University of Sheffield | Mechanical Engineering

MEng Mechanical Engineering

Developing an Open-Source Frequency Domain Modal Analysis Algorithm in Python

Adam AWADALLA

May 2025

Timothy J. Rogers

Report submitted to the University of Sheffield in partial fulfillment of the requirements for the degree of Master of Engineering

Word Count: $\sim$ 7500

Contents

Acknowledgements

I would like to express my deepest gratitudes to my primary supervisor, Dr. Tim Rogers, and my secondary supervisors, Dr. Max Champneys and Dr. Brandon O'Connel, for all of their unwavering support, guidance, and understanding throughout this project. Thanks also to my partner, Katie, without whom I would not have submitted this work. Finally, my Father, for supporting me, his financial burden.

To everyone mentioned here, and to all those who have contributed—whether through discussion, critique, or moral support—I offer my heartfelt thanks.

Abstract

Structural dynamics relies on modal analysis to characterise vibratory behaviour in engineering systems, yet commercial software obscures underlying algorithms and imposes high licensing costs. This work aims to deliver a transparent, open-source alternative by implementing the polyreference least-squares complex frequency-domain estimator (pLSCF, PolyMAX) and a least-squares frequency-domain (LSFD) mode-shape routine in Python. Methods include a NumPy/SciPy - based architecture, rigorous unit- and integration-testing, PEP-8 documentation, and model-order-reduction via matrix truncation and singular-value decomposition to accelerate computation. Results on a ten-degree-of-freedom benchmark show natural frequencies and damping ratios within $9 \times 10^{-5}\%$ of analytical values; mode shapes reach modal-assurance-criterion scores $\geq$ 0.997. Computational time is reduced by up to two orders of magnitude relative to brute-force fitting, and accuracy is retained under 25 dB signal-to-noise conditions after peak-segmentation pre-processing. The study concludes that the presented library supplies a robust, high-performance, and freely accessible alternative to proprietary tools while providing a foundation for future enhancements such as MAC-based stability criteria, PolyMAX Plus noise handling, and an interactive graphical interface.

Chapter 1

Introduction

The theoretical and experimental study of structural dynamics has regularly helped engineers grasp the behaviour of systems and structures encountered in everyday life, and has consequently aided in making them safer, lighter, and greener. This ranges from the design of aeroplanes and cars to the stability of buildings as well as the functionality of household appliances like washing machines or air conditioners. Hence, engineers and researchers have been consistently striving to further understand and predict the dynamic characteristics of these various structures. This understanding is crucial for ensuring safety and compliance, performance, and reliability. [1–3]

1.1 Modal Analysis

1.1.1 Theoretical Overview

In linear structural dynamics, Modal Analysis is universally recognized as the pre-eminent solution for the identifying and characterizing structures or systems. It achieves this by studying the structure's modal properties or parameters — its natural frequencies, mode shapes, and damping ratios. Whether using mathematical modelling or experimental testing, modal analysis is key for engineers to understand the response of structures to various excitations or forces, ensuring safety, compliance with standards and regulations, and supporting many research areas such as Structural Health Monitoring or System Identification. [4–6]

Fundamentally, modal analysis is the decomposition of the complex oscillatory behaviour of structures into smaller, more intuitive components called modes. Each mode is a mathematical representation of a specific vibration pattern associated with a natural frequency, the frequency (or set of) at which a system tends to oscillate when displaced, and a corresponding mode shape, a vector which describes the relative movement among the degrees-of-freedom. The damping ratio, a unitless parameter, quantifies the energy dissipation of the system for each mode. The modal properties are defined by the interaction of the system's physical properties, its mass, stiffness, and damping. These inherent properties guide the system's vibration when subject to external forcing or initial displacements or velocities.

1.1.2 Modal Analysis in practice, academia, and industry

There are two mainstream modal analysis procedures which address the need for studying the modal properties: Numerical Modal Analysis, and Experimental Modal Analysis, often referred to as Modal Testing. Numerical Modal Analysis is used to simulate dynamic behaviour when modal testing is unimplementable or unnecessary. It involves discretization — typically using finite element modelling — and solving the resulting equations of motion under appropriate initial and boundary conditions. Numerical Modal Analysis offers a faster means of evaluating simpler systems without the need for experimental testing.

In contrast, Modal Testing relies on the principle that, in a mostly linear system, the same modal parameters used to predict the system's response can be obtained from a measurement of that response. It utilizes various experimental methods such as shaker testing or impact testing to gather response data using sensors like accelerometers or laser vibrometers. One would pair the experimental set-up with computational algorithms to extract the modal parameters from the test data. An example set-up for a modal test using a shaker is showcased in Figure 1.1. Modal Testing is often the first step for applications such validating models, or measuring the effect of environmental and operational conditions on structures. This underscores why modal testing is typically preferred and more prevalent, as it directly captures the system's response in a way that numerical modal analysis is unable to. Due to the widespread adoption of modal testing in structural dynamics, the term modal analysis is generally interpreted as modal testing, and due to the nature of the work in this report, the term modal analysis will also be referring to modal testing.

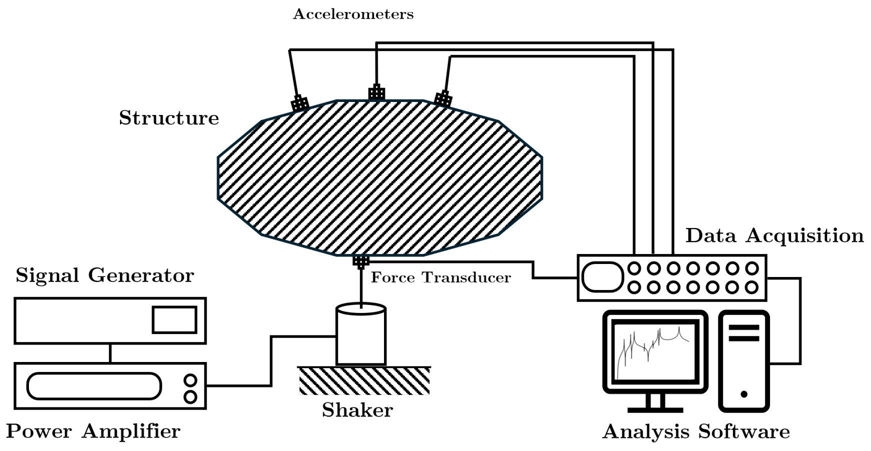


Figure 1.1: Typical electrodynamic shaker experimental set up, adapted from He & Fu [5]

In modern research areas such as structural health monitoring (SHM)—where the primary objective is to observe changes in an asset's structural condition through continuous monitoring—modal analysis is frequently employed because a system's modal parameters are highly sensitive to its physical state [7]. Natural frequencies and mode shapes are considered damage-sensitive features; whether changes result from cracks, corrosion, environmental effects, or fatigue, any structural alteration will modify the physical parameters and lead to a quantifiable change in the modal parameters. By statistically comparing a structure's current modal parameters with its baseline measurements, taken when the structure is in good condition, early signs of damage and degradation can be detected. This approach enables the effective monitoring of essential infrastructure, including bridges [8–10], buildings [11], and wind turbines [12, 13], making modal analysis an invaluable tool for ensuring the safety and longevity of many everyday structures. See Figure 1.2 for examples of modal testing in SHM contexts for civil and structural engineering.

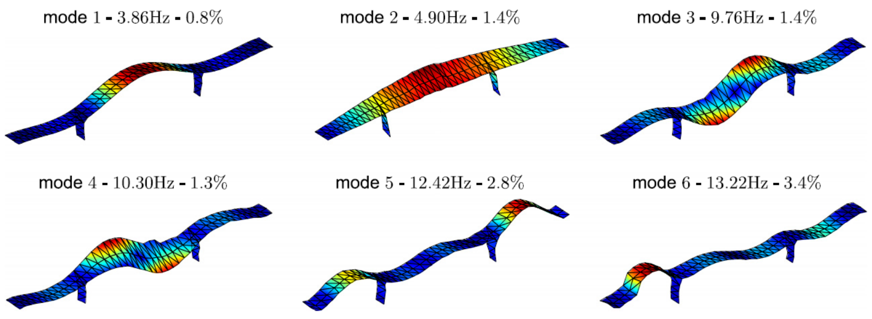


Figure 1.2: Model analysis results of the Z24 bridge where Maeck et al. in [9, 10, 14] performed an experimental campaign on a bridge in Switzerland to benchmark vibration based techniques for damage identification

Civil and structural engineering practices apply modal analysis to predict responses to seismic, wind, wave, and traffic loads—thereby enhancing safety and extending service life [10, 15–17]. In automotive applications, it's used to optimize lightweight, high-strength vehicles and improve NVH performance (e.g., gear noise, road harshness) [18–22], and aerospace relies on it to balance structural integrity with weight savings through vibration control and wind/fluid load assessment [4, 5, 23–25]. See Figure 1.3 for a modal test on a model aircraft.

Figure 1.3: Peeters et al. [26], performed modal tests on wind-tunnel model and full sized F16 aircraft

1.1.3 Mathematical Basis

The standard procedure for modal analysis begins with forming the equations of motion (EOMs) that represent a system, typically, second order matrix differential equations. Consider a system with an arbitrary number N, degrees-of-freedom (DOFs) as shown in Figure 1.4. Using Newton's second law [27],

$$\sum F_i = m_i \ddot{\mathbf{x}}_i$$

Where F_i is a force acting on the i-th DOF, m_i is the mass, and $\mathbf{x}_i$ is the coordinate, or alternatively the Euler-Lagrange equation as such [27]:

$$\frac{d}{dt}\left(\frac{\partial T}{\partial \dot{q}_i}\right) - \frac{\partial T}{\partial q_i} + \frac{\partial U}{\partial q_i} = Q_i$$

Where q_i is a generalized coordinate (take $q_i = x_i$ for this system), T is the system's kinetic energy, U is the potential energy, and Q_i represents the non-conservative forces. The resulting equations of motion for the system are given in Equation 1.1.

$$\begin{aligned} m_1\ddot{x}_1 + (c_1 + c_2)\dot{x}_1 - c_2\dot{x}_2 + (k_1 + k_2)x_1 - k_2x_2 &= F_1 \\ m_2\ddot{x}_2 + (c_2 + c_3)\dot{x}_1 - c_2\dot{x}_1 - c_3\dot{x}_3 + (k_2 + k_3)x_2 - k_2x_1 - k_3x_3 &= F_2 \\ \dots \\ m_N\ddot{x}_N + (c_N + c_{N+1})\dot{x}_N - c_{N-1}\dot{x}_{N-1} + (k_N + k_{N+1})x_N - k_{N-1}x_{N-1} &= F_N \end{aligned} \tag{1.1}$$

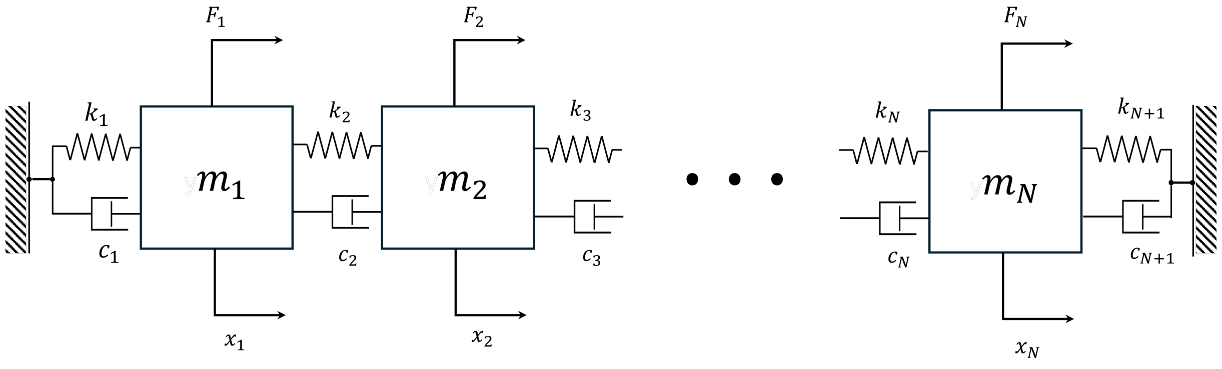


Figure 1.4: A "lumped-mass " system of N degrees-of-freedom

By assembling the physical parameters into respective matrices as highlighted in Equation 1.2, the characteristic differential equation of the system is obtained:

$$\begin{aligned} [\mathbf{K}] &= \begin{bmatrix} k_1 + k_2 & -k_2 & 0 & \dots & 0 \\ -k_2 & k_2 + k_3 & -k3 & \dots & 0 \\ 0 & -k_3 & k_3 + k_4 & -k_4 & 0 \\ \dots & \dots & \dots & \ddots & \dots \\ 0 & 0 & 0 & -k_N & k_N + k_{N+1} \end{bmatrix} \\ [\mathbf{C}] &= \begin{bmatrix} c_1 + c_2 & -c_2 & 0 & \dots & 0 \\ -c_2 & c_2 + c_3 & -c3 & \dots & 0 \\ 0 & -c_3 & c_3 + c_4 & -c_4 & 0 \\ \dots & \dots & \dots & \ddots & \dots \\ 0 & 0 & 0 & -c_N & c_N + c_{N+1} \end{bmatrix} \\ [\mathbf{M}] &= \mathbf{diag}(m_i) \qquad \forall i = 1, 2, \dots, N \end{aligned} \tag{1.2}$$

$$\therefore [\mathbf{M}]\ddot{\mathbf{x}} + [\mathbf{C}]\dot{\mathbf{x}} + [\mathbf{K}]\mathbf{x} = \mathbf{F} \tag{1.3}$$

For simplicity, assume the system presented vibrates freely, and is undamped, so $\mathbf{F}$ and $[\mathbf{C}]$ are zero. If the solution of the presented differential equation is harmonic, i.e. $\mathbf{x}(t) = \Psi e^{j\omega t}$, one finds that equation 1.3 reduces to:

$$([\mathbf{K}] - \omega^2[\mathbf{M}])\Psi = 0 \tag{1.4}$$

Rearranging the terms, the equation into takes the general form of an eigenvalue problem:

$$\begin{aligned} [\mathbf{A}] &= [\mathbf{M}]^{-1}[\mathbf{K}] \\ [\mathbf{A}]\Psi &= \omega^2\Psi \end{aligned} \tag{1.5}$$

Here, ω^2 is the eigenvalue of matrix $\mathbf{A}$ — often referred to as an *eigenfrequency* — and Ψ is the corresponding eigenvector representing the mode shape.

