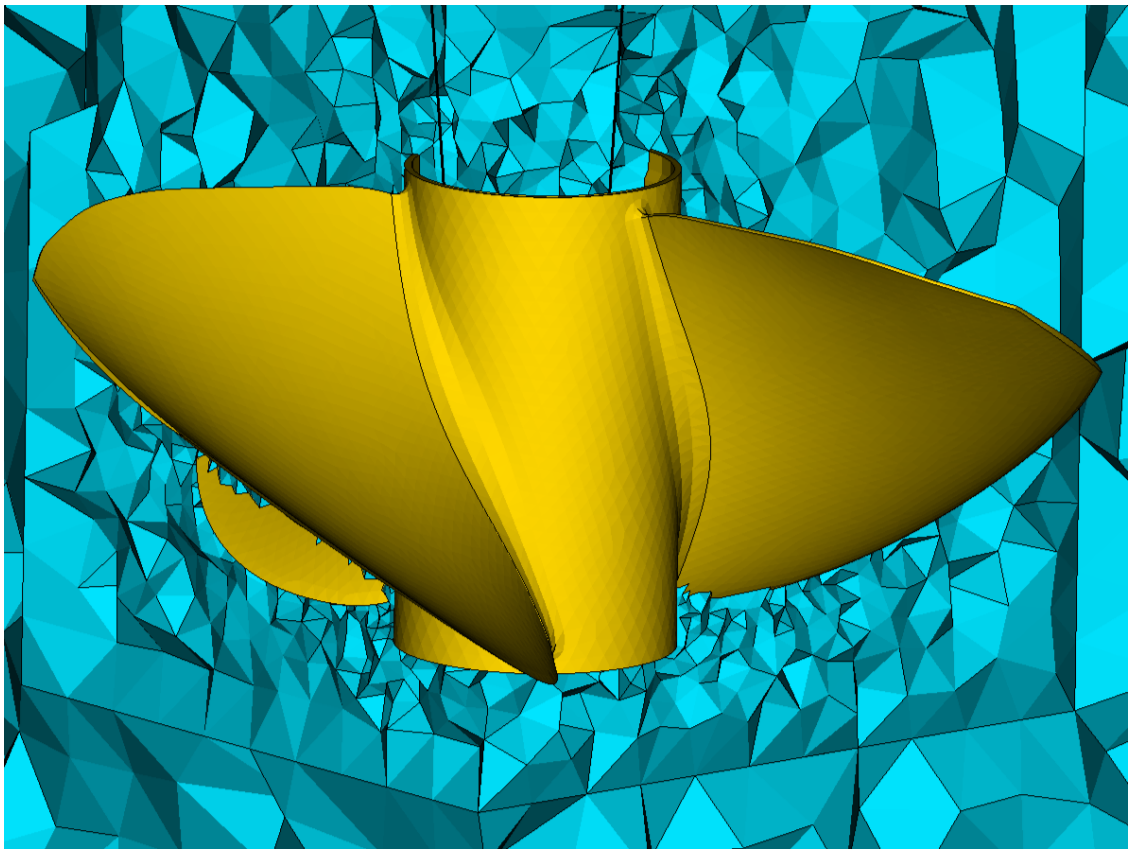


Model Updating and Calibration of a Finite Element Model of Marine Propeller Through Measurements and Simulation

Master's thesis in the Master's Programme Sound and Vibration

Kshitij Ravindra Patil

DEPARTMENT OF ARCHITECTURE AND CIVIL ENGINEERING
DIVISION OF APPLIED ACOUSTICS

CHALMERS UNIVERSITY OF TECHNOLOGY
Master's thesis ACEX30
Gothenburg, Sweden 2025

MASTER'S THESIS ACEX30

Model Updating and Calibration of a Finite Element Model of Marine Propeller Through Measurements and Simulation

Master's Thesis in the Master's Programme Sound and Vibration

Kshitij Ravindra Patil

Department of Architecture and Civil Engineering
Division of Applied Acoustics
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2026

Model Updating and Calibration of a Finite Element Model of Marine Propeller Through Measurements and Simulation
Master's Thesis in the Master's Programme Sound and Vibration

Kshitij Ravindra Patil

© Kshitij Ravindra Patil, 2026.

Examensarbete ACEX30
Institutionen för Arkitektur och Samhällsbyggnadsteknik
Chalmers Tekniska Högskola, 2026

Department of Architecture and Civil Engineering
Division of Applied Acoustics
Chalmers University of Technology
SE-412 96 Göteborg
Sweden
Telephone +46 31 772 1000

Cover:
Submerged Finite Element model of the propeller under investigation.
Department of Architecture and Civil Engineering
Göteborg, Sweden, 2026

Model Updating and Calibration of a Finite Element Model of Marine Propeller Through Measurements and Simulation
Master's thesis in the Master's Programme Sound and Vibration

Kshitij Ravindra Patil

Department of Architecture and Civil Engineering
Division of Applied Acoustics
Chalmers University of Technology

ABSTRACT

This thesis presents a practical workflow for calibrating finite-element (FE) models to experimental modal data for structures tested in air and in water. The method follows a frequency-response based calibration with damping equalization so that measured and simulated frequency responses can be compared in a consistent way before parameter tuning [Abrahamsson and Kammer, 2015]. For submerged cases, the fluid is modeled as a potential-based acoustic domain to account for added mass and radiation damping in line with standard fluid-structure interaction (FSI) theory [Richter, 2017].

