dynamical systems. *Proceedings of the National Academy of Sciences*, 113(15):3932–3937, 2016.

[167] M. Raissi, P. Perdikaris, and G.E. Karniadakis. Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. *Journal of Computational Physics*, 378:686–707, 2019.

[168] F. Sun, Y. Liu, Q. Wang, and H. Sun. PiSL: Physics-informed Spline Learning for data-driven identification of nonlinear dynamical systems. *Mechanical Systems and Signal Processing*, 191:110165, 2023.

[169] S. Hedayatrasa, O. Fink, W. Van Paepegem, and M. Kersemans. k-space physics-informed neural network (k-pinn) for compressed spectral mapping and efficient inversion of vibrations in thin composite laminates. *Mechanical Systems and Signal Processing*, 223:111920, 2025.

[170] M. Haywood-Alexander and E. Chatzi. Physics-informed neural networks for one-step-ahead prediction of dynamical systems. *Structural Health Monitoring 2023*, 2023.

[171] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[172] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala. Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019.

[173] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[174] A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, and A. Lerer. Automatic differentiation in pytorch. 2017.

[175] C.E. Rasmussen. Gaussian processes in machine learning. In *Summer school on machine learning*, pages 63–71. Springer, 2003.

[176] GPy. GPy: A gaussian process framework in python. `http://github.com/SheffieldML/GPy`, since 2012.

[177] A.G. de G. Matthews, M. van der Wilk, T. Nickson, K. Fujii, A. Boukouvalas, P. León-Villagrá, Z. Ghahramani, and J. Hensman. GPflow: A Gaussian process library using TensorFlow. *Journal of Machine Learning Research*, 18(40):1–6, apr 2017.

[178] M. van der Wilk, V. Dutordoir, S.T. John, A. Artemev, V. Adam, and J. Hensman. A framework for interdomain and multioutput Gaussian processes. *arXiv:2003.01115*, 2020.

[179] J.R. Gardner, G. Pleiss, D. Bindel, K. Q Weinberger, and A.G. Wilson. Gpytorch: Blackbox matrix-matrix gaussian process inference with gpu acceleration. In *Advances in Neural Information Processing Systems*, 2018.

[180] K. Worden, William E. Becker, T.J. Rogers, and E.J. Cross. On the confidence bounds of gaussian process narx models and their higher-order frequency response functions. *Mechanical Systems and Signal Processing*, 104:188–223, 2018.

[181] R. Nayek, R. Fuentes, K. Worden, and E.J. Cross. On spike-and-slab priors for bayesian equation discovery of nonlinear dynamical systems via sparse linear regression. *Mechanical Systems and Signal Processing*, 161:107986, 2021.

[182] T.J. Rogers, K. Worden, and E.J. Cross. On the application of Gaussian process latent force models for joint input-state-parameter estimation: With a view to Bayesian operational identification. *Mechanical Systems and Signal Processing*, 140:106580, 2020.

[183] Assessment of alternative covariance functions for joint input-state estimation via Gaussian Process latent force models in structural dynamics. *Mechanical Systems and Signal Processing*, 213:111303, 2024.

[184] T. Simpson, N. Dervilis, and E. Chatzi. Machine learning approach to model order reduction of nonlinear systems via autoencoder and LSTM networks. *Journal of Engineering Mechanics*, 147(10):04021061, 2021.

[185] F. Dipietrangelo, F. Nicassio, and G. Scarselli. Structural health monitoring for impact localisation via machine learning. *Mechanical Systems and Signal Processing*, 183:109621, 2023.

[186] D.P. Kingma. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[187] C.R. Farrar, S.W. Doebling, and D.A. Nix. Vibration–based structural damage identification. *Philosophical Transactions of the Royal Society of London. Series A: Mathematical, Physical and Engineering Sciences*, 359(1778):131–149, 2001.

[188] N. Tsivouraki, S. Fassois, and K. Tserpes. On the random vibration-based progressive fatigue damage detection and classification for thermoplastic coupons under population and operational uncertainty. *Composite Structures*, 349:118524, 2024.

[189] G. Manson, K. Worden, and D. Allman. Experimental validation of a structural health monitoring methodology: Part III. Damage location on an aircraft wing. *Journal of Sound and Vibration*, 259(2):365–385, 2003.

[190] L.A. Bull, P.A. Gardner, J. Gosliga, T.J. Rogers, N. Dervilis, E.J. Cross, E. Papatheou, A.E. Maguire, C. Campos, and K. Worden. Foundations of population-based SHM, Part I: Homogeneous populations and forms. *Mechanical Systems and Signal Processing*, 148:107141, 2021.