The process of reducing the governing equations into eigenvalue problems is mathematically elegant, robust, and applicable to all linear systems. Despite this approach's elegance, its practical applications rarely exist, as real world structures and systems seldom conform to the idealized assumptions of lumped-mass systems. Regardless, the superposition principle that any linear system's motion can be written as a sum of its natural modes remains the cornerstone of modern experimental modal analysis and system-identification techniques. When a structure is excited, its physical coordinates can be expanded as a truncated series of modal contributions. As in practice, the truncation keeps a manageable number of the most significant or contributing modes. Theoretically, this modal sum can reproduce the full measured response [4–6].

1.2 Computation in engineering, and the software engineering industry

Building on the earlier discussion that highlighted the integration of computation with modal testing, it is evident that the engineering industry is heavily reliant on software and computational methods. This reliance arises from the extensive volumes of data, such as laboratory results or continuous monitoring outputs, that require processing, as well as from the complexity of calculations that are impractical to perform manually. Common examples include the use of advanced 3D modelling and drawing software, such as Fusion and SolidWorks, programming for data analysis using languages like Python, MATLAB or C++, and various simulation tools like Simulink, ANSYS, or OpenFOAM. It is a safe assumption that software usage is indispensable to engineering practices.

Multiple classifications, interpretations, and naming conventions exist among software developers regarding software categorization. To avoid ambiguity in this report, the terms ”proprietary software ” and ”open-source software ” are defined explicitly. Open-source software (OSS) refers to software licensed to allow free use, modification, and redistribution of both the software and its source code. In contrast, proprietary software denotes software tools or programming environments that require the purchase of a licence for use and restrict access to the source code, thereby preventing modification and redistribution. Previous work [28] presented a more comprehensive comparison of the two software distribution paradigms. To avoid redundancy, only a brief summary of those findings is provided here, followed by an overview of the conclusions and the implications for future work.

The earliest efforts in software commercialization demonstrated considerable foresight, causing a widespread adoption of proprietary tools in engineering industries. These tools gained traction due to rigorous testing, continuous support, adherence to industry standards, and user-friendly interfaces that reduced errors and enhanced usability. Conversely, inherent drawbacks such as high costs, usage restrictions, lack of source code access, potential obsolescence, and privacy concerns have emerged. In contrast, open-source alternatives offer cost-free access, greater user control, and community-driven support, though they often face challenges with limited testing and installation complexity. As a result, selecting between open-source and proprietary solutions requires a careful evaluation of these trade-offs, balancing reliability and robustness against cost and flexibility.

1.3 Project Scope

The pivotal role that modal analysis plays in advancing engineering fields provides motivation for this project. As highlighted in Section 1.1.2, modal analysis is essential for ensuring the comfort, performance, and safety of vehicles and aircraft, as well as for safeguarding critical infrastructure such as buildings, bridges, railway tracks, and offshore structures. Despite its widespread application, many engineers and researchers rely on software tools to perform modal analysis without fully understanding the underlying theory and algorithms, which hinders their ability to interpret results or adapt the methods for unique challenges that are presented quite frequently. This reliance has enabled companies, e.g. Siemens or Structural Vibration Solutions, to charge exorbitant prices for software licences — a situation that thrives in a market dominated by a monopoly of robust solutions. To address this gap, the project introduces an accessible, well-documented, and rigorously tested open-source software tool that produces accurate results, elucidates the fundamental theory, and permits customization for novel applications. The development of a Python library for modal analysis represents the first robust and reliable open-source solution in this domain.

1.3.1 Aim and Objectives

This project is part of a broader development initiative aimed at developing and providing modal-analysis open-source software, making it essential to establish a clear aim that outlines specific objectives and goals. Accordingly, this project contributes to the initiative by integrating and testing the polyreference least-squares complex frequency domain modal parameter estimator (pLSCF, or PolyMAX© commercially) in a Python library. Modal Analysis assists in engineering design decisions for compliance standards, and various research contexts, such as structural health/condition monitoring, and system identification. The software aims reduce uncertainty in data interpretation, allowing engineers and researchers to direct their focus on efficient experimental design and testing. This aim can be effectively achieved through the following objectives, which provide a structured approach to achieving a full implementation of pLSCF in the library.

- Research the pLSCF method, and obtain a complete algorithmic understanding of the method for implementation.
- Integrate pLSCF in the open-source Python library.
- Perform unit tests on the algorithm's individual functions and classes, and test the complete algorithm using simulated and experimental data as a benchmark for the algorithm's efficiency and consistency.
- Perform the same objectives for the Least-Squares Frequency Domain method, which is a simpler, supplementary algorithm used to estimate the mode shapes.
- Help in additional tasks, like signal processing and conditioning, algorithm optimization, and providing documentation for users, which will contribute to the usability and robustness of the published software.

Chapter 2

Software Development

This chapter aims to outline the methodologies implemented to achieve the objectives of the project. It begins by presenting the mathematical and scientific foundations utilised throughout the development process, including any novel approaches specifically developed for this work. Additionally, it discusses key challenges and considerations encountered throughout the development process, providing clarity on the decisions made to address these issues

2.1 Modal Analysis Algorithms: mathematical overview

2.1.1 Frequency Response Functions and *poles*

The process for obtaining the modal parameters presented in Section 1.1.1 represents a complete solution to the system depicted in Figure 1.4 under unforced conditions. In practical applications, however, an input force is applied to elicit the system response from which the modal parameters are then derived. The response of the structure is most commonly captured using a numeric construct called the Frequency Response Function (FRF). An FRF is a transfer function, which represents the ratio between a structure's response to the input excitation forces in the frequency-domain [5,29,30]. Revisiting the system's differential equation to include both damping and an external force transforms Equation 1.4 into

$$([\mathbf{K}] + j\nu[\mathbf{C}] - \nu^2[\mathbf{M}])\mathbf{X} = \mathbf{F} \tag{2.1}$$

Where ν denotes the frequency of the excitation force $\mathbf{F}$, and $\mathbf{X}$ represents the system's response. Consequently, the response can be expressed as

$$\mathbf{X} = [H(\nu)]\mathbf{F} \tag{2.2}$$

with the FRF matrix defined as $H(\nu) = ([\mathbf{K}] + j\nu[\mathbf{C}] - \nu^2[\mathbf{M}])^{-1}$. According to the derivation provided in Appendix A.1, the FRF correlating the i^{th} output to the j^{th} input is represented using the system's modal parameters as

$$H_{ij}(\nu) = \frac{X_i}{F_j} = \sum_{k=1}^{N}\left(\frac{\varphi_{ik}\varphi_{jk}}{\lambda_k^2 - \nu^2}\right) \tag{2.3}$$

In the free vibration analysis of a damped system, the eigenvalue problem yields the system poles, which are complex numbers encapsulating both the natural frequencies and damping ratios. In the undamped case, these poles reduce to the natural frequencies of the system (set $\zeta = 0$). For each mode, the conjugate pair of poles is given by

$$\lambda_i, \lambda_i^* = -\zeta_i \omega_i \pm j\omega_i \sqrt{1 - \zeta_i^2} \tag{2.4}$$

allowing the natural frequencies and damping ratios to be determined from

$$\begin{aligned} \omega_i &= |\lambda_i| \\ \zeta_i &= \frac{\Re(\lambda_i)}{\omega_i} \end{aligned} \tag{2.5}$$

In practice, most deterministic modal analysis algorithms yield a set of system poles that encapsulate the structure's dynamic characteristics. These poles are evaluated against specific stability criteria to determine their physical relevance. Stable poles, which meet the required conditions, represent valid dynamic modes, while unstable poles may indicate unphysical results or simply artefacts of overfitted models.

Stabilisation Diagrams

Practitioners typically assess the stability of poles using a stabilisation diagram. An example stabilisation plot is shown in Figure 2.1.

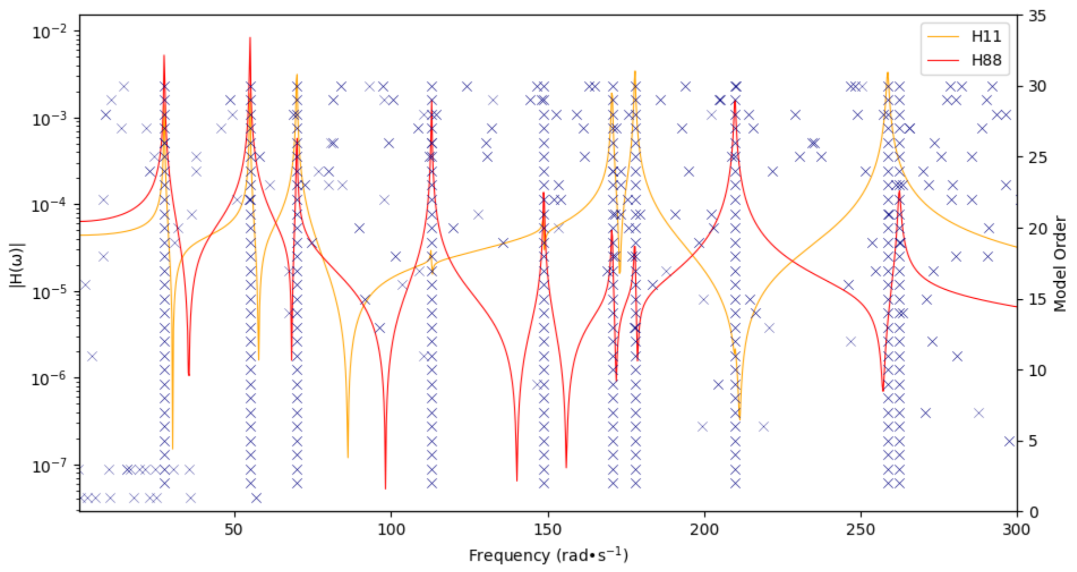


Figure 2.1: Stabilisation diagram for a 10 degree-of-freedom oscillator, for model orders between 1 and 30. Poles are marked as 'X' on the plot.
H_{11} denotes the FRF of x_1 and f_1, same for H_{88}.

In modal curve-fitting, algorithms typically fit the analytical functions based on an expected number of modes, or model order. Stabilisation diagrams are used to find the model order which provides the most stable poles based on the stability criteria. This is done by finding the poles for all the model orders within a desired range and plotting the stability diagram. For linear systems, a stability criteria can look like:

1. $Re(\lambda) < 0$, a positive real part of a root means that either the natural frequency or damping ratio are negative, which is unphysical.

2. For an arbitrary model order M, a pole for a specific mode converges on a natural frequency and damping ratio as $M \to \infty$, within a certain tolerance interval.

2.1.2 The polyreference-LSCF modal parameter estimation algorithm

The algorithm implemented in this work is the polyreference least-squares complex-frequency-domain (pLSCF) method, an industry-standard technique for frequency-domain modal analysis. pLSCF is essentially a modal curve-fitting algorithm that builds on the classical least-squares complex-frequency-domain (LSCF) approach. Its key feature is its polyreference formulation, so rather than fitting each input-output transfer function separately, it performs a single, simultaneous fit for every output channel using all available input channels.

Building on the single-reference formulation proposed by Guillaume et al. [31, 32], the original LSCF fits each input-output pair individually, a procedure that becomes computationally taxing when channel counts are high. This inefficiency motivated the development of polyreference implementations. Peeters et al. introduced one such implementation, PolyMAX, in [33–36]. The polyreference LSCF (pLSCF) method offers several advantages: it improves computational efficiency, facilitates the separation of closely spaced modes, and removes the need to perform a Singular Value Decomposition of the modal residues when estimating mode shapes, contributing even more to the computational efficiency. The latter benefit is discussed in detail in Section 2.1.3.