The study is organized in two stages. First, a suspended plate is used as a benchmark to develop the end-to-end pipeline; second, the workflow is transferred to a marine propeller with practical constraints on where excitation and sensing can be performed. Roving-hammer testing provides single-output, multi-input frequency response functions. The computational toolchain combines ANSA for geometry, meshing, and coupling definition, ADINA for coupled structure-acoustic eigenanalysis, META for post-processing and data export, and Python scripts for peak selection, mode pairing, objective construction, and optimization. The calibration proceeds in a frequency-only stage with gated peak pairing to avoid mode swapping, followed by an optional magnitude stage that uses per-input gain factors to accommodate test variability. The workflow, scripts, and configuration files are designed to be reproducible and transferable, enabling application to both suspended and submerged configurations while documenting assumptions and limitations that arise from test setup and identifiability considerations [Richter, 2017].

Key words: FRF calibration, damping equalization, model reduction, MAC, added mass, radiation damping, Fluid-Structure Interaction

Contents

ABSTRACT I

SAMMANFATTNING I

CONTENTS IV

PREFACE V

1 INTRODUCTION 1
- 1.1 Problem Statement 1
- 1.2 Objectives 1
- 1.3 Scope and Assumptions 2
- 1.4 Method Overview and Contributions 2

2 BACKGROUND AND THEORY 3
- 2.1 Structural Dynamics and Modal Analysis 3
  - 2.1.1 Equations of Motion (Discrete FE Form) 3
  - 2.1.2 Natural Frequencies and Mode Shapes 3
  - 2.1.3 Light Damping and Why Fixed "Equalized" Damping is Used 3
  - 2.1.4 Frequency-Response Functions (FRFs) 3
  - 2.1.5 Mode-Shape Correlation and Pairing (MAC) 4
  - 2.1.6 Sensitivities for Calibration 4
  - 2.1.7 Reduced "Diagonal" Model Used Inside The Optimizer 4
  - 2.1.8 Two Objective Types Used in Practice 4
  - 2.1.9 Special Care With Triplet Modes (Propeller) 5
  - 2.1.10 What Makes Parameters Identifiable 5
  - 2.1.11 What Changes in Water (FSI, Simple Acoustic View) 6

3 SOFTWARE STACK AND DATA FLOW 7
- 3.1 Software Used and Their Roles 7
- 3.2 Folder structure 8
- 3.3 File Formats and Why They are Used 9

4 EXPERIMENTAL SETUP 11
- 4.1 Test Articles 11
  - 4.1.1 Rectangular Aluminum Plate (Suspended and Submerged) 11
  - 4.1.2 Three-Bladed Propeller (Suspended and Submerged) 12
- 4.2 Suspension and Boundary Conditions 13
- 4.3 Measurement Instruments 13
- 4.4 Measurement Grids 14
  - 4.4.1 Limitations of the Propeller Measurement Layout (and Implications for Calibration) 15
- 4.5 Data Acquisition Settings 16
- 4.6 Test Procedure 16
- 4.7 Data Export and Storage 17

5 FE MODELING 18
5.1 Software Roles in the Modeling Chain 18
5.2 Geometry and Meshing 18
5.2.1 Plate 18
5.2.2 Propeller 19
5.3 Boundary Conditions and Suspension Modeling 21
5.3.1 Suspended (air) 21
5.3.2 Submerged (water) 21
5.4 Modal Analysis Settings 21
5.5 Outputs for Calibration 21

6 CALIBRATION METHODOLOGY 23
6.1 Purpose, Scope, and Design Choices 23
6.2 Data Inputs and Outputs 23
6.3 Measurement Damping Equalization 23
6.4 Reduced Model From FE 24
6.5 Pairing With MAC and Gates 24
6.6 FRF Comparison with Equalized Damping and One Global Gain 24
6.7 What is Minimized and in What Order 25
6.8 Choosing Parameters 26
6.9 Optimizer Logic 26
6.10 Quality Control Files 27
6.11 Problems Met 27

7 RESULTS 28
7.1 Suspended Plate (Air) 28
7.1.1 Inputs 28
7.1.2 Parameter, Bounds and Steps 28
7.1.3 Pairing Quality (MAC) 28
7.2 Submerged Plate (Water) 31
7.2.1 Inputs 31
7.2.2 Parameter, Bounds and Steps 32
7.2.3 Pairing Quality (MAC) 32
7.3 Suspended Propeller 34
7.3.1 Inputs 34
7.3.2 Parameters, Bounds and Steps 34
7.3.3 Pairing Quality (MAC) 35
7.4 Submerged Propeller 36
7.4.1 Inputs 36
7.4.2 Parameters, Bounds and Steps 36
7.4.3 Pairing Quality (MAC) 37

8 DISCUSSION 41

9 CONCLUSIONS 42

10 FUTURE WORK 43

10.1 Measurement Plan Improvements for the Propeller 43
10.2 Model and Parameter Set 43
10.3 Robust Pairing and Identifiability 43
10.4 Tooling and Automation 43

REFERENCES 45

APPENDIX I
10.5 00_unv_to_csv.py I
10.6 01_build_sysid_equalize.py V
10.7 02_build_reduction_from_op2.py X
10.8 03_calibrate.py XV

Preface

In this study, a practical approach to FE model calibration based on Dr. T. Abrahamsson's work has been explored. The project has been carried out between January 2025 to December 2025.