[191] J. Gosliga, P.A. Gardner, L.A. Bull, N. Dervilis, and K. Worden. Foundations of population-based SHM, Part II: Heterogeneous populations–Graphs, networks, and communities. *Mechanical Systems and Signal Processing*, 148:107144, 2021.

[192] P. Gardner, L.A. Bull, J. Gosliga, N. Dervilis, and K. Worden. Foundations of population-based SHM, Part III: Heterogeneous populations–mapping and transfer. *Mechanical Systems and Signal Processing*, 149:107142, 2021.

[193] G. Tsialiamanis, C. Mylonas, E. Chatzi, N. Dervilis, D.J. Wagg, and K. Worden. Foundations of population-based SHM, Part IV: The geometry of spaces of structures and their feature spaces. *Mechanical Systems and Signal Processing*, 157:107692, 2021.

[194] D.S. Brennan, J. Gosliga, E.J. Cross, and K. Worden. Foundations of population-based SHM, Part V: Network, framework and database. *Mechanical Systems and Signal Processing*, 223:111602, 2025.

[195] C. Cortes. Support-Vector Networks. *Machine Learning*, 1995.

[196] A.J. Hughes, L.A. Bull, P. Gardner, R.J. Barthorpe, N. Dervilis, and K. Worden. On risk-based active learning for structural health monitoring. *Mechanical Systems and Signal Processing*, 167:108569, 2022.

[197] D.P. Kingma. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

[198] C. Mylonas, I. Abdallah, and E. Chatzi. Conditional variational autoencoders for probabilistic wind turbine blade fatigue estimation using Supervisory, Control, and Data Acquisition data. *Wind Energy*, 24(10):1122–1139, 2021.

[199] T. Simpson, K. Vlachas, A. Garland, N. Dervilis, and E. Chatzi. VpROM: a novel variational autoencoder-boosted reduced order model for the treatment of parametric dependencies in nonlinear systems. *Scientific Reports*, 14(1):6091, 2024.

[200] Nicholas E. Silionis, Theodora Liangou, and Konstantinos N. Anyfantis. Deep learning-based surrogate models for spatial field solution reconstruction and uncertainty quantification in structural health monitoring applications. *Computers & Structures*, 301:107462, 9 2024.

[201] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio. Generative adversarial networks. *Communications of the ACM*, 63(11):139–144, 2020.

[202] G. Tsialiamanis, M.D. Champneys, N. Dervilis, D.J. Wagg, and K. Worden. On the application of generative adversarial networks for nonlinear modal analysis. *Mechanical Systems and Signal Processing*, 166:108473, 2022.

[203] L.A. Bull, P.A. Gardner, N. Dervilis, and K. Worden. Normalising flows and nonlinear normal modes. *IFAC-PapersOnLine*, 54(7):655–660, 2021.

[204] Bernard W Silverman. *Density estimation for statistics and data analysis*. Routledge, 2018.

[205] M. Mirza. Conditional generative adversarial nets. *arXiv preprint arXiv:1411.1784*, 2014.

[206] M. Arjovsky, S. Chintala, and L. Bottou. Wasserstein generative adversarial networks. In *International Conference on Machine Learning*, pages 214–223. PMLR, 2017.

[207] D. Rezende and S. Mohamed. Variational inference with normalizing flows. In *International Conference on Machine Learning*, pages 1530–1538. PMLR, 2015.

[208] L. Dinh, D. Krueger, and Y. Bengio. Nice: Non-linear independent components estimation. *arXiv preprint arXiv:1410.8516*, 2014.

[209] V. Stimper, D. Liu, A. Campbell, V. Berenz, L. Ryll, B. Schölkopf, and J. M. Hernández-Lobato. normflows: A PyTorch Package for Normalizing flows. *Journal of Open Source Software*, 8(86):5361, 2023.

[210] Philip Matthew Daborn. *Smarter dynamic testing of critical structures*. PhD thesis, University of Bristol, 2014.

[211] Phil M Daborn, PR Ind, and DJ Ewins. Enhanced ground-based vibration testing for aerodynamic environments. *Mechanical Systems and Signal Processing*, 49(1-2):165–180, 2014.