The pLSCF method adopts a right matrix fractional polynomial model for fitting measured FRF data, the following is a brief overview of the algorithms process, a full derivation is available in B:

$$[H(\omega)] = [N(\omega)][D(\omega)]^{-1} \tag{2.6}$$

Such that $H(\omega) \in \mathbb{C}^{N_{outputs} \times N_{inputs}}$ is the FRF matrix, where $D(\omega) \in \mathbb{C}^{N_{inputs} \times N_{inputs}}$, is the denominator matrix polynomial, and $N(\omega) \in \mathbb{C}^{N_{outputs} \times N_{inputs}}$, is the numerator matrix polynomial. The rows corresponding to each output o in the FRF matrix can be represented as such:

$$\langle H_o(\omega) \rangle = \langle N_o(\omega) \rangle \, [D(\omega)]^{-1} \tag{2.7}$$

The row vector numerator polynomial for the o^{th} output, and the denominator matrix polynomial are defined in terms of a polynomial basis function, $\Omega(\omega)$, and their respective polynomial coefficients, β and α as such:

$$\langle N_o(\omega) \rangle = \sum_{r=1}^{p} \Omega_r(\omega) \, \langle \beta_{or}(\omega) \rangle \tag{2.8}$$

$$[D(\omega)] = \sum_{r=1}^{p} \Omega_r(\omega)[\alpha_r] \tag{2.9}$$

With the polynomial basis function $\Omega_r(\omega) = e^{j\omega\Delta tr}$. Although not initially obvious as polynomials with conventional form $p(x) = c_0 + c_1 x + c_2 x^2 + \cdots + c_n x^n$, the basis functions are expressed in the s-domain where $s = e^{j\omega\Delta t}$. The polynomial coefficients, $\alpha_r \in \mathbb{R}^{N_{inputs} \times N_{inputs}}$ and $\beta_{or} \in \mathbb{R}^{1 \times N_{inputs}}$, are assembled into matrix form.

$$\theta = \begin{pmatrix} \beta_0 \\ \beta_1 \\ \beta_2 \\ \dots \\ \beta_{N_o} \\ \alpha \end{pmatrix} \in \mathbb{R}^{(N_{outputs}+N_{inputs})(p+1)\times N_{inputs}} \tag{2.10}$$

The weighted nonlinear least squares error is defined by the difference of the model FRF, $H(\omega_k)$, and the measured FRF, $\hat{H}(\omega_k)$, as

$$\epsilon_o^{NLS}(\theta, \omega_k) = w_o(\omega_k)(H_o(\omega_k) - \hat{H}_o(\omega_k)) \tag{2.11}$$

This error produces the nonlinear least squares cost function.

$$l^{NLS}(\theta) = \sum_{o=1}^{N_{out}} \sum_{k=1}^{N_f} \mathbf{tr}\{(\epsilon_o^{NLS}(\theta, \omega_k))^H \epsilon_o^{NLS}(\theta, \omega_k)\} \tag{2.12}$$

Where, $\bullet^H$ is the Hermitian transpose of a matrix, and $\mathbf{tr}\{\bullet\}$ is its trace. A subsequent linearisation is performed to approximate this least squares problem, through right-multiplying the error term by the denominator polynomial D. The linearisation results in a cost function in terms of the polynomial coefficients:

$$l^{LS}(\theta) = \mathbf{tr}\{\theta^T J^H J \theta\} \tag{2.13}$$

In Equation 2.13, the term J is the Jacobian matrix, containing block elements of the weighted polynomial basis, and the same polynomial basis premultiplied by the measured data. Minimising the cost function gives the normal equation:

$$2Re(J^H J)\theta = 0 \tag{2.14}$$

Polynomial coefficients, α and β are obtained by imposing a constraint on the system equations and solving the resulting linear system $\mathbf{A} \cdot \mathbf{x} = \mathbf{B}$. When fitting theoretical transfer functions to measured data, the mathematical tools themselves rarely present difficulties; indeed, extracting a polynomial from FRF measurements is straightforward using the pLSCF method. The true challenge lies in constructing a model that remains physically meaningful. Ultimately, pLSCF seeks to determine modal parameters via the system poles. One must return to the fundamental definition of a system pole — the frequency at which the response becomes infinite — and thus enforce the condition that the denominator polynomial equals zero. It is quickly apparent that the poles are the roots of the polynomial D. Using the coefficients of the denominator polynomial, the roots can be found using the so-called companion matrix. The companion matrix of a matrix polynomial can be defined as:

$$C = \begin{pmatrix} 0 & I & \dots & 0 & 0 \\ 0 & 0 & I & \dots & 0 \\ \dots & \dots & \dots & \dots & \dots \\ 0 & 0 & \dots & 0 & I \\ -\alpha_0^T & -\alpha_1^T & -\alpha_2^T & \dots - \alpha_{p-2}^T & -\alpha_{p-1}^T \end{pmatrix} \tag{2.15}$$

An interesting property of companion matrices is that its eigenvalues are the roots of its polynomial, as such, the poles and the modal participation factors can be found using an eigendecomposition of the companion matrix.

$$CQ = \Lambda Q \tag{2.16}$$

Where Q is the eigenmatrix, and contains the modal participation factors, and Λ is a diagonal matrix which has the system's discrete time poles on its diagonal entries. A model of polynomial order p will give $N_{inputs} \cdot p$ number of poles and $Q \in \mathbb{C}^{N_{inputs} \cdot p \times N_{inputs} \cdot p}$ participation factors.

2.1.3 Modeshape estimation using LSFD

In practice, modeshapes are estimated using the Least Squares Frequency Domain algorithm, which is a linear regression designed to find the modal residues, a product of the modeshapes and the participation factors, and the upper and lower residuals, values which account for non-ideal vibratory behaviour. The LSFD fits a new model to measured FRF data as shown in Equation B.35.

$$H(s) = \sum_{m=1}^{N_{modes}} \left(\frac{\Psi_m \cdot L_m^T}{s - \lambda_m} + \frac{\Psi_m^* \cdot L_m^H}{s - \lambda_m^*} \right) + \frac{\mathbf{LR}}{s^2} + \mathbf{UR} \tag{2.17}$$

Where Ψ_m is the modeshape vector for the mode m, L_m is the mode's participation factor, $\mathbf{LR}$ and $\mathbf{UR}$ are the upper and lower residuals, and $s = j\omega$. By taking the Modal Residues for the i^{th} mode $[Res_i] = \Psi_m \cdot L_m^T$, they can be found, alongside the Upper and Lower residuals, in one single linear least squares step. With no knowledge of the modal participation factors, the modeshape can only be estimated using a singular value decomposition (SVD) of the residue matrix.

$$[Res_i] = \mathbf{U\Sigma V^T} \tag{2.18}$$

Where $\mathbf{\Sigma}$ is a diagonal matrix containing the singular values of the matrix, $\mathbf{U}$ is a matrix containing left singular vectors, and $\mathbf{V}$ is a matrix containing right singular vectors. This process can yield accurate modeshapes only if the residue matrix is a rank-1, which mat not always be the case. Knowledge of the modal participation factors, which the pLSCF algorithm provides, removes this limitation by allowing a vector to be calculated using the properties of the outer product and rearranging. The use of the outer product provides quicker computation, as matrix-vector multiplication is generally considered quicker than the SVD, this is particularly important for the frequently encountered, high number of output systems.

$$\mathbf{\Psi}_i = \frac{[Res_i]\mathbf{L}_i}{\mathbf{L}_i^H \mathbf{L}_i} \tag{2.19}$$

2.2 Implementing pLSCF

2.2.1 Project environment: software development practices

The overarching aim of the project is the development of Python library for structural dynamics and modal analysis. However, it is impractical for one individual to develop a comprehensive and robust library for such a large scope, and hence the focused aim of implementing and testing one popular algorithm. The library is being developed collaboratively by academics, and students as part of a large coordinated effort. In such large-scale projects, effective management is needed to prevent disruptive mishaps to the codebase and to check that the contributors are using an up-to-date version of the source code. Version control systems, such as Git, are standard practice in software development for collaborative project management. Git enables contributors to work independently on "branches," allowing them to develop and test code without immediately affecting the main codebase, while maintaining a detailed history of changes stored within a "repository ". The Git repository is hosted and managed on GitHub, which is an online platform which simplifies the usage of Git version control, as it adds more intuitive project management functionalities. Furthermore, the project adopts an Agile Project Management paradigm; focusing on iterative improvement, through periodic feedback, leading to a faster, more efficient development process.

2.2.2 Software Testing

A pervasive challenge associated with open-source projects is the lack formal or monetary incentives for the developers to comprehensively test published work, resulting in superficially checked software which allows latent issues and bugs to persist. This deficiency in systemic validation undermines the software's reproducibility and accuracy in research. Addressing these gaps through implementing rigorous software testing is the most effective remedy. Comprising unit tests, which validate the functionality of individual components such as functions or objects in isolation, and integration (or end-to-end) tests, which verify that the individually tested components interact correctly when implemented together. These tests are typically done using a known set of inputs and expected behaviours.

Consider a function, with the sole purpose of obtaining the companion matrix from a known set of polynomial coefficients, α. The code for the function can be seen in Listing 2.1, the unit test to validate this code gave a simple set of polynomial coefficients, e.g. `alpha = [1, 2, 3, 4, 5, 6, 7, 8, 1]`, and asserted that the returned companion matrix was equal to a hand calculated companion matrix.

2.2.3 Optimisation

Python's readable syntax has made it the *lingua franca* of scientific computing, yet this convenient syntax, which abstracts the user from the machine operations, imposes a speed penalty. Recent benchmarks for relevant applications confirm this gap: Diehl et al. report the slowest runtimes when solving the 1-D heat equation in Python compared with nine other languages [38], and Almurayh notes similar results for large-scale matrix multiplication [39]. In modal analysis the data volumes, for instance continuous high sample frequency measurements from aircraft or civil-engineering sensors, magnify these overheads, so scripts written in native Python may finish running long after a structure's service life.

Most practitioners therefore rely on NumPy's N-Dimensional Array `ndarray`. NumPy [37] is an open source library extensively used in scientific computing, due to it's C-coded backend. NumPy's array has become the de facto format for arrays in Python programming, and as such, many libraries relevant in scientific [40] and mathematical computing [41], and data visualisation [42] have relied on NumPy. The development of the open-source modal analysis library relies heavily on NumPy's data containers and functions, as well as SciPy, a python library which integrates signal processing functions and numerical methods, due to their speedy linear algebra operations.

Even with NumPy, cubic-time operations like multiplying two $n \times n$ matrices remain $\mathcal{O}(n^3)$, so a ten-fold increase in dimension inflates work by three orders of magnitude. Efficient modal analysis code thus requires both the right low-level tools, which NumPy and SciPy provide, and careful limitation aware software design.

Applying the discussed principles into the implementation of pLSCF is a straightforward process. Consider the calculation for the 3-Dimensional array (tensor) R, defined for the o^{th} on the 3^{rd} dimension by

$$R_o = Re(X_o^H X_o) \tag{2.20}$$

```python
import numpy as np

def _calc_comp_mat(alpha:np.ndarray):

    n_in = alpha.shape[-1]
    a1   = alpha[:-n_in,:]
    mp = a1.shape[0]


    comp_mat= np.eye(mp-n_in)
    comp_mat = np.vstack((np.zeros((n_in,mp-n_in)),comp_mat))

    return np.hstack((comp_mat,a1)).T

def test_calc_comp_mat():

    n_inputs = 2
    _p = 2
    # literally alpha is 1 to 8, with I(N_inputs) as alpha_p.
    alpha_known = np.array([[1,2],
                            [3,4],
                            [5,6],
                            [7,8],
                            [1,0],
                            [0,1]])
    # Hand calculated
    comp_mat_known = np.array([[0,0,1,0],
                               [0,0,0,1],
                               [1,3,5,7],
                               [2,4,6,8]])

    comp_mat_test = p._calc_comp_mat(alpha_known)

    assert np.allclose(comp_mat_known,comp_mat_test), "Error in
       companion matrix value"
    assert comp_mat_test.shape == (n_inputs*_p,n_inputs*_p), "
       Error in companion matrix shape"
```

Listing 2.1: Python function which constructs and returns the companion matrix from input argument α. This function, and others, depend on NumPy [37], which is discussed further section 2.2.3.

Where X_o is the weighted polynomial basis of the least squares problem for the o^{th} output/DOF. A progression of the implemented calculation is shown in Listing **??**. The first implementation relies on Python's native matrix multiplication operator. The second implementation leverages NumPy's Einstein Summation Convention function, the recommended function for multiplying arrays of dimensions higher than 2. This function however, struggles at higher model orders and higher sensor numbers, this is likely due to the overhead involved with translating the subscript inputs into array dimensions. The final implementation leverages the `dot` function, which is a highly optimised 2D matrix multiplication routine, as it is able to divide the multiplication into different routines which are run simultaneously.

,

```python
import numpy as np


# First implementation
def calc_r(X):
    n_outputs = X.shape[-1]

    for output in range(n_outputs)
        r[:,:,output] = np.real(X[:,:,output].H @ X[:,:,output])

    return r

# Second implementation: improved using numpy functions
def calc_r(X):
    r = np.einsum('ijk,ljk -> ilk',X.conj(),X).real
    return r

# Optimum implementation
def calc_r(X):
    n_out = X.shape[-1]
    r = np.empty((X.shape[1],X.shape[1],X.shape[-1]), dtype=np.
       complex128)
    xh = np.conjugate(X).transpose((1,0,2))
    for output in range(n_out):
        r[:,;,output] =  np.dot(xh[:,:,output],X[:,:,output])

    return np.real(r)
```

Listing 2.2: Various implementations of calculating the tensor R

2.2.4 Code readability and usability

Highly optimised code comes with the inconvenience of being abstract and difficult to visually interpret. Developers must consider the user when writing their code, as it is a widely cited [43–47] estimate in software engineering that developers will spend a 10-to-1 ratio of their time reading code as compared to writing it. The most common practices for making code readable are inline comments, and documentation. Inline comments are simple text statements describing a line of code which may be unintuitive, while documentation refers to a block of text that describes a function, class, or module in depth. An example usage of both documentation and comments is in Listing 2.3.