This thesis has been carried out in collaboration with Volvo Penta AB, Sweden. The measurement tests required for this model calibration routine were carried out in Volvo Penta's test facility at Krossholmen. The FE modeling and calibration work were carried out at the Volvo Penta office at Lundby.

The measurement and testing at Krossholmen were guided and supervised under Jonas Tornqvist. The FE modeling and calibration work have been carried out with the support and guidance of Kenth Nilsson, who acted as the supervisor for this thesis project.

The thesis is part of the Department of Applied Acoustics, Chalmers University of Technology, Sweden. I would like to thank Wolfgang Kropp for his help and support as the examiner for this project. I would also like to thank my colleague Qianyi Song, who acted as an opposition for the thesis defense.

Finally, I extend my gratitude to my family, the entire team at Volvo Penta, at Chalmers University, and everyone involved in supporting me throughout this process.

Gothenburg, January 2026
Kshitij Ravindra Patil

1 Introduction

In engineering, there is an increasing reliance on finite-element (FE) models to predict dynamic behavior of structures that operate in both air and water– such as marine propellers. To be trustworthy, these models must be calibrated against measurements so that their natural frequencies, mode shapes, and frequency responses match what is observed in tests. For submerged structures, calibration must also reflect added mass and radiation damping from the surrounding fluid, which requires consistent fluid-structure interaction (FSI) formulation [Richter, 2017]. A practical, repeatable workflow that joins laboratory measurements with FE modeling and an optimizer is therefore valuable to both research and industry.

1.1 Problem Statement

This thesis seeks to calibrate FE models of two systems: (i) a plate tested in air and in water, and (ii) a suspended three-bladed propeller, so that simulated modal properties align with experimental data from roving-hammer tests. The work must handle typical constraints: single accelerometer output, many input (impact) points, possible unit and orientation mismatches between test and FE coordinates, near degenerate ("triplet") modes in the propeller, and practical issues in exporting modal data from solvers and post-processors. The central technical question is how to convert raw measurement data and FE modal results into a straightforward calibration problem that remains stable in air and under water. The calibration follows a frequency-based method with damping equalization so measurements and simulations can be compared on a consistent basis before tuning parameters[Abrahamsson and Kammer, 2015].

1.2 Objectives

Objectives

1. Build a reproducible end-to-end workflow that takes UNV measurement data, exports FE modal data, and performs calibration with damping equalization [Abrahamsson and Kammer, 2015].
2. Integrate a potential-based acoustic fluid in the FE model for submerged cases to represent added mass and radiation damping in line with linear Fluid Structure Interactions theory [Richter, 2017].
3. Implement robust data conditioning: title-based UNV parsing, axis selection, frequency-grid checks, peak detection, and mode-pairing with modal assurance criteria (MAC).
4. Construct reduced-order calibration models using eigen-based sensitivities from finite differences (forward or central), with bounds, priors, and observability diagnostics.
5. Document practical guidelines for parameter selection, frequency-band selection, etc.

1.3 Scope and Assumptions

- **Articles under study:** A rectangular aluminum plate (air and water) and a three-bladed propeller (Volvo Penta) in air and submerged.
- **Measurements:** Roving-hammer tests 80 impact points for the plate and 24 impact points for the propeller and one fixed accelerometer in both cases. Single-input/linear hammer taps are assumed (guarded by coherence checks)
- **Modeling (air):** Solid elements for structures, and suspension represented by linear springs to approximate elastics bands.
- **Modeling (water):** Acoustics fluid domain coupled to the structure and pressure-release free surface on top surface are some of the considerations for the submerged case modeling.
- **Linearity:** Small-amplitude vibrations and linear material behavior are assumed for both test and model stages.

It is important to note that, for the purpose of this thesis, the geometries of the articles under study are assumed to be ideally manufactured and identical to their CAD models.

1.4 Method Overview and Contributions

The thesis adapts a frequency-response-based calibration with damping equalization to lab measurement data, then extends it to submerged conditions via a potential-flow acoustic fluid [Abrahamsson and Kammer, 2015; Richter, 2017]. The workflow is organized into three programmed stages:

- **Stage 01 (Equalized FRFs):** Convert measured UNV (Universal file format used to store measurement data) blocks to per-input CSVs, select the response axis, and build an "equalized" FRF pack to neutralize damping differences before comparison [Abrahamsson and Kammer, 2015].
- **Stage 02 (Reduced model and gradients):** Export nominal parameters and +/- delta parameters FE modal sets (frequencies and mode shapes), pair modes with MAC (Hungarian assignment) under gating, and assemble diagonal gradients of ω^2 with respect to parameters (forward or central differences).
- **Stage 03 (Calibration):** Solve a **frequency-only** least-squares with gated peak pairing, optionally follow with a **log-magnitude FRF** least-squares that includes per-input gains to avoid force/response scaling differences. Bounds and active-parameter flags manage search space and identifiability.

Contributions include: (i) a scriptable pipeline that runs end-to-end on suspended and submerged data, (ii) a robust UNV parsing and grid-checking layer, (iii) a reduced-order calibration tolerant to partial mode matches and triplets, and (iv) practical guidance on bands, parameters, and minimal test upgrades that most improve observability [Abrahamsson and Kammer, 2015].

2 Background and Theory

This section summarizes the ideas needed to build and calibrate dynamic FE models from test data. It introduces the equations of motion, modal analysis, frequency-response functions (FRFs), and practical topics like damping, peak picking, and mode-shape correlation. It also highlights what changes when the structure is in water and why a simple acoustic fluid model is often enough for calibration purposes [Richter, 2017]. Finally, it addresses the idea of damping equalization in frequency-response based calibration, which is used later in the workflow [Abrahamsson and Kammer, 2015].