[212] Randall L Mayes and Daniel P Rohe. Physical vibration simulation of an acoustic environment with six shakers on an industrial structure. In *Shock & Vibration, Aircraft/Aerospace, Energy Harvesting, Acoustics & Optics, Volume 9: Proceedings of the 34th IMAC, A Conference and Exposition on Structural Dynamics 2016*, pages 29–41. Springer, 2016.

[213] David O Smallwood. Multiple shaker random control with cross coupling. *Combined Environments: Technology Interrelations*, pages 341–347, 1978.

[214] Randy Mayes, Luke Ankers, Phil Daborn, Tony Moulder, and Philip Ind. Optimization of shaker locations for multiple shaker environmental testing. *Experimental Techniques*, 44:283–297, 2020.

[215] GD Nelson, DP Rohe, and RA Schultz. Strategies for shaker placement for impedance-matched multi-axis testing. *IMAC XXXVII, Orlando, FL*, 2019.

[216] Brian C Owens, Randall L Mayes, Moheimin Khan, D Gregory Tipton, and Brandon Zwink. Flight environments demonstrator: Part i—experiment design and test planning. In *Sensors and Instrumentation, Aircraft/Aerospace, Energy Harvesting & Dynamic Environments Testing, Volume 7: Proceedings of the 37th IMAC, A Conference and Exposition on Structural Dynamics 2019*, pages 323–333. Springer, 2020.

[217] Christopher Beale, Ryan Schultz, Chandler Smith, and Timothy Walsh. Degree of freedom selection approaches for mimo vibration test design. In Matt Allen, Sheyda Davaria, and R. Benjamin Davis, editors, *Special Topics in Structural Dynamics & Experimental Techniques, Volume 5*, pages 81–90, Cham, 2023. Springer International Publishing.

[218] Moheimin Khan and Tyler Schoenherr. Evaluating degree of freedom selection methods for mimo vibration modeling. In Tyler Schoenherr, Alexandra Karlicek, and Dagny Beale, editors, *Dynamic*

Environments Testing, Vol. 7: Proceedings of the 42nd IMAC, A Conference and Exposition on Structural Dynamics 2024, pages 47–54, Cham, 2024. Springer Nature Switzerland.

[219] Jerome S Cap, Shantisa Norman, and David O Smallwood. The derivation of multiple-input-multiple output (mimo) acoustic test specifications to simulate a missile flight. Technical report, Sandia National Lab.(SNL-NM), Albuquerque, NM (United States); Sandia . . . , 2016.

[220] Umberto Musella, Giacomo D'Elia, Simone Manzato, Bart Peeters, Patrick Guillaume, and Francesco Marulo. Analyses of target definition processes for mimo random vibration control tests. In *Special Topics in Structural Dynamics, Volume 6: Proceedings of the 35th IMAC, A Conference and Exposition on Structural Dynamics 2017*, pages 135–148. Springer, 2017.

[221] Ryan Schultz and Garrett Nelson. Techniques for modifying mimo random vibration specifications. In *Sensors and Instrumentation, Aircraft/Aerospace and Dynamic Environments Testing, Volume 7: Proceedings of the 40th IMAC, A Conference and Exposition on Structural Dynamics 2022*, pages 71–83. Springer, 2022.

[222] Odey Yousef, Fernando Moreu, and Arup Maji. The exact error prediction method for mimo controlled tests. *Mechanical Systems and Signal Processing*, 223:111877, 2025.

[223] Ryan Schultz. Demonstration of output weighting in mimo control. In *Society for Experimental Mechanics Annual Conference and Exposition*, pages 141–149. Springer, 2023.

[224] Levi H Manring, John F Schultze, Sandra J Zimmerman, and Brian P Mann. Improving convergence of the matrix power control algorithm for random vibration testing. *Mechanical Systems and Signal Processing*, 182:109574, 2023.

[225] Celia Hameury, Giovanni Ferrari, Giulio Franchini, and Marco Amabili. An experimental approach to multi-input multi-output nonlinear active vibration control of a clamped sandwich beam. *Mechanical Systems and Signal Processing*, 216:111496, 2024.

[226] Siyu Ren, Huaihai Chen, and Ronghui Zheng. Multi-shaker time-varying root mean square nonstationary random vibration environmental control test. *Journal of Sound and Vibration*, 593:118647, 2024.

[227] Yi Ma, Huaihai Chen, and Ronghui Zheng. Control strategy for multi-axial swept sine on random mixed vibration testing. *Journal of Sound and Vibration*, 527:116846, 2022.