There are additional code writing guidelines offered by the Python Enhancement Proposal (PEP) [48]. PEP is concerned with standardising the naming conventions of variables, functions, classes, and files, as well as the formatting of written code. For example, PEP recommends that lowercase with underscores be used for variables, functions, and filenames, e.g. `dynamics.py`, `y=x+10`, while uppercase is recommended for constants, or variables that are meant to be unchanged. e.g. `PLANCK = 1.616255e-35`. These guidelines add a layer of simplicity which helps the user better understand how the code works.

```python
def random_function(input1, input2):
    """
    random_function performs... {insert summary}
    This block of text enclosed by 3 quotation marks is referred
       to as docstrings in Python programming

    Args:
        input1 (expected data type): description
        input2 (expected data type): description

    Returns:
        output (data type): description

    Reference:
        [1] Book title, article etc..

    """


    # this is an inline comment
    # It can provide quick explanations to unintelligible lines
       of code.
    # np.dot calculates the matrix multiplication of 2
       dimensional arrays, assuming their shapes are compatible.
    output = np.dot(input1, input2)
    return output
```

Listing 2.3: Example documentation and inline comment usage in Python code.

Chapter 3

Results

3.1 Benchmarking results

3.1.1 Simulated systems: 10 degree-of-freedom oscillator

To benchmark pLSCF, a number of lumped mass systems, as well as experimental data was used. Consider a 10 degree-of-freedom oscillator, with physical parameters as seen in Table 3.1.

Table 3.1: Physical Parameters of lumped mass-spring 10-DOF system.

i	m_i [kg]	k_i [$10^4\,\mathrm{N\,m^{-1}}$]	c_i [$10^{-1}\,\mathrm{N\,s\,m^{-1}}$]
1	1	2	2
2	2	3	3
3	3	4	4
4	4	1	1
5	5	6	6
6	1	2	2
7	2	3	3
8	3	4	4
9	4	5	5
10	5	3	3
11	–	4	4

Following the calculation of the modal properties using the method highlighted in Section 1.1.1, the analytical FRF is found using equation 2.3. This synthetic data is passed through the implemented pLSCF algorithm and a stabilisation diagram is constructed for a maximum model order of 30, Figure 3.1 showcases the diagram.

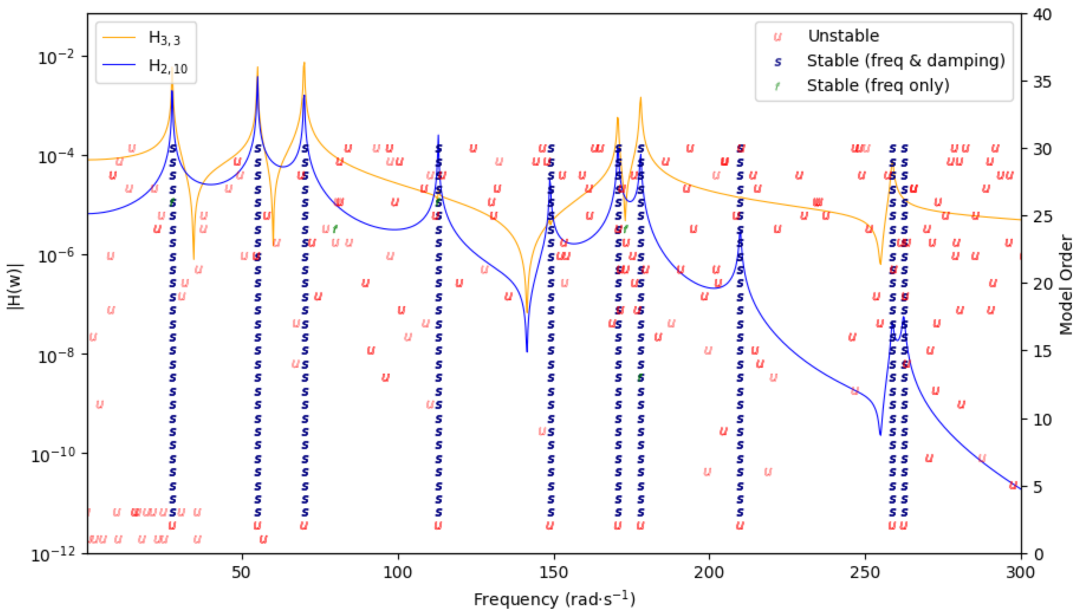


Figure 3.1: Stabilisation for the system with the physical parameters shown for model orders of up to 30, stability criteria used (1) tolerance of 0.2rad/s in natural frequency, and 0.02 in damping ratio for increasing model order. (2) $Re(\lambda) < 0$

Interpreting the stabilisation diagram, and selecting the most stable and suitable poles, the LSFD algorithm is used to estimate the modeshapes and reconstruct the FRF. The modeshapes for the 6^{th} mode are showcased in Figure 3.2, comparing the true modeshapes and the LSFD estimated ones.

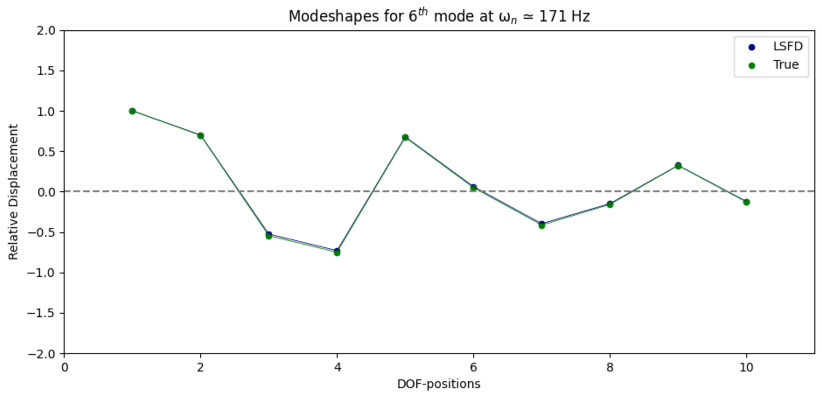
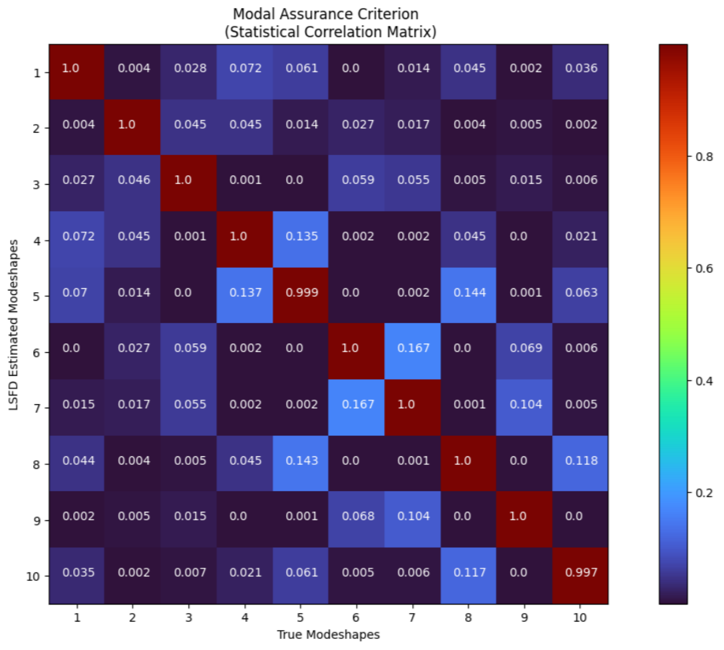


Figure 3.2: Modeshape plot of 2 Mass-normalised modeshapes, true value and LSFD Estimate. Both vectors are normalised to have a maximum value of 1. plot generated using matplotlib [42].

This plot gives a more qualitative impression for how closely the LSFD algorithm solution tracks the theoretical value. A more quantitative approach is using the *Modal Assurance Criterion (MAC)* matrix, a statistical measure of correlation between 2 sets of modeshape vectors on the same space. The (i, j) element of the MAC matrix, which quantifies how strongly the i^{th} mode shape of one system aligns with the j^{th} mode shape of another, is defined by:

$$\mathbf{MAC}(\{\Psi_i\}, \{\Psi_j\}) = \frac{|\{\Psi_i^H\}\{\Psi_j\}|^2}{(\{\Psi_i^H\}\{\Psi_i\})(\{\Psi_j^H\}\{\Psi_j\})} \tag{3.1}$$

One can simply interpret this as the squared cosine angle between both vectors, being 1 if they are collinear, or 0 if perpendicular. This might also be interpreted as matrix of pairwise squared correlations, R^2, between the 2 sets of vectors. The MAC matrix correlating the LSFD solution to the modeshapes and the true value is seen in Figure 3.3.



	1	2	3	4	5	6	7	8	9	10
1	1.0	0.004	0.028	0.072	0.061	0.0	0.014	0.045	0.002	0.036
2	0.004	1.0	0.045	0.045	0.014	0.027	0.017	0.004	0.005	0.002
3	0.027	0.046	1.0	0.001	0.0	0.059	0.055	0.005	0.015	0.006
4	0.072	0.045	0.001	1.0	0.135	0.002	0.002	0.045	0.0	0.021
5	0.07	0.014	0.0	0.137	0.999	0.0	0.002	0.144	0.001	0.063
6	0.0	0.027	0.059	0.002	0.0	1.0	0.167	0.0	0.069	0.006
7	0.015	0.017	0.055	0.002	0.002	0.167	1.0	0.001	0.104	0.005
8	0.044	0.004	0.005	0.045	0.143	0.0	0.001	1.0	0.0	0.118
9	0.002	0.005	0.015	0.0	0.001	0.068	0.104	0.0	1.0	0.0
10	0.035	0.002	0.007	0.021	0.061	0.005	0.006	0.117	0.0	0.997

Figure 3.3: Modal Assurance Criterion matrix (rounded to 3 decimal places), plot generated using matplotlib [42].

Additionally, to gauge the inherent similarity among the true mode shapes themselves, the autocorrelation MAC matrix is computed for the true modes (i.e. the MAC of the true shapes against themselves). This serves as a baseline when comparing the LSFD estimated vs. true using a MAC matrix. The autocorrelation MAC is shown in Figure 3.4.

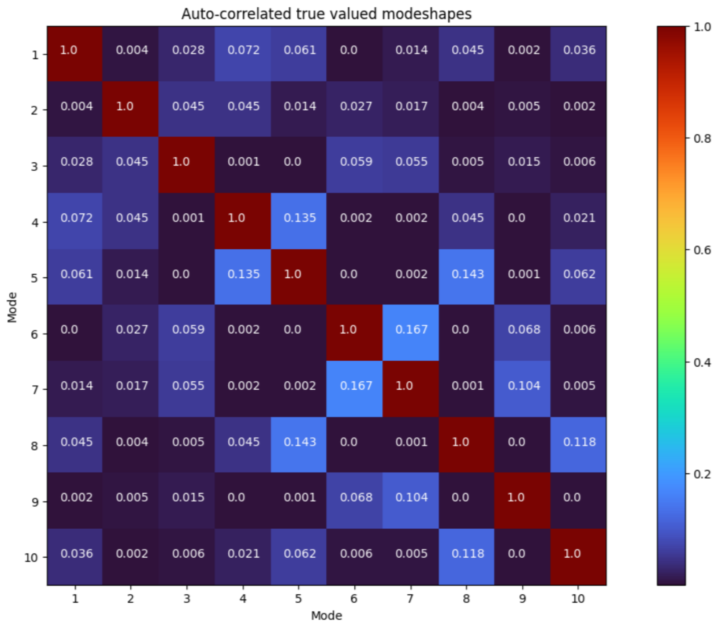


Figure 3.4: Autocorrelation MAC matrix for the analytically calculated modeshapes, plot generated using matplotlib [42].

3.1.2 Introducing noise into the measurements

In Modal Testing practice, it is common, sometimes standard, to use a white noise signal for the forced excitation, this subjects the tested structure to a broad range of frequencies. If the tested structure is linear and other test parameters aren't obstructive, one can consider the structure as a frequency shaping filter on the white noise input, emphasising the magnitude at the natural frequencies and appearing random or noisy at others. The existence of noise in the measurement chain stresses the need for algorithms performant with noisy data sets. To benchmark the robustness of the implemented pLSCF algorithm, a Gaussian noise spectrum is generated using pre-existing functions, Algorithm 3.1 showcases the process, the accelerance FRF is selected to resemble real measured data; typical dynamic data is measured by accelerometers.