2.1 Structural Dynamics and Modal Analysis

2.1.1 Equations of Motion (Discrete FE Form)

For a linear FE model, the equation of motion is written as

$$M\ddot{x}(t) + C\dot{x}(t) + Kx(t) = f(t) \tag{2.1}$$

with mass matrix M, damping matrix C, stiffness matrix K, displacement vector x, and external force f (e.g. the hammer). This forms the basis for both the simulation and test-based calibration.

2.1.2 Natural Frequencies and Mode Shapes

With no forcing and no damping, the free-vibration problem looks like

$$(K - \omega^2 M)\phi = 0 \tag{2.2}$$

The solutions give natural angular frequencies ω_i and real mode shape ϕ_i.
Modes are often mass-normalized so that

$$\phi_i^T M \phi_j = \delta_{ij} \tag{2.3}$$

which makes later formulas simpler and matches how MAC (Modal Assurance Criterion) is computed.

2.1.3 Light Damping and Why Fixed "Equalized" Damping is Used

In small-amplitude hammer tests on metal parts, damping is usually light and can be represented by a per-mode ratio ζ_i.
In calibration it is hard to know the real damping distribution and the exact input/output gains, so the workflow compares data and model under a common, fixed damping level (e.g., 0.01 for all modes) before tuning parameters. This is the idea behind damping equalization that Dr. Thomas Abrahamsson talks about, whose paper forms the basis of this thesis. The equalization reduces bias from unknown damping and gain and focuses the match on peaks and overall shape [Abrahamsson and Kammer, 2015].

2.1.4 Frequency-Response Functions (FRFs)

Taking the Fourier transform gives us the FRF matrix

$$H(\omega) = (-\omega^2 M + j\omega C + K)^{-1} \tag{2.4}$$

Depending on the measured quantity, we usually use receptance (displacement/force), mobility (velocity/force), or accelerance (acceleration/force). The hammer and accelerometer tests in this case give us accelerance FRFs.
Near a resonance, a single output / single input FRF can be written as a sum of modal terms

$$H(\omega) \approx \sum \frac{C_i U_i}{(\omega_i^2 - \omega^2) + 2j\zeta_i\omega_i\omega} \tag{2.5}$$

where U_i and C_i tell us how the input and output participate in mode i. This explains why FRFs show peaks at the natural frequencies and why peak width reflects damping.

2.1.5 Mode-Shape Correlation and Pairing (MAC)

The Modal Assurance Criterion (MAC) compares two mode shapes **a** and **b** as

$$MAC(a, b) = \frac{|a^T b|^2}{(a^T a)(b^T b)} \in [0, 1] \tag{2.6}$$

High MAC (close to 1) means that the shapes are very similar.
The workflow uses MAC with an assignment step (Hungarian algorithm) to pair nominal and +/- delta FE modes and applies gates (minimum MAC and maximum allowed frequency separation) to avoid bad pairs.

2.1.6 Sensitivities for Calibration

The calibration needs to know how ω_i^2 changes when a parameter p_k changes.
Exact derivatives are rarely available for full FE models, so the scripts use finite differences from nominal and +/- delta FE modal solutions:

- **Forward difference:**

$$\partial\omega_i^2/\partial p_k \approx (\omega_{i,+}^2 - \omega_{i,0}^2)/\Delta p_k \tag{2.7}$$

- **Central difference:**

$$\partial\omega_i^2/\partial p_k \approx (\omega_{i,+}^2 - \omega_{i,-}^2)/(2\Delta p_k) \tag{2.8}$$

Central differences reduces bias but will require one extra solution per parameter.

2.1.7 Reduced "Diagonal" Model Used Inside The Optimizer

After pairing, the nominal squared frequencies $\omega_{i,0}^2$ are adjusted with a linear sensitivity model

$$\omega_i^2(p_0 + \Delta p) \approx \omega_{i,0}^2 + \sum_k \frac{\partial\omega_i^2}{\partial p_k}\Delta p_k \tag{2.9}$$

This creates an effective diagonal stiffness in modal space. With a fixed damping level (equalized) and simple input/output selectors, this gives us a closed-form FRF used to compare with the measure, equalized FRF magnitude.

2.1.8 Two Objective Types Used in Practice

- **Stage-A: Frequency-only** Detect reliable peaks (by prominence), pair them with a gated rule, and minimize frequency errors (optionally weighted). This is robust and usually run first.

- **Stage-B: FRF log-magnitude (optional)** Fit the equalized FRF magnitudes using the reduced model and allow a per-input gain to absorb small force/response scaling differences. This is only used once Stage-A aligns the peaks reasonably well [Abrahamsson and Kammer, 2015].