[228] PM Daborn. Scaling up of the impedance-matched multi-axis test (immat) technique. In *Shock & Vibration, Aircraft/Aerospace, Energy Harvesting, Acoustics & Optics, Volume 9: Proceedings of the 35th IMAC, A Conference and Exposition on Structural Dynamics 2017*, pages 1–10. Springer, 2017.

[229] Giacomo D'Elia, Umberto Musella, Emiliano Mucchi, Patrick Guillaume, and Bart Peeters. Analyses of drives power reduction techniques for multi-axis random vibration control tests. *Mechanical Systems and Signal Processing*, 135:106395, 2020.

[230] Ethan Cramer, Dustin Harvey, and Richard Zhang. Rapid, approximate multi-axis vibration testing. In Julie Harvie, editor, *Dynamic Environments Testing, Volume 7*, pages 91–100, Cham, 2024. Springer Nature Switzerland.

[231] Ryan Schultz. Utilizing under-determined solutions for mimo vibration control. In Tyler Schoenherr, Alexandra Karlicek, and Dagny Beale, editors, *Dynamic Environments Testing, Vol. 7: Proceedings of the 42nd IMAC, A Conference and Exposition on Structural Dynamics 2024*, pages 71–77, Cham, 2024. Springer Nature Switzerland.

[232] David O Smallwood and Thomas L Paez. A frequency domain method for the generation of partially coherent normal stationary time domain signals. *Shock and vibration*, 1(1):45–53, 1993.

[233] Ryan A Schultz and Garrett D Nelson. Input signal synthesis for open-loop multiple-input/multiple-output testing. In *Sensors and Instrumentation, Aircraft/Aerospace, Energy Harvesting & Dynamic Environments Testing, Volume 7: Proceedings of the 37th IMAC, A Conference and Exposition on Structural Dynamics 2019*, pages 245–257. Springer, 2020.

[234] Daniel Rohe, Ryan Schultz, and Norman Hunter. Rattlesnake: An open-source multi-axis and combined environments vibration controller. Technical report, Sandia National Lab.(SNL-NM), Albuquerque, NM (United States), 2021.

[235] Daniel Rohe. Sdynpy: A structural dynamics python library. Technical report, Sandia National Lab.(SNL-NM), Albuquerque, NM (United States), 2022.

[236] Diego M. Tuozzo, Olavo M. Silva, Lucas V.Q. Kulakauskas, Jacson G. Vargas, and Arcanjo Lenzi. Time-harmonic analysis of acoustic pulsation in gas pipeline systems using the finite element transfer matrix method: Theoretical aspects. *Mechanical Systems and Signal Processing*, 186:109824, 2023.

[237] Ye Tang, Hu-Jie Zhang, Li-Qun Chen, Qian Ding, Qiuyang Gao, and Tianzhi Yang. Recent progress on dynamics and control of pipes conveying fluid. *Nonlinear Dynamics*, 112(10):8431–8451, October 2024.

[238] Shulian Liu, Cunkai Gu, Qiuhui Tian, Dongliang Huang, and Yizhe Guo. Research on the vibration mechanism of compressor complex exhaust pipeline based on transient flow. *International Journal of Hydrogen Energy*, 2024.

[239] Reciprocating compressors for petroleum, chemical, and gas industry services. API 618 - 5th edition, American Petroleum Institute (API), Washington DC, United States, dec 2007.

[240] Avneet Singh, Justin Hollingsworth, Terry Kreuz, Karl Wygant, Rainer Kurz, and Kelsi Katcher. Chapter 6 - transport of natural gas. In Klaus Brun, Tim Allison, Rainer Kurz, and Karl Wygant, editors, *Energy Transport Infrastructure for a Decarbonized Economy*, pages 237–289. Elsevier, 2025.

[241] *Guidelines for the avoidance of vibration induced fatigue failure in process pipework*. Energy Institute, London, UK, 2nd edition edition, January 2008.

[242] Chaoqian Chen Andrii Babenko Zisheng Xu Jiancheng Cai, Minghan Hu and Shiju E. Numerical study of the pulsatile flow and aeroacoustics of straight and curved pipes. *Numerical Heat Transfer, Part A: Applications*, 85(12):1903–1921, 2024.

[243] Ze-Chao Wang, Wang-Ji Yan, and Ka-Veng Yuen. A transfer matrix method-based closed-form solution of sensitivities of dynamic properties and frf for multi-span pipes under complex boundary conditions. *Mechanical Systems and Signal Processing*, 198:110428, 2023.