Algorithm 3.1: Pseudocode description of generating noisy measurements, by adding white noise described by the Signal-to-Noise ratio and a uniformly sampled phase

```
$A \leftarrow$ calculated accelerance (complex FRF)
$\mathrm{SNR}_{\mathrm{dB}} \leftarrow 25$ (*desired SNR in decibels*)

$r(\omega) \leftarrow \dfrac{|A(\omega)|}{10^{\mathrm{SNR}_{\mathrm{dB}}/20}}$        ▷ radius for each frequency bin

$\phi(\omega) \overset{\text{i.i.d.}}{\sim} \mathcal{U}(0, 2\pi)$        ▷ independent random phase
$W(\omega) \leftarrow r(\omega)\, e^{j\phi(\omega)}$        ▷ complex noise on a circle of radius $r$

$A_{\mathrm{noisy}} \leftarrow A + W$        ▷ noise is added linearly
```

A stabilisation diagram, Figure 3.5, is constructed for 70 model orders. It is quickly apparent that the noise biases the algorithm towards non-dynamic modes, even with a liberal tolerance, leading to non-converging and unstable poles. A proposed solution to mitigate this unwanted deviation is segmenting the response function about each peak, and pass through the individual segments into the algorithm. This solution can be computationally quicker in some cases as the required model orders for each segment are typically lower. The stabilisation diagram from implementing the proposed solution is showcased in Figure 3.6, and due to the large number of poles this solution produces, Figure 3.7 provides visual clarity by omitting the unstable poles from the diagram.

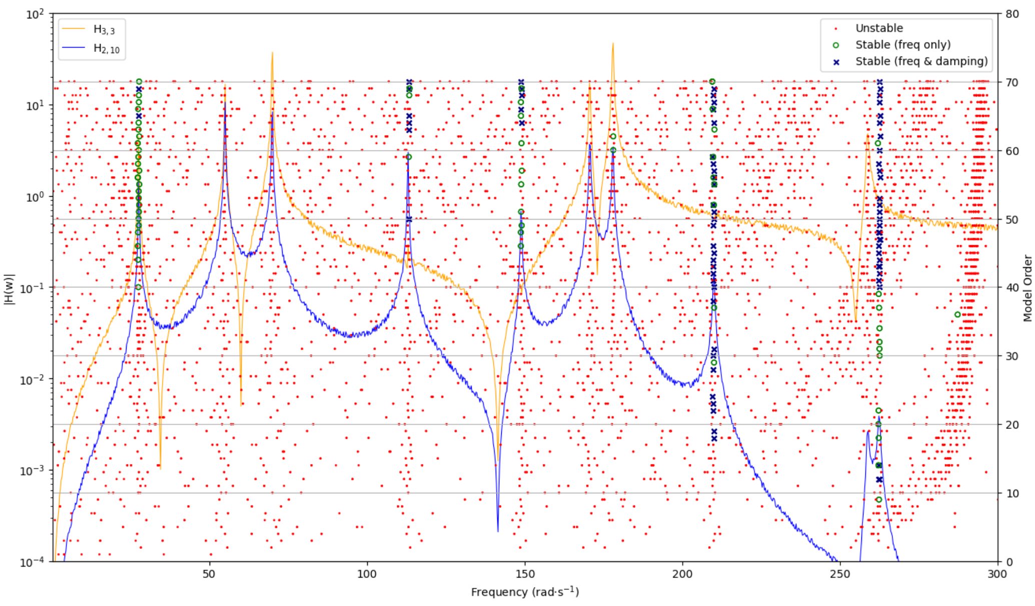


Figure 3.5: Accelerance FRF plot, and stabilisation plot for a maximum model order of 70, with frequency tolerance of 0.4, and damping tolerance of 0.05

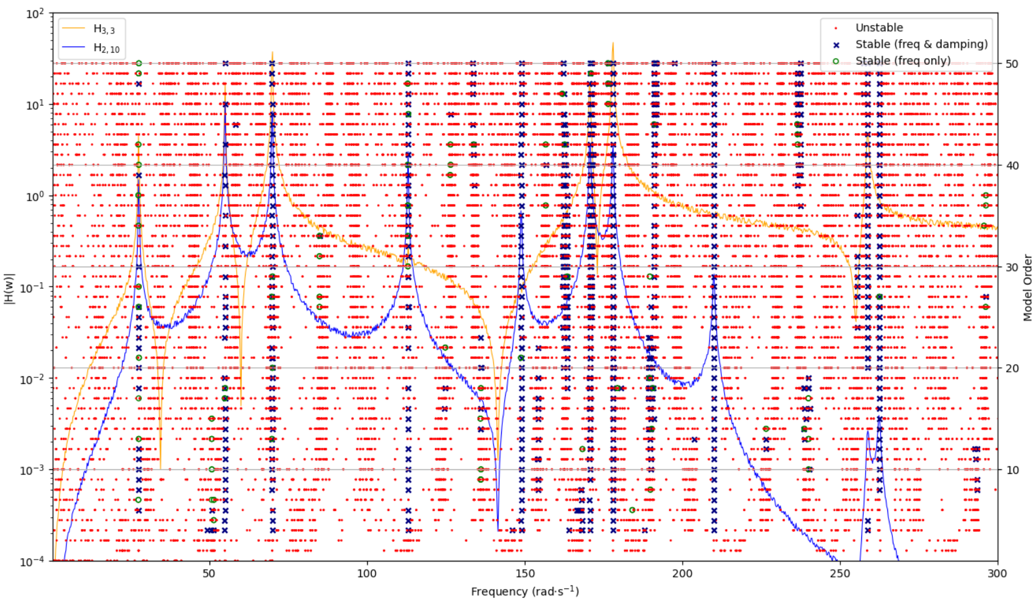


Figure 3.6: Accelerance FRF plot, and stabilisation plot for a maximum model order of 50, using the proposed segmentation method, with frequency tolerance of 0.05 and damping tolerance of 0.01

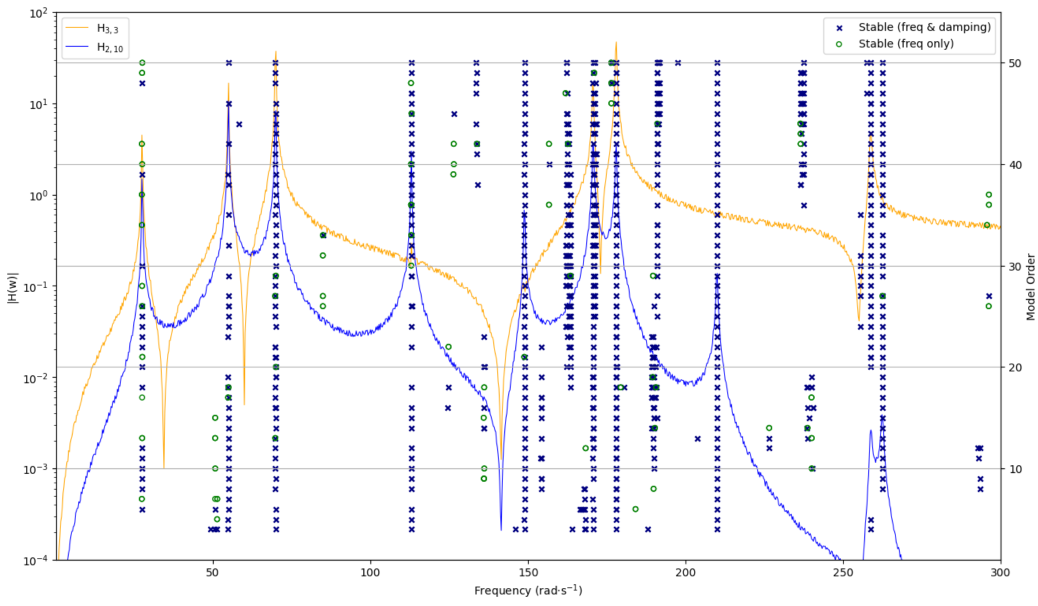


Figure 3.7: Accelerance FRF plot, and stabilisation plot for a maximum model order of 50, using the proposed segmentation method, with unstable poles omitted

3.2 Model Order Reduction: Technique Comparison

In practice, a practitioner would rarely re-run the identification algorithm for every candidate model order; Instead, they first obtain a higher model order, and reduce it using various methods to the desired orders. This section showcases benchmarks for two separate model reduction schemes, SVD based reduction, and direct matrix truncation, against the baseline ”brute-force” loop that recomputes each order from scratch.

For a square matrix $\mathbf{A} = \mathbf{U\Sigma V^T} \in \mathbb{R}^{n\times n}$ with rank$-n$, SVD based reduction aims to find the rank$-k$ realization of A through eliminating the $n-k$ lowest valued singular values, and their corresponding singular vectors. Hence a rank$-k$ reduced matrix $A_k = \mathbf{U_k \Sigma_k V_k^T}$, where $\mathbf{U_k} \in \mathbb{R}^{n\times k}$, $\mathbf{V_k} \in \mathbb{R}^{n\times k}$, and $\mathbf{\Sigma_k} \in \mathbb{R}^{k}$. For the same square matrix $\mathbf{A}$, direct truncation takes the $\mathbb{R}^{k\times k}$ square and leaves out the rest. Applying these schemes to pLSCF, the appropriate matrix to reduce is the design matrix of the least-squares problem $\mathbf{M}$ (See Equation B.28). To benchmark the speed of each method, three maximum model orders are chosen, 10, 20, and 100, each scheme would calculate all the model orders between 1 and the maximum one. Table 3.2 showcases the different runtimes. While Tables 3.3 and 3.4 showcase the accuracy for a simulated 2-DOF oscillator using different model order reduction/realisation methods. All these methods are available to the user for a preference in their analyses.

Table 3.2: Speed test of MOR techniques at various reduced model orders.

Method	Order 10 (s)	Order 20 (s)	Order 100 (s)
Direct truncation	0.008327	0.009515	10.648412
SVD-based rank reduction	0.006160	0.017249	36.642554
Brute force	0.052435	0.101725	49.564672

Table 3.3: Natural Frequency accuracy components for each MOR technique.

Method	ω_1	ω_2
Truncation	0.87172796	0.50329227
SVD-based rank reduction	0.87172831	0.50329231
Brute force	0.87173224	0.50329240
Actual	0.87172752	0.50329212

Table 3.4: Damping ratio accuracy components for each MOR technique.

Method	ζ_1	ζ_2
Truncation	0.05476992	0.03162261
SVD-based rank reduction	0.05477160	0.03162370
Brute force	0.05477117	0.03162255
Actual	0.05477226	0.03162278

Chapter 4

Discussion, and Further Work

This project set out to (a) gain an implementable understanding of the pLSCF modal parameter estimation method (b) integrate the algorithm, together with a supplementary LSFD routine, into an open-source Python library, (c) verify their numerical behaviour and performance and (d) improve the overall usability of the software.

4.1 Accuracy of pLSCF and LSFD

The stability diagram for the *clean* FRF data showed quick convergence of the poles on the true natural frequencies. The pLSCF algorithm employs a weighted least-squares regression that emphasizes where the variance is lowest for each output i.e. the FRF troughs, this ”down-weighing” of the spectral lines where the peaks lie creates a large error when a pole is misplaced, causing the LS solution converge remarkably quickly. In fact, for the simulated 10-DOF system, it reached its first stable solution at a model order of 3, despite the actual system containing 10 modes. Stability was assessed using predefined tolerances on the maximum deviation for both natural frequencies and damping ratios, and the model of order 3 not only fulfilled these criteria but did so with a maximum deviation from the theoretical values of just $8.8 * 10^{-5}\%$. Table 4.1 shows a full comparison of the true poles and pLSCF estimates.

Table 4.1: Comparison of pLSCF and true poles (imaginary parts < 0 shown). Percentage difference between the poles is highlighted with a 10^5 scaling factor (So first percentage difference is 0.00005504%)

No.	pLSCF pole	True pole	$\Delta\%(\times 10^{-5})$
1	$-0.003818 - 27.583987\,i$	$-0.003804 - 27.583994\,i$	5.504
2	$-0.015049 - 54.925932\,i$	$-0.015084 - 54.925965\,i$	8.831
3	$-0.024417 - 69.887393\,i$	$-0.024421 - 69.887393\,i$	0.561
4	$-0.063769 - 112.954553\,i$	$-0.063794 - 112.954549\,i$	2.250
5	$-0.110767 - 148.843795\,i$	$-0.110772 - 148.843808\,i$	0.937
6	$-0.145576 - 170.624492\,i$	$-0.145564 - 170.624491\,i$	0.742
7	$-0.158286 - 177.918991\,i$	$-0.158276 - 177.918990\,i$	0.589
8	$-0.220284 - 209.891063\,i$	$-0.220272 - 209.891056\,i$	0.665
9	$-0.334599 - 258.689376\,i$	$-0.334602 - 258.689377\,i$	0.114
10	$-0.344212 - 262.390535\,i$	$-0.344245 - 262.390541\,i$	1.251

Having established a stable and accurate set of poles using pLSCF, the Least Squares Frequency Domain (LSFD) algorithm was utilised to recover the modeshapes. Using the poles and their corresponding participation factors (the eigenvectors of the companion matrix, 2.16) as inputs to the LSFD algorithm, the full modeshape matrix was recovered in one least-squares step. The quality of the estimated modeshapes was examined using the Modal Assurance Criterion (MAC). The diagonal of the cross-correlation MAC matrix between the LSFD estimated modeshapes and the true modeshapes has a minimum value of 0.997 as seen in Figure 3.3. Such values are typically interpreted as ”identical modeshape” in Modal Analysis. To ensure that the high MAC values reflected genuine physical correspondence rather than a coincidental numerical match, an additional check was performed: the previous cross-correlation MAC was compared against the autocorrelation MAC, Figure 3.4, of the true modeshapes themselves. The two matrices are indistinguishable to two significant figures, this implies that the LSFD estimated modeshapes retain the inherent dynamic content as well as the orthogonality properties of the system. If one were to substitute the analytically calculated modeshapes with the estimated ones in other tasks, it would cause trivial, if any, modelling error. Overall, the pLSCF-LSFD process, produces highly accurate poles and mode shapes, for an *ideal* data set.