2.1.9 Special Care With Triplet Modes (Propeller)

A three bladed propeller is almost cyclically symmetric (three identical sectors shifted $120°$). In an ideal perfectly symmetric structure, this symmetry makes some eigenvalues repeated: one "axisymmetric" family (single mode) and a "whirling" family that is a double (two linearly independent shapes at the same frequency). This is a standard result from cyclical-symmetry theory in turbomachinery and rotor dynamics (the three sector case produces one single and one doublet). In reality, the symmetry is never perfect. Small geometric/ material differences between blades ("mistuning"), slightly uneven suspension, or measurement equipment loading break the exact symmetry. When that happens, the repeated frequencies split into several nearby frequencies. With three sectors, this often shows up as three close peaks whose mode shapes look like the "same blade bending/torsion pattern" rotated by $120°$ around the hub, i.e., each blade has a very similar local mode at a slightly different frequency due to mistuning [Ewins, 2000]. In short:

- **Ideal C3 symmetry** $\rightarrow$ one single + one doublet (exactly repeated).
- **Real, slightly mistuned C3** $\rightarrow$ the doublet splits; with noticeable mistuning, we observe very close peaks that look like rotated versions of the same blade-dominant shape (mode localization on each blade).
- **Single-output testing** $\rightarrow$ different hammer locations excite different linear combinations of these near-duplicate shapes, so which peak is "strongest" may change across inputs.

This matters for mode pairing and calibration because the three peaks are close together and look very similar, simple "nearest frequency" pairing will swap modes between runs (or between nominal and delta FE solutions). If pairing swaps, the finite-difference gradients

$$\partial\omega^2/\partial p$$

become noisy or wrong, and the optimizer chases false values instead of physics. With only one accelerometer (single output), MAC values between these shapes can also be deceptive across the trio, making it harder to decide which is the true pair.

2.1.10 What Makes Parameters Identifiable

A parameter is identifiable if:

1. it causes noticeable shifts in the chosen peaks,
2. there are enough clean peaks in the selected band, and
3. its effect is not almost the same as another parameter's effect.

If the gradients are small, peaks are few/noisy, or gradients are nearly collinear, the problem becomes ill-conditioned. The remedy would be to narrow or shift the band, use central differences, or drop/freeze weak parameters.

2.1.11 What Changes in Water (FSI, Simple Acoustic View)

In water, the structure is coupled to an acoustic fluid that adds mass (peaks move down) and radiation damping (peaks broaden).
In a linear approximation, the acoustic pressure satisfies a Helmholz type equation, and the fluid structure interface enforces normal velocity continuity. The free surface can be modeled as a pressure release where applicable [Richter, 2017].
In FE, this is handled by adding an acoustic fluid mesh and including fluid properties ρ_f and bulk modulus K_f (where speed of sound $c = \sqrt{K_f/\rho_f}$) in the model. These can then be included as calibration parameters [Richter, 2017].

3 Software Stack and Data Flow

This section explains how the software tools are used, what data is stored at each step, and why specific file formats are chosen.

3.1 Software Used and Their Roles

BETA CAE Systems – ANSA
ANSA is a commercial CAE pre-processor used to bring CAD into analysis-ready state through geometry cleanups, topology checks, part organization, and meshing.

How it is used here: ANSA is used for geometry clean-up and meshing of the structure (plate/propeller) and, for submerged cases, building the surrounding fluid volume. No material definitions or solver controls are assigned in ANSA. The output is a Nastran file (.nas) containing nodes, elements, and geometry.

Bentley Systems – ADINA
ADINA is a general-purpose finite element program that supports structural dynamics, fluids, and fluid-structure interaction (FSI). It provides eigenvalue extraction for both pure structural and coupled fluid-structure systems.

How it is used here: ADINA reads the ANSA-exported .nas mesh and is the only place where materials and physics are defined. For suspended (air) cases, structural materials (e.g. E,ν, ρ) and elastic suspensions (spring elements) are specified, and a structural modal analysis is performed. For submerged cases, a potential-based fluid is added (water density and bulk modulus), and the top free surface is set to pressure-release. The submerged structural surface is coupled to the fluid domain to form the FSI model. ADINA produces eigenfrequencies and mode shapes and is saved as an .op2 solution file. Since reading the OP2 file can be challenging, frequencies are read from the solver's .out file and the mode shapes are exported via META to CSV files for clarity and consistency.

BETA CAE Systems – META
META (META-Post) is a CAE post-processor for visualizing results, comparing models to tests, computing modal correlation metrics (e.g. MAC), and exporting data. It reads FE result formats and measurement files such as Nastran (.nas), .op2, and Universal files (.unv).

How it is used here: META is used to: (i) open ADINA OP2 results and export clean comma-separated data like `mode_###.csv` per mode containing columns like `node_id`, `Dispx`, `Dispy`, `Dispz` that describe the mode shape. And (ii), open measurement UNVs to check axes/units. Keeping this export in CSV removes uncertainty about units and coordinate systems and makes scripts reliable.

Siemens Simcenter Testlab
Simcenter Testlab is used for data acquisition and modal testing. It can export the measurement frequency response functions (FRFs) to the neutral UNV format.

How it is used here: The UNV produced by Testlab is the raw data. A python script

reads the UNV and writes one FRF CSV per hammer input with columns `f_hz`, `re`, `im`. The UNV itself is never altered.

Python
Python provides a reproducible environment for data conversion, model reduction, and parameter estimation. NumPy/SciPy handle the numerical work, Pandas handles tabular I/O, YAML/JSON/NPZ handle configurations and compact binary data. Spyder has been used as the choice of IDE on Windows.

How it is used: Four scripts implement the pipeline:

1. **00** converts measurement UNV → per-input FRF CSVs.
2. **01** builds a compact, equalized FRF pack (single NPZ with a shared frequency grid).
3. **02** assembles a reduced modal model from FE CSV exports and computes frequency sensitivity gradients (forward or central differences).
4. **03** performs calibration in two stages (frequency-only, then optionally an FRF magnitude), writing a readable `summary.txt`.
   Using CSV for tables (FRFs, modes, mode frequencies), NPZ for fast exact dtype arrays (equalized FRFs and reduced model), and YAML for readable configurations for the scripts (paths, bands, parameters) balances transparency with speed and makes the workflow robust and repeatable.