[244] Jiang hai Wu, Arris S. Tijsseling, and Yu dong Sun. Vibration analysis by impedance synthesis method of three-dimensional piping connected to a large circular cylindrical shell. *Mechanical Systems and Signal Processing*, 188:110063, 2023.

[245] M.L. Munjal. *Acoustics of Ducts and Mufflers*. Wiley, 2014.

[246] Teresa Bravo and Cédric Maury. Converging rainbow trapping silencers for broadband sound dissipation in a low-speed ducted flow. *Journal of Sound and Vibration*, 589:118524, 2024.

[247] Paulo H. Mareze, Olavo M. Silva, William D'A. Fonseca, Eric Brand ao, and Luís Godinho. Optimization of acoustic porous material absorbers modeled as rigid multiple microducts networks:

Metamaterial design using additive manufacturing. *Journal of Sound and Vibration*, 596:118739, 2025.

[248] H. Tijdeman. On the propagation of sound waves in cylindrical tubes. *Journal of Sound and Vibration*, 39(1):1 – 33, 1975.

[249] M. S. Howe. The damping of sound by wall turbulent shear layers. *The Journal of the Acoustical Society of America*, 98(3):1723–1730, 1995.

[250] year Peters, M. C. A. M. and Hirschberg, A. and Reijnen, A. J. and Wijnands, A. P. J. Damping and reflection coefficient measurements for an open pipe at low mach and low helmholtz numbers. *Journal of Fluid Mechanics*, 256:499–534.

[251] Olavo Silva, D Tuozzo, J Vargas, L Kulakauskas, A Fernandes, J Souza, A Rocha, Arcanjo Lenzi, R Timbó, C Mendonça, and A Brandão. Numerical modelling of low-frequency acoustically induced vibration in gas pipeline systems. In *Proc. 29th International Conference on Noise and Vibration engineering (ISMA2020) - At: Online - Virtual Plataform*, 09 2020.

[252] A. Boresi, K. Chong, and J. D. Lee. *Elasticity in Engineering Mechanics*. Wiley, 2010.

[253] Olavo M. Silva, Jacson G. Vargas, Diego M. Tuozzo, Lucas V. Q. Kulakauskas, Ana P. Rocha, Andre F. Fernandes, and Jose L. Souza. Openpulse: Open source software for pulsation analysis of pipeline systems. `https://open-pulse.github.io/OpenPulse/`, 2024.

[254] Riverbank Computing Limited. *PyQt5 Documentation*, 2024. Accessed: 2024-11-23.

[255] Will Schroeder, Ken Martin, and Bill Lorensen. *The Visualization Toolkit: An Object-Oriented Approach to 3D Graphics*. Kitware, Inc., 4th edition, 2006. Accessed: 2024-11-23.

[256] Christophe Geuzaine and Jean-François Remacle. *Gmsh: A 3D finite element mesh generator with built-in pre- and post-processing facilities*, 2009. Accessed: 2024-11-23.

[257] Charles R. Harris, K. Jarrod Millman, Stéfan J. van der Walt, et al. Array programming with numpy. *Nature*, 585:357–362, 2020.

[258] Intel Corporation. *PyPardiso Project*, 2024. Accessed: 2024-11-23.

[259] John D. Hunter. Matplotlib: A 2d graphics environment. *Computing in Science & Engineering*, 9(3):90–95, 2007.

[260] Wes McKinney et al. pandas: a foundational python library for data analysis and manipulation. *Journal of Open Source Software*, 1(5):1–6, 2010.

[261] Andrew Collette. *HDF5 for Python*, 2024. Accessed: 2024-11-23.

[262] Poetry Authors. *Poetry - Python dependency management and packaging made easy*, 2024. Accessed: 2024-11-23.

[263] Eric W. Lemmon, Ian H. Bell, Marcia L. Huber, and Mark O. McLinden. *REFPROP: Reference Fluid Thermodynamic and Transport Properties*. National Institute of Standards and Technology (NIST), Gaithersburg, MD, USA, 10.0 edition, 2018. Accessed: 2024-11-23.

[264] Klemen Zaletelj, Tomaž Bregar, Domen Gorjup, and Janko Slavič. ladisk/pyema: v0.24, 2020.

[265] pyfrf. https://pypi.org/project/pyFRF/, 2022.

[266] Aleš Zorman, Domen Gorjup, and Janko Slavič. ladisk/pyexsi: Release of the version v0.4, 2021.