When uniformly sampled white noise is added to the response, pLSCF struggles to converge on the system modes, even when generous tolerances for natural frequency and damping are applied, and the model order is deliberately over-fitted (see Figure 3.5). The difficulty arises because noise creates spurious, peak-like features in the FRF that bias the least-squares fit.

To mitigate this, the FRF is first segmented around each true resonance. Peak locations are detected from the summation function (the sum of all FRFs) whose genuine modes are prominent. Modal identification is then carried out on each segment separately, restoring the expected convergence even with tight tolerances and lower model orders, as illustrated in Figures 3.6 and 3.7.

A trade-off is that different modes may now stabilise at different model orders; for example, in Figure 3.7 the third mode converges at a lower order than the ninth. Ideally, each mode would be assigned the model order at which it first stabilises, but the current software does not automate this choice. The user must therefore inspect the stability diagram, select the appropriate poles manually, and extract them from their respective data containers. This may seem trivial or a minor inconvenience, but choosing the wrong order for even one mode can propagate significant errors into the modal parameters and distort any subsequent analyses.

4.2 Future Implementations

Future work will first focus on incorporating modeshape convergence into the stability assessment. Coupling the existing pLSCF pole estimator with the LSFD modeshape estimator will enable a MAC-based stability criterion, allowing convergence to be judged not only by natural frequencies and damping ratios, but also by modal orthogonality. Once this methodological link is in place, development will turn to usability: it has been highlighted that the pole selection procedure performed by the user could benefit from a dedicated graphical interface for interactivity. With these foundations established, the backend engine will be broadened to include PolyMAX Plus, a version of pLSCF which is better suited for handling noisy responses. The software will then be released under an open-source licence to encourage external validation and community-driven improvement. Finally, a wider contributor base will facilitate the integration of additional, application-specific identification and modal analysis algorithms, extending the library’s applicability to increasingly challenging scenarios.

4.3 Concluding Remarks

This project delivers a fully tested Python implementation of the polyreference LSCF algorithm and its LSFD mode shape estimator. On a ten-DOF benchmark it reached stable poles at model order 3 with errors below $9 * 10^{-5}\%$ while the ensuing mode shapes matched the analytical set with $MAC \geq 0.997$. A simple peak-segmentation step restored these accuracies under strong white-noise contamination, and speed trials showed that SVD-based or truncation-based model-order reduction is up to 100 $\times$ faster than brute-force recalculation without loss of fidelity.

Future development will focus on integrating modeshape information into the stability test so that MAC convergence is assessed alongside frequency and damping, while a graphical, click-to-select interface will make pole selection intuitive and robust. The backend will be extended with PolyMAX Plus to handle noisier data sets, and once these foundations are in place the entire codebase will be released under a permissive open-source licence, inviting peer review and accelerating community-driven enhancements.

Appendix A

Multi-degree-of-freedom systems

A.1 Forced Vibration Solution

The following is the author's interpretation and summary of the work by Fu in [5], Ewins in [4], and Rogers in [29] and [30] As previously highlighted in Section 1, the solution to a dynamic system can be represented by an eigenvalue problem:

$$(\mathbf{K} - \omega^2\mathbf{M})\{\varphi\} = 0 \tag{A.1}$$

Considering 2 modes, i and j, equation A.1 becomes:

$$(\mathbf{K} - \omega_i^2\mathbf{M})\{\varphi\}_i = 0 \tag{A.2}$$

and

$$(\mathbf{K} - \omega_j^2\mathbf{M})\{\varphi\}_j = 0 \tag{A.3}$$

Pre-multiplying A.2 by $\{\varphi\}_j^T$, and Transposing A.3 and post-multiplying by $\{\varphi\}_i$ results in:

$$\{\varphi\}_j^T(\mathbf{K} - \omega_i^2\mathbf{M})\{\varphi\}_i = 0 \tag{A.4}$$

and

$$\{\varphi\}_j^T(\mathbf{K} - \omega_j^2\mathbf{M})\{\varphi\}_i = 0 \tag{A.5}$$

subtracting A.4 and A.5:

$$(\omega_i^2 - \omega_j^2)\{\varphi\}_j^T\mathbf{M}\{\varphi\}_i = 0 \tag{A.6}$$

suggesting that for $i \neq j$, $\{\varphi\}_j^T\mathbf{M}\{\varphi\}_i = 0$, substituting this property in equation A.5, $\{\varphi\}_j^T\mathbf{K}\{\varphi\}_i = 0$ holds true. Considering the case where $i = j$, let

$$\{\varphi\}_i^T\mathbf{M}\{\varphi\}_i = m_i \tag{A.7}$$

and

$$\{\varphi\}_i^T\mathbf{K}\{\varphi\}_i = k_i \tag{A.8}$$

where m_i and k_i are the modal mass and stiffness respectively, the eigenvalue for each mode λ_i can be represented as:

$$\lambda_i = \frac{k_i}{m_i}$$

. The properties highlighted by equations A.4, A.5, A.6, A.7 and A.8 are expressed in matrix form more concisely, where:

$$[\Phi]^T\mathbf{M}[\Phi] = \text{diag}(m_i) \tag{A.9}$$

and

$$[\Phi]^T\mathbf{K}[\Phi] = \text{diag}(k_i) \tag{A.10}$$

Referring to the mathematical representation of the frequency response function, from equation **??**, where $H(\omega) = (\mathbf{K} - \omega^2\mathbf{M})^{-1}$, one can express the FRF of a system in terms of its modal parameters as follows,

$$[\Phi]^T(\mathbf{K} - \omega^2\mathbf{M})[\Phi] = [\Phi]^T[H(\omega)]^{-1}[\Phi] \tag{A.11}$$

Substituting from A.9 and A.10, the FRF matrix is expressed as:

$$[H(\omega)] = [\Phi][\omega_i^2 - \omega^2][\Phi]^T \tag{A.12}$$

For a single element in a receptance FRF matrix $H_{oj}(\omega)$, the equation representing it is:

$$H_{jk}(\omega) = \frac{\varphi_{j1}\varphi_{k1}}{\omega_1^2 - \omega^2} + \frac{\varphi_{j2}\varphi_{k2}}{\omega_2^2 - \omega^2} + \cdots + \frac{\varphi_{jn}\varphi_{kn}}{\omega_n^2 - \omega^2} \tag{A.13}$$

where n is the number of modes. Equation A.13 indicates that the value of the FRF at a single frequency line can be interpreted as the contribution of all individual modes to the specific vibration pattern associated with the frequency line. Essentially the FRF is represented as a sum of partial fractions. This representation of the FRF is the basis of the validity of the pLSCF method, as it similarly assumes a rational fractional representation of the FRF.

Appendix B

Derivation of p-LSCF modal parameter estimation method

B.1 The right-matrix rational fractional model

The polyreference least-squares complex frequency domain method employs a right matrix fractional model to fit MIMO Frequency Response Function measurements into a set of rational polynomial transfer functions:

$$[H(\omega)] = [N(\omega)][D(\omega)]^{-1} \tag{B.1}$$

Such that $H(\omega) \in \mathbb{C}^{N_{outputs} \times N_{inputs}}$ is the FRF matrix, where $D(\omega) \in \mathbb{C}^{N_{inputs} \times N_{inputs}}$, is the denominator matrix polynomial, and $N(\omega) \in \mathbb{C}^{N_{outputs} \times N_{inputs}}$, is the numerator matrix polynomial. The rows corresponding to each output o in the FRF matrix can be represented as such:

$$\langle H_o(\omega) \rangle = \langle N_o(\omega) \rangle \, [D(\omega)]^{-1} \tag{B.2}$$

The row vector numerator polynomial for the o^{th} output, and the denominator matrix polynomial are defined in terms of a polynomial basis function, $\Omega(\omega)$, and their respective polynomial coefficients, β and α as such:

$$\langle N_o(\omega) \rangle = \sum_{r=1}^{p} \Omega_r(\omega) \, \langle \beta_{or}(\omega) \rangle \tag{B.3}$$

$$[D(\omega)] = \sum_{r=1}^{p} \Omega_r(\omega) [\alpha_r] \tag{B.4}$$

With the polynomial basis function $\Omega_r(\omega) = e^{j\omega\Delta tr}$. Although not initially obvious as polynomials with conventional form $p(x) = c_0 + c_1 x + c_2 x^2 + \cdots + c_n x^n$, the basis functions are expressed in the s-domain where $s = e^{j\omega\Delta t}$. The polynomial coefficients, $\alpha_r \in \mathbb{R}^{N_{inputs} \times N_{inputs}}$ and $\beta_{or} \in \mathbb{R}^{1 \times N_{inputs}}$, are assembled into matrix form:

$$\beta_o = \begin{pmatrix} \beta_{o0} \\ \beta_{o1} \\ \beta_{o2} \\ \cdots \\ \beta_{op} \end{pmatrix} \in \mathbb{R}^{(p+1) \times N_{inputs}} \tag{B.5}$$

$$\alpha = \begin{pmatrix} \alpha_0 \\ \alpha_1 \\ \alpha_2 \\ \dots \\ \alpha_p \end{pmatrix} \in \mathbb{R}^{N_{inputs}*(p+1)\times N_{inputs}} \tag{B.6}$$

$$\theta = \begin{pmatrix} \beta_0 \\ \beta_1 \\ \beta_2 \\ \dots \\ \beta_{N_o} \\ \alpha \end{pmatrix} \in \mathbb{R}^{(N_{outputs}+N_{inputs})(p+1)\times N_{inputs}} \tag{B.7}$$

B.2 Minimizing the sum of the squared residuals

The collection of both sets of coefficients into one variable θ, makes performing the least squares problem simpler, in a sense, as it becomes the one unknown in this least squares model. As typical in any fitting method, one must minimize the error between the model and the real or measured value. The nonlinear least-squares error for:

Measured FRF:	$\hat{H}_o(\omega_k)$
Model FRF:	$H_o(\omega_k)$

is weighted such that:

$$\epsilon_o^{NLS}(\theta, \omega_k) = w_o(\omega_k)(H_o(\omega_k) - \hat{H}_o(\omega_k)) \tag{B.8}$$

Where $\epsilon_o^{NLS} \in \mathbb{C}^{1\times N_{inputs}}$, $w_o(\omega_k)$ is a scalar weighing function which captures the variation and deviation between multiple inputs on the same measurement point, and $\forall k = 0, 1, 2, \dots, N_{frequency}$. Said weighing function is typically denoted by

$$w_o(\omega_k) = \frac{1}{\sqrt{\mathbf{var}[H_o(\omega_k)]}} \tag{B.9}$$

(See [reference for weighted linear regressions] for more information on weighted least squares.) One can then define the nonlinear cost function as the sum of the error "squared", (hermitian inner product), over the data points, in this case, spectral lines and outputs;

$$l^{NLS}(\theta) = \sum_{o=1}^{N_{out}} \sum_{k=1}^{N_f} \mathbf{tr}\{(\epsilon_o^{NLS}(\theta, \omega_k))^H \epsilon_o^{NLS}(\theta, \omega_k)\} \tag{B.10}$$

In this equation, $\mathbf{tr}\{\bullet\}$ denotes the trace of a matrix, also known as the sum of diagonal elements, and $\bullet^H$ denotes the Hermitian (conjugate) transpose. The trace operator is used as the trace of a product of 2 matrices $\mathbf{A} \in \mathbb{C}^{m\times n}$ and $\mathbf{B} \in \mathbb{C}^{m\times n}$ will equal the sum of each individual element in $\mathbf{A}$ with the individual elements of $\mathbf{B}$. This provides a sum of all square residuals/errors in the cost function.

$$\mathbf{tr}\{\mathbf{A}^H\mathbf{B}\} = \mathbf{tr}\{\mathbf{A}\mathbf{B}^H\} = \mathbf{tr}\{\mathbf{B}^H\mathbf{A}\} = \mathbf{tr}\{\mathbf{B}\mathbf{A}^H\} = \sum_{i=1}^{m}\sum_{j=1}^{n} a_{ij}b_{ij} \tag{B.11}$$

B.3 Linearizing the error

One can then obtain the polynomial coefficients through minimizing the cost function in B.10, by setting the derivative $\frac{\partial l^{NLS}}{\partial \theta}$ equal to zero, however a nonlinear cost function will yield nonlinear derivative equations, (typically called normal equations in linear regression). A subsequent linearization of the cost function can approximate (suboptimally) the least squares problem, this is achieved through right multiplying the cost function with denominator polynomial $\mathbf{D}$. This gives a linear error:

$$\begin{aligned} \epsilon_o^{LS}(\omega_k, \theta) &= w_o(\omega_k)(N_o(\omega_k, \beta_o) - \hat{H}_o(\omega_k) D(\omega_k, \alpha)) \\ &= w_o(\omega_k) \sum_{r=0}^{p} (\Omega_r(\omega_k)\beta_{or} - \Omega_r(\omega_k)\hat{H}_o(\omega_k)\alpha_r) \end{aligned} \tag{B.12}$$