Order of use in this project: ANSA (mesh) → ADINA (define physics and solve) → META (export modes CSVs and post processing) → Python (convert UNV, build equalized FRFs, build reduced model, calibrate)

3.2 Folder structure

The project uses a simple, stable layout so scripts can rely on relative paths:

```
  <ProjectRoot>/
calib_config.yaml            # config for the suspended case
calib_config_wet.yaml        # config for the submerged case
selectors_nodes.yaml         # FE node IDs for selectors

01_raw/
  <measurement>.unv          # raw UNV from Simcenter
  frf_meas/                  # FRF CSVs (one CSV/hammer point)
  frf_meas_wet/              # FRF CSVs for submerged
                               measurements
02_proc/
  HXeq_FRF.npz               # equalized FRF pack (suspended)
  HXeq_FRF_wet.npz           # equalized FRF pack (submerged)

03_fe/
  op2_csv/
    nominal/
```

```
      frequencies.csv          # table of mode_id,freq_hz (Hz)
      modes/
        mode_007.csv           # node_id,Dispx,Dispy,Dispz
        mode_008.csv             (global, m)
        ...
    <param>_plus/              # e.g. rho_plate_plus (+delta file
      frequencies.csv            for plate density)
      modes/
        mode_###.csv
    <param>_minus/             # present when using central diffs
   reduced/
    reduced_model.npz          # nominal w0 and gradient diagonals
    reduced_model_wet.npz        G_i
    reduced_model*.qc.json     # MAC statistics and matched counts

04_results/
  best_params.json             # calibrated parameter values
  summary.txt                  # readable summary of the run
scripts/
  00_unv_to_csv.py
  01_build_sysid_equalize.py
  02_build_reduction.py
  03_calibrate.py
```

This structure separates raw data (`01_raw`) from processed packs (`02_proc`), FE exports and reductions (`03_fe`), and final results (`04_results`). Since each stage writes its own outputs, running one step again does not risk overwriting earlier raw data.

3.3 File Formats and Why They are Used

UNV (Universal File Format)
The UNV from Simcenter is kept as the main raw measurement. It contains many "function" blocks, each labeled with a title that names the channel pair (for example, Acc:001:+Z/Hammer:010:-Z). Each block stores a frequency vector and complex FRF values. The UNV is kept unchanged so that FRF CSVs can be regenerated consistently whenever needed.

CSV for FRFs (measurement)
Each hammer impact point is written as a single CSV with three columns: frequency in Hz (`f_hz`) and the real and imaginary parts of the FRF (`re, im`). CSV is chosen because it is readable, easy to audit, and universally supported. Storing one file per input keeps the files small and makes it easy to see if any input is missing.

NPZ for damping equalized FRFs (measurement pack)
The damping equalized pack (`HXeq_FRF.npz`) stores a single frequency vector and a 3-D complex array containing all FRFs aligned on that grid. NPZ is a zipped file format, and it allows exact data types (`float64` and `complex128`) without changes. It avoids the parsing of many CSVs during calibration.

CSV for FE frequencies and mode shapes
Mode frequencies from the ADINA solutions are read into `frequencies.csv` (with columns `mode_id, freq_hz`). Mode shapes (nodal displacements) are exported using META to one CSV per mode (`mode_###.csv`) with global displacements at each structural node (`node_id, Dispx, Dispy, Dispz`).

NPZ for the reduced model
The reduced model (`reduced_model.npz`) stores the nominal angular frequencies (ω_0), the diagonal of their squares ($K_0 = \omega_0^2$), and one diagonal gradient vector per parameter (`Gs`). It also stores the modal input/output selectors (`U` and `C`), the equalized damping ratio (`zeta0`), and the list of parameter names. This model is diagonal in modal space and forms the main aspect of a fast analytical FRF reduced model, so NPZ is ideal. The arrays are loaded directly into NumPy without conversion and the file is compact.

JSON for quality control and final parameters
The pairing and quality statistics (mean and minimum MAC values, number of modes matched, and any gates used) are written to a `reduced_model.qc.json` file next to the reduced model NPZ. The calibrated parameters are written to `04_results/best_params.json`. JSON is chosen because it is both human readable and machine readable.

YAML for configuration
The configuration instructions for the calibration is stored in `calib_config.yaml` (plate or propeller in air) and in `calib_config_wet.yaml` (plate or propeller submerged). YAML permits comments, is easy to read, and is less error-prone than long command lines. The configuration file specifies paths for input files, the frequency band of interest, MAC pairing thresholds, the equalized damping to assume, and the parameter definitions (nominal value, lower and upper bounds, and finite-difference step). Using a text based configuration file makes the run transparent and reproducible. A separate `selectors_nodes.yaml` can holds FE node IDs, if the reduced model needs specific DOF selectors.

4 Experimental Setup

This section describes how the measurements were planned, executed, exported, and organized for both structures studied in this thesis: the plate and the three-bladed propeller. Each article was tested in suspended (air) and submerged (water) conditions.

4.1 Test Articles

4.1.1 Rectangular Aluminum Plate (Suspended and Submerged)

The first article is a rectangular aluminum plate with dimensions 280 x 305 mm and thickness t = 4.3 mm. The plate was chosen as a preliminary control case before moving on to the propeller. In the **suspended** condition, the plate was hung by elastic bands to approximate free-free boundary conditions and to isolate the structural dynamics from support stiffness as much as possible [Figure. 4.2]. Choosing the elastic suspension also helps in avoiding spring resonances. In the **submerged** condition, the same plate and suspension were used, but the assembly was immersed in a water tank at a fixed depth of 13 cm from the surface [Figure. 4.3]. Keeping the suspension arrangement same across the two conditions allowed the study to focus on the effects of fluid loading (added mass and radiation damping) instead of changes in boundary conditions. Nominal material values used as nominal in the FE model of the plate were $E = 61$ GPa, $\nu = 0.33$, and $\rho = 2700\ kg/m^3$

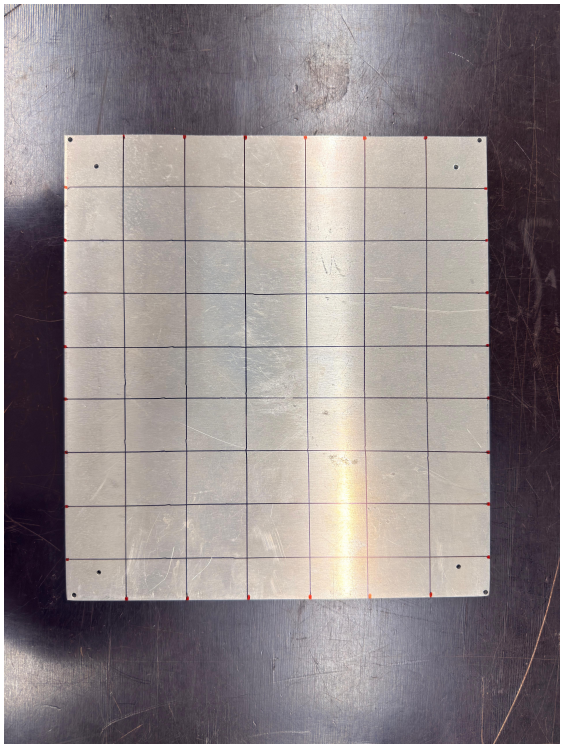


Figure 4.1: Plate Top View

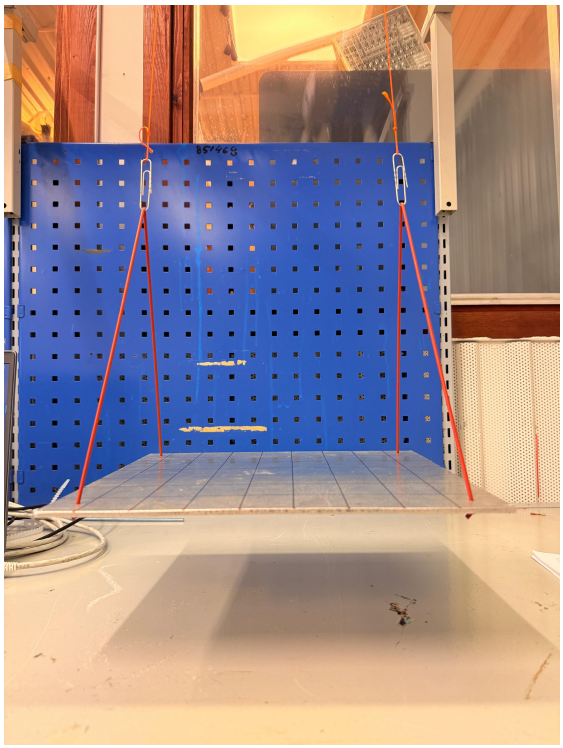


Figure 4.2: Plate suspended with elastic bands to approximate free-free boundary conditions

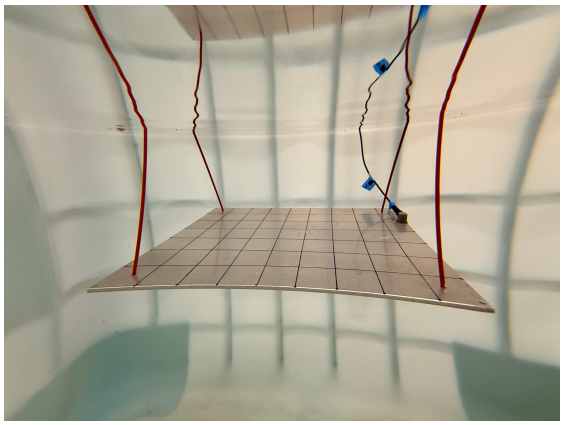
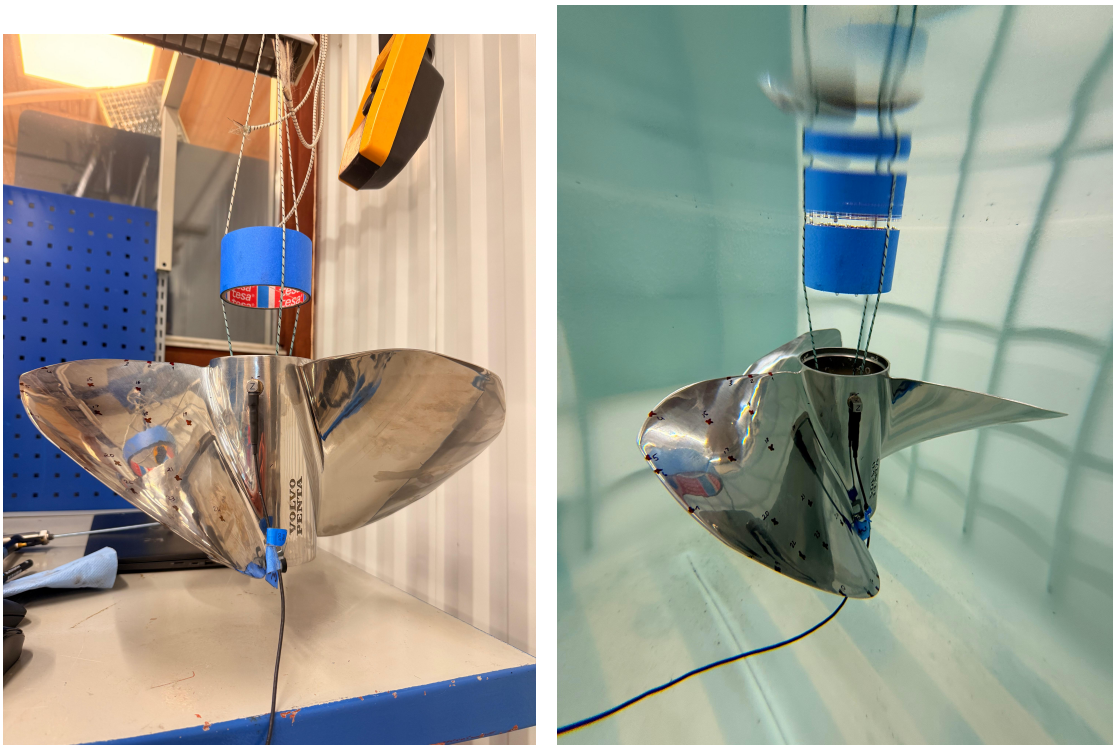


Figure 4.3: Plate submerged at fixed depth

4.1.2 Three-Bladed Propeller (Suspended and Submerged)

The second article is a three-bladed propeller from Volvo Penta. In the **suspended** condition, the entire propeller was hung using elastic bands attached at one end of the hub to approximate free-free conditions [see Figure. 4.4a]. A single tri-axial accelerometer was mounted on the hub. In the **submerged** condition, the same configuration was used, and the assembly was immersed in a water tank at a fixed depth [see Figure. 4.4b]. To limit test time while still exciting the global dynamics of the structure, impacts were applied only on one blade on a grid of 24 points ranging from the root to the tip of the blade. It will be seen in Section 4.4.1 why this was not an ideal setup. Nominal values used in the FE model of the propeller were $E_{prop} = 190$ GPa, $\nu = 0.27$, and $\rho_{prop} = 8000 kg/m^3$



(a) Suspended Propeller **(b)** Propeller submerged at fixed depth

Figure 4.4: Propellers in the suspended and submerged conditions

4.2 Suspension and Boundary Conditions

Both articles were suspended using elastic (bungee-type) bands. This method provides low but finite stiffness and reduces boundary constraints, as compared to rigid mounting/clamping. The arrangement was kept the same between air and water tests to keep changes in support conditions to a minimum. The equivalent vertical stiffness of the suspension is later represented in the FE models by linear springs, and is treated as a calibration parameter when relevant (more so for the plate). For the submerged measurements, the water tank was large enough to avoid strong wall reflections in the frequency band of interest. The free surface was allowed to settle between impacts to limit sloshing. During FSI modeling in ADINA, the open top was modeled as a pressure-release boundary.

4.3 Measurement Instruments

Excitation was provided by a roving modal impact hammer. The hammer was operated with a force-based trigger and a small pre-trigger buffer. This is done so that each recorded time history captured the full impulse. The response was measured by a tri-axial accelerometer [see Table. 4.1]. For the plate, the accelerometer was mounted and used at the same location for all impacts with the +Z axis of the accelerometer aligned with the plate's normal. For the propeller, the accelerometer was mounted on the hub near the measured blade. Sensor mass and mounting were noted so that any mass-loading effect at higher modes could be discussed later.

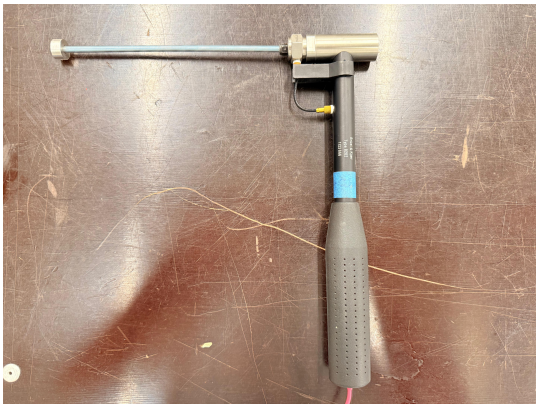
	Instrument	Model ID	Sensitivity
1.	Modal Impact Hammer	Bruel & Kjaer 8202	-0.89 mV/N
2.	Tri-axial Accelerometer	DYTRAN 3273	X: 108.54 mV/g Y: 105.43 mV/g Z: 107.42 mV/g

Table 4.1: Measurement Instruments and their sensitivities

For the submerged case an extension was made for the modal hammer tip to allow underwater excitation (see Fig. 4.5b). This extended tip configuration of the modal hammer was calibrated against a 1 kg suspended mass before measurement procedure. The accelerometer was also prepared for the submerged conditions by including a heat shrink sleeve at the connection to avoid any water damage.