Stacking the error in terms for all spectral lines in one matrix $E_o^{LS}(\theta) \in \mathbb{C}^{N_f \times N_{in}}$:

$$E_o^{LS}(\theta) = \begin{pmatrix} \epsilon_o^{LS}(\omega_1, \theta) \\ \epsilon_o^{LS}(\omega_2, \theta) \\ \epsilon_o^{LS}(\omega_3, \theta) \\ \dots \\ \epsilon_o^{LS}(\omega_{N_f}, \theta) \end{pmatrix} = \begin{pmatrix} X_o & Y_o \end{pmatrix} \begin{pmatrix} \beta_o \\ \alpha \end{pmatrix} \tag{B.13}$$

Here, new variables $\mathbf{X}$ and $\mathbf{Y}$ are introduced:

$$X_o = \begin{pmatrix} w_o(\omega_1)\Big(\Omega_0(\omega_1) + \Omega_1(\omega_1) \dots \Omega_p(\omega_1)\Big) \\ w_o(\omega_2)\Big(\Omega_0(\omega_2) + \Omega_1(\omega_2) \dots \Omega_p(\omega_2)\Big) \\ \vdots \\ w_o(\omega_{N_f})\Big(\Omega_0(\omega_{N_f}) + \Omega_1(\omega_{N_f}) \dots \Omega_p(\omega_{N_f})\Big) \end{pmatrix} \in \mathbb{C}^{N_f \times (p+1)} \tag{B.14}$$

$$Y_o = \begin{pmatrix} -w_o(\omega_1)\Big(\Omega_0(\omega_1) + \Omega_1(\omega_1) \dots \Omega_p(\omega_1)\Big) \otimes \hat{H}_o(\omega_1) \\ -w_o(\omega_2)\Big(\Omega_0(\omega_2) + \Omega_1(\omega_2) \dots \Omega_p(\omega_2)\Big) \otimes \hat{H}_o(\omega_2) \\ \vdots \\ -w_o(\omega_{N_f})\Big(\Omega_0(\omega_{N_f}) + \Omega_1(\omega_{N_f}) \dots \Omega_p(\omega_{N_f})\Big) \otimes \hat{H}_o(\omega_{N_f}) \end{pmatrix} \in \mathbb{C}^{N_f \times N_{in}(p+1)} \tag{B.15}$$

Where $\otimes$ is the Kronecker product. In these equations, $\mathbf{X}$ is used to capture the frequency content of the least squares problem, and $\mathbf{Y}$ is used to capture both the frequency content and the measured response data. Using these matrices, one can reconstruct the nonlinear cost function into one that is linear:

$$\begin{aligned} l^{LS}(\theta) &= \sum_{o=1}^{N_{out}} \sum_{k=1}^{N_f} \mathbf{tr}\{(\epsilon_o^{LS}(\omega_k, \theta))^H \epsilon_o^{LS}(\omega_k, \theta)\} \\ &= \sum_{o=1}^{N_{out}} \mathbf{tr}\Big\{(E_o^{LS}(\theta))^H E_o^{LS}(\theta)\Big\} \\ &= \sum_{o=1}^{N_{out}} \mathbf{tr}\Big\{\begin{pmatrix} \beta_o^T & \alpha^T \end{pmatrix} \begin{pmatrix} X_o^H \\ Y_o^H \end{pmatrix} \begin{pmatrix} X_o & Y_o \end{pmatrix} \begin{pmatrix} \beta_o \\ \alpha \end{pmatrix}\Big\} \end{aligned} \tag{B.16}$$

If one defines a *Jacobian* matrix $\mathbf{J} \in \mathbb{C}^{N_f N_{out} \times (N_{in}+N_out)(p+1)}$ for the problem as such:

$$\mathbf{J} = \begin{pmatrix} X_1 & 0 & \dots & 0 & Y_1 \\ 0 & X_2 & \dots & 0 & Y_2 \\ \dots & \dots & \dots & \dots & \dots \\ 0 & 0 & 0 & X_{N_{out}} & Y_{N_{out}} \end{pmatrix} \tag{B.17}$$

The cost function can be represented as:

$$l^{LS}(\theta) = \mathbf{tr}\{\theta^T \mathbf{J}^H \mathbf{J} \theta\} \tag{B.18}$$

To obtain real values of θ, one must place a constraint on the cost function such that:

$$l^{LS}(\theta) = \mathbf{tr}\{\theta^T Re(\mathbf{J}^H \mathbf{J})\theta\} \tag{B.19}$$

Where the Gramian matrix of $\mathbf{J}$ can be represented in terms a set of variables, $\mathbf{R}$,$\mathbf{S}$ and $\mathbf{T}$:

$$Re(\mathbf{J}^H \mathbf{J}) = \begin{pmatrix} R_1 & 0 & \dots & 0 & S_1 \\ 0 & 0 & \dots & 0 & S_2 \\ \dots & \dots & \dots & \dots & \dots \\ 0 & 0 & \dots & R_{N_{out}} & S_{N_{out}} \\ S_1^T & S_2^T & \dots & S_{N_{out}}^T & \sum_{o=1}^{N_{out}} T_o \end{pmatrix} \tag{B.20}$$

in which:

$$R_o = Re(X_o^H X_o) \tag{B.21}$$

$$S_o = Re(X_o^H Y_o) \tag{B.22}$$

$$T_o = Re(Y_o^H Y_o) \tag{B.23}$$

B.4 The *Normal Equations*, and extracting the modal parameters

The cost function can then be minimized in terms of α and β to find the best least-squares fit:

$$\frac{\partial l^{LS}(\theta)}{\partial \beta_o} = 2(R_o \beta_o + S_o \alpha) = 0 \qquad \forall O = 1, 2, \dots, N_{out} \tag{B.24}$$

$$\frac{\partial l^{LS}(\theta)}{\partial \alpha} = 2 \sum_{o=1}^{N_{out}} (S_o^T \beta_o + T_o \alpha) \tag{B.25}$$

Giving normal equations of this least squares problem in terms of the wanted polynomial coefficients, one can also assemble those normal equations into 1 equation:

$$\frac{\partial l^{LS}(\theta)}{\partial \theta} = 2Re(\mathbf{J}^H \mathbf{J})\theta = 0 \tag{B.26}$$

The denominator coefficients α are used to obtain the poles and the modal participation factors, which are sufficient information for the constructing a stabilization diagram. Hence, one can further reduce the normal equations by setting:

$$\beta_o = R_o^{-1} S_o \alpha \tag{B.27}$$

This yields the reduced normal equation:

$$\begin{aligned}\left\{2\sum_{o=1}^{N_{out}}(T_o - S_o^T R_o^{-1} S_o)\right\}\alpha &= 0 \\ \mathbf{M}\alpha &= 0\end{aligned} \tag{B.28}$$

For a non-trivial solution to the normal equation, a constraint is set on α, where:

$$\alpha_p = \mathbf{I}_{N_{in}} \tag{B.29}$$

The rest of the denominator coefficients are then found using:

$$\mathbf{M}(1:N_{in}*p, 1:N_{in}*p)\begin{pmatrix}\alpha_0\\ \alpha_1\\ \alpha_2\\ \dots\\ \alpha_{p-1}\end{pmatrix} = \mathbf{M}(1:N_{in}*p, N_{in}*p+1:N_{in}*(p+1)) \tag{B.30}$$

The least-squares estimate for α is then:

$$\hat{\alpha}_{LS} = \begin{Bmatrix}\alpha\\ \mathbf{I}_{N_{in}}\end{Bmatrix}$$

This makes the denominator polynomial $\mathbf{D}$ a *monic* polynomial. Based on the fundamental definition of system poles, which is the points at which the system's response is "infinite", one can say that for an arbitrary rational polynomial $p(x)$:

$$p(x) = \frac{a_0 + a_1 x + a_2 x^2 + \dots + a_n x^n}{b_0 + b_1 x + b_2 x^2 + \dots + b_n x^n} \tag{B.31}$$

To achieve said "infinite" value, one must find values of x that represent the roots of the denominator polynomial. In the pLSCF model, one can exploit the monic property of the denominator polynomial. Frobenius companion matrices are square matrices that represent monic polynomials, given one has a monic polynomial

$$p(x) = x^n + a_{n-1}x^{n-1} + a_{n-2}x^{n-2} + \dots + a_1 x + a_0$$

The companion matrix of said polynomial is defined as:

$$C(p) = \begin{bmatrix}0 & 0 & \dots & 0 & -a_0\\ 1 & 0 & \dots & 0 & -a_1\\ 0 & 1 & \dots & 0 & -a_2\\ \vdots & \vdots & \ddots & \vdots & \vdots\\ 0 & 0 & \dots & 1 & -a_{n-1}\end{bmatrix} \tag{B.32}$$

A property of the companion matrix $C(p) \in \mathbb{R}^{n\times n}$ is that its eigenvalues $\lambda_1, \lambda_2, \dots, \lambda_n$ are the roots of $p(x)$, where $p(\lambda_i) = 0, \forall i = 1, 2, \dots, n$. This property aids in finding the system poles, after constructing the companion matrix for the denominator polynomial, an Eigendecomposition of the matrix yields the discrete time poles as the eigenvalues, and the corresponding eigenvectors are the modal participation factors:

$$C(\mathbf{D}) = \begin{pmatrix}0 & I & \dots & 0 & 0\\ 0 & 0 & I & \dots & 0\\ \dots & \dots & \dots & \dots & \dots\\ 0 & 0 & \dots & 0 & I\\ -\alpha_0^T & -\alpha_1^T & -\alpha_2^T & \dots - \alpha_{p-2}^T & -\alpha_{p-1}^T\end{pmatrix} \tag{B.33}$$

$$C(\mathbf{D})[\mathbf{L}] = \Lambda[\mathbf{L}] \tag{B.34}$$

Where the matrix $[\mathbf{L}]$ is the Eigenmatrix (matrix with columns as eigenvectors), representing the modal participation factors of each mode, and the matrix Λ contains the discrete time poles on its diagonal elements. The transpose of a companion matrix is used for pLSCF, this however does not affect the numerical value of the poles or participation factors [reference proof]. A p-order right matrix-fraction polynomial estimation should yield pN_{in} number of poles.

B.5 Finding the modeshapes using the Least-Squares Frequency Domain (LSFD) method

In practice, modeshapes are estimated using the Least Squares Frequency Domain algorithm, which is a linear regression designed to find the modal residues, a product of the modeshapes and the participation factors, and the upper and lower residuals, values which account for non-ideal vibratory behaviour. The LSFD fits a new model to measured FRF data as shown in Equation B.35.

$$H(s) = \sum_{m=1}^{N_{modes}} \left(\frac{\Psi_m \cdot L_m^T}{s - \lambda_m} + \frac{\Psi_m^* \cdot L_m^H}{s - \lambda_m^*} \right) + \frac{\mathbf{LR}}{s^2} + \mathbf{UR} \tag{B.35}$$

Where Ψ_m is the modeshape vector for the mode m, L_m is the mode's participation factor, $\mathbf{LR}$ and $\mathbf{UR}$ are the upper and lower residuals, and $s = j\omega$. By taking the Modal Residues for the i^{th} mode $[Res_i] = \Psi_m \cdot L_m^T$, they can be found, alongside the Upper and Lower residuals, in one single linear least squares step. By taking the Jacobian matrix J to be

$$J = \begin{bmatrix} \frac{1}{j\omega_2 - \lambda_1} + \frac{1}{j\omega_2 - \lambda_1^*} & \frac{1}{j\omega_2 - \lambda_2} + \frac{1}{j\omega_2 - \lambda_2^*} & \cdots & \frac{1}{j\omega_2 - \lambda_{N_m}} + \frac{1}{j\omega_2 - \lambda_{N_m}^*} & \frac{-1}{\omega_2^2} & 1 \\ \frac{1}{j\omega_3 - \lambda_1} + \frac{1}{j\omega_3 - \lambda_1^*} & \frac{1}{j\omega_3 - \lambda_2} + \frac{1}{j\omega_3 - \lambda_2^*} & \cdots & \frac{1}{j\omega_3 - \lambda_{N_m}} + \frac{1}{j\omega_3 - \lambda_{N_m}^*} & \frac{-1}{\omega_2^2} & 1 \\ \cdots & \cdots & \cdots & \vdots & \cdots & \vdots \\ \frac{1}{j\omega_{N_f} - \lambda_1} + \frac{1}{j\omega_{N_f} - \lambda_1^*} & \frac{1}{j\omega_{N_f} - \lambda_2} + \frac{1}{j\omega_{N_f} - \lambda_2^*} & \cdots & \frac{1}{j\omega_{N_f} - \lambda_{N_m}} + \frac{1}{j\omega_{N_f} - \lambda_{N_m}^*} & \frac{-1}{\omega_{N_f}^2} & 1 \end{bmatrix} \tag{B.36}$$

The residues and the upper and lower residuals can be estimated as:

$$\begin{bmatrix} Re[Res]_{1o} \\ \cdots \\ Re[Res]_{N_m o} \\ Im[Res]_{1o} \\ \cdots \\ Im[Res]_{N_m o} \\ [LR]_o \\ [UR]_o \end{bmatrix} = Re(J^H J)^{-1} Re(J^H H_o) \forall O = 1, 2, 3, \ldots, N_{out} \tag{B.37}$$

With no knowledge of the modal participation factors, the modeshape can only be estimated using a singular value decomposition (SVD) of the residue matrix.

$$[Res_i] = \mathbf{U \Sigma V^T} \tag{B.38}$$

Where $\mathbf{\Sigma}$ is a diagonal matrix containing the singular values of the matrix, $\mathbf{U}$ is a matrix containing left singular vectors, and $\mathbf{V}$ is a matrix containing right singular vectors. This process can yield accurate modeshapes only if the residue matrix is a rank-1, which mat not always be the case. Knowledge of the modal participation factors, which the pLSCF algorithm provides, removes this limitation by allowing a vector to be calculated using the properties of the outer product and rearranging. The use of the outer product provides quicker computation, as matrix-vector multiplication is generally considered quicker than the SVD, this is particularly important for the frequently encountered, high number of output systems.

$$\boldsymbol{\Psi}_i = \frac{[Res_i]\mathbf{L}_i}{\mathbf{L}_i^H \mathbf{L}_i} \tag{B.39}$$

Bibliography

[1] S. S. Rao and P. Griffin, *Mechanical vibrations*, 6th ed. Harlow, England: Pearson, 2018.

[2] N. M. M. Maia, D. D. Maio, A. Carrella, F. Marulo, C. Zang, J. E. Cooper, K. Worden, and T. A. N. Silva, *Structural Dynamics in Engineering Design*. Wiley, 2024.

[3] K. Worden and G. R. Tomlinson, *Nonlinearity in structural dynamics : detection, identification, and modelling*. Bristol [England] ; Philadelphia: Institute of Physics, 2001.

[4] D. Ewins, *Modal Testing: Theory, Practice and Application*, ser. Engineering dynamics series. Wiley, 2000.

[5] Z. Fu and J. He, *Modal Analysis*. Butterworth-Heinemann, 2001.

[6] N. M. M. Maia and J. M. J. M. Montalvao e Silva, *Theoretical and experimental modal analysis*, ser. Mechanical engineering research studies. Engineering dynamics series ; 9. Baldock : New York: Research Studies Press ; Wiley, 1997.

[7] C. Farrar and K. Worden, *Structural Health Monitoring A Machine Learning Perspective*. Wiley, 01 2013.

[8] A. Bunce, D. S. Brennan, A. Ferguson, C. O'Higgins, S. Taylor, E. J. Cross, K. Worden, J. Brownjohn, and D. Hester, "On population-based structural health monitoring for bridges: Comparing similarity metrics and dynamic responses between sets of bridges," *Mechanical Systems and Signal Processing*, vol. 216, p. 111501, 2024. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0888327024003996

[9] J. Maeck, B. Peeters, and G. DeRoeck, "Damage identification on the z24 bridge using vibration monitoring," *Smart Materials and Structures*, vol. 10, pp. 512–517, 06 2001.

[10] B. Peeters and G. De Roeck, "One year monitoring of the z24-bridge: Environmental influences versus damage events," *Proceedings of SPIE - The International Society for Optical Engineering*, vol. 2, 05 2000.

[11] S. Saeed, S. H. Sajid, and L. Chouinard, "Optimal sensor placement for enhanced efficiency in structural health monitoring of medium-rise buildings," *Sensors*, vol. 24, no. 17, 2024. [Online]. Available: https://www.mdpi.com/1424-8220/24/17/5687

[12] L. Bull, P. Gardner, J. Gosliga, T. Rogers, N. Dervilis, E. Cross, E. Papatheou, A. Maguire, C. Campos, and K. Worden, "Foundations of population-based shm, part i: Homogeneous populations and forms," *Mechanical Systems and Signal Processing*, vol. 148, p. 107141, 2021. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0888327020305276

[13] S. Tsiapoki and C. Colomer Segura, “Seven years shm of offshore wind turbine foundations: Review, experiences and outlook,” 2024, Conference paper, cited by: 0; All Open Access, Gold Open Access. [Online]. Available: https://www.scopus.com/inward/record.uri?eid=2-s2.0-85202581460&doi=10.58286%2f29681&partnerID=40&md5=23be5f28faab56d14121c846f24d1619

[14] J. MAECK and G. DE ROECK, “Description of z24 benchmark,” *Mechanical Systems and Signal Processing*, vol. 17, no. 1, pp. 127–131, 2003. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0888327002915487

[15] C. Rainieri and G. Fabbrocino, *Operational Modal Analysis of Civil Engineering Structures : An Introduction and Guide for Applications*, 1st ed. New York, NY: Springer New York : Imprint: Springer, 2014.

[16] M. Archila, R. Boroschek, C. Ventura, and S. Molnar, *Modal Testing of a Repaired Building After 2010 Chile Earthquake*, 05 2013, pp. 119–125.

[17] P. Reynolds, I. Díaz, and D. Nyawako, “Vibration testing and active control of an office floor,” 01 2009.

[18] Z. Sun, G. C. Steyer, G. Meinhardt, and M. Ranek, “Nvh robustness design of axle systems,” *SAE Transactions*, vol. 112, pp. 1746–1754, 2003. [Online]. Available: http://www.jstor.org/stable/44745550

[19] E. Abe and H. Hagiwara, “Advanced method for reduction in axle gear noise,” *SAE Transactions*, vol. 84, pp. 670–682, 1975. [Online]. Available: http://www.jstor.org/stable/44681958

[20] J. W. Martz, E. L. Peterson, G. W. Knobeloch, and G. D. Angus, “Four steps for vehicle ride improvement,” *SAE Transactions*, vol. 88, pp. 814–827, 1979. [Online]. Available: http://www.jstor.org/stable/44633919

[21] M. French and M. Jay, “An introduction to automotive nvh testing,” *Experimental Techniques*, vol. 22, no. 4, pp. 32–33, 1998. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1111/j.1747-1567.1998.tb02336.x

[22] Y. Liu, H. Dai, P. Wu, Z. Jing, Y. Song, and T. Li, *Research on Vibration and Noise of SUV Chassis Suspension System*, 2020, pp. 307–310.

[23] J. Wright and J. Cooper, *Introduction to Aircraft Aeroelasticity and Loads*, ser. Aerospace Series. Wiley, 2008. [Online]. Available: https://books.google.co.uk/books?id=BU_tRQaz9gIC

[24] D. A. Chamberlain and C. K. Mechefske, “Experimental modal analysis of a half-scale model twin-engine aircraft rear fuselage engine mount support frame,” *Proceedings of the ASME Design Engineering Technical Conference*, vol. 8, 2017, cited by: 7. [Online]. Available: https://www.scopus.com/inward/record.uri?eid=2-s2.0-85034862660&doi=10.1115%2fDETC2017-67389&partnerID=40&md5=314237f6ffca59db53004dd5b88f91ce

[25] Z. Saffry, D. L. Majid, F. I. Romli, F. Mustapha, and E. J. Abdullah, “Identification of modal properties of composite thin plate using oma in wind tunnel environment,” in *Advanced Research in Material Science and Mechanical Engineering*, ser. Applied Mechanics and Materials, vol. 446. Trans Tech Publications Ltd, 1 2014, pp. 606–610.

[26] J. Lau, B. Peeters, J. Debille, Q. Guzek, W. Flynn, D. Lange, and T. Kahlmann, *Ground Vibration Testing Master Class: modern testing and analysis concepts applied to an F-16 aircraft*, 04 2011, pp. 221–228.

[27] S. T. Thornton and J. B. Marion, *Classical dynamics of particles and systems*. Andover: Cengage Learning, 2014. [Online]. Available: https://www.worldcat.org/title/classical-dynamics-of-particles-and-systems/oclc/951438161&referer=brief_results

[28] A. Awadalla, "Developing an open-source frequency domain modal analysis algorithm in python - interim report," School of Mechanical, Aerospace, and Civil Engineering, University of Sheffield, Tech. Rep., 2024.

[29] T. J. Rogers, "Lecture 5 - forced vibration," Lecture Slides for MEC326 Course, Department of Mechanical Engineering, University of Sheffield, 2023, accessed on: Dec. 1st, 2024.

[30] ——, "Lecture 6 - modal properties 2," Lecture Slides for MEC326 Course, Department of Mechanical Engineering, University of Sheffield, 2023, accessed on: Dec. 1st, 2024.

[31] P. Guillaume, P. Verboven, and S. Vanlanduit, "Frequency domain maximum likelihood identification of modal parameters with confidence intervals," 01 1998.

[32] P. Guillaume, P. Verboven, S. Vanlanduit, H. Van der Auweraer, and B. Peeters, "A poly-reference implementation of the least-squares complex frequency-domain estimator," *Proceedings of IMAC*, vol. 21, 01 2003.

[33] B. Peeters, H. Van Der Auweraer, P. Guillaume, and J. Leuridan, "The polymax frequency-domain method: A new standard for modal parameter estimation?" *Shock and Vibration*, vol. 11, no. 3-4, p. 395 – 409, 2004, cited by: 912; All Open Access, Gold Open Access. [Online]. Available: https://www.scopus.com/inward/record.uri?eid=2-s2.0-4644323283&doi=10.1155%2f2004%2f523692&partnerID=40&md5=564fa529224241631a3740c34109f97a

[34] B. Peeters, P. Guillaume, H. Van der Auweraer, B. Cauberghe, P. Verboven, and J. Leuridan, "Automotive and aerospace applications of the polymax modal parameter estimation method," *Proceedings of IMAC 22, the International Modal Analysis Conference*, 01 2004.

[35] B. Peeters and H. Van der Auweraer, "Polymax: a revolution in operational modal analysis," *1st International Operational Modal Analysis Conference*, 01 2005.

[36] B. Peeters, H. Van der Auweraer, J. Leuridan, and T. Vasel, "Polymax modal parameter estimation: Challenging automotive and aerospace applications," pp. 1–13+572, 01 2004.

[37] C. R. Harris, K. J. Millman, S. J. van der Walt, R. Gommers, P. Virtanen, D. Cournapeau, E. Wieser, J. Taylor, S. Berg, N. J. Smith, R. Kern, M. Picus, S. Hoyer, M. H. van Kerkwijk, M. Brett, A. Haldane, J. F. del Río, M. Wiebe, P. Peterson, P. Gérard-Marchant, K. Sheppard, T. Reddy, W. Weckesser, H. Abbasi, C. Gohlke, and T. E. Oliphant, "Array programming with NumPy," *Nature*, vol. 585, no. 7825, pp. 357–362, Sep. 2020. [Online]. Available: https://doi.org/10.1038/s41586-020-2649-2

[38] P. Diehl, M. Morris, S. R. Brandt, N. Gupta, and H. Kaiser, *Benchmarking the Parallel 1D Heat Equation Solver in Chapel, Charm++, C++, HPX, Go, Julia, Python, Rust, Swift, and Java*. Springer Nature Switzerland, 2024, p. 127–138. [Online]. Available: http://dx.doi.org/10.1007/978-3-031-48803-0_11

[39] A. Almurayh, “Improved matrix multiplication by changing loop order,” *Mobile Information Systems*, vol. 2022, no. 1, p. 9650652, 2022. [Online]. Available: https://doi.org/10.1155/2022/9650652

[40] P. Virtanen, R. Gommers, T. E. Oliphant, M. Haberland, T. Reddy, D. Cournapeau, E. Burovski, P. Peterson, W. Weckesser, J. Bright, S. J. van der Walt, M. Brett, J. Wilson, K. J. Millman, N. Mayorov, A. R. J. Nelson, E. Jones, R. Kern, E. Larson, C. J. Carey, İ. Polat, Y. Feng, E. W. Moore, J. VanderPlas, D. Laxalde, J. Perktold, R. Cimrman, I. Henriksen, E. A. Quintero, C. R. Harris, A. M. Archibald, A. H. Ribeiro, F. Pedregosa, P. van Mulbregt, and SciPy 1.0 Contributors, “SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python,” *Nature Methods*, vol. 17, pp. 261–272, 2020.

[41] Wes McKinney, “Data Structures for Statistical Computing in Python,” in *Proceedings of the 9th Python in Science Conference*, Stéfan van der Walt and Jarrod Millman, Eds., 2010, pp. 56 – 61.

[42] J. D. Hunter, “Matplotlib: A 2d graphics environment,” *Computing in Science and Engineering*, vol. 9, no. 3, pp. 90–95, 2007.

[43] N. Cheshire, “Why coding standards?” *ADTmag*, February 2005. [Online]. Available: https://adtmag.com/Articles/2005/02/01/Why-Coding-Standards.aspx

[44] M. Andra, “Two strategies for writing better code,” Marius Andra’s Blog, September 2012. [Online]. Available: https://mariusandra.com/blog/2012/09/two-strategies-for-writing-better-code/

[45] R. C. Martin, *Clean Code: A Handbook of Agile Software Craftsmanship*. Upper Saddle River, NJ: Prentice Hall PTR, 2008.

[46] Istanbul Technical University, Dept. of Computer Engineering, “Operator overloading,” Lecture notes, Object Oriented Programming course, n.d., accessed May 7, 2025. [Online]. Available: https://web.itu.edu.tr/bkurt/Courses/blg332e/week5.pdf

[47] T. Jenkins, “Yet another python book,” Online; last updated April 22, 2025, April 2025, section 9.1.1 “Code is Read”, accessed May 7, 2025. [Online]. Available: https://www.tony-jenkins.org.uk/pybook/

[48] G. van Rossum, B. Warsaw, and A. Coghlan, “Style guide for python code (pep 8),” Python Software Foundation, Python Enhancement Proposal PEP 8, July 2001. [Online]. Available: https://peps.python.org/pep-0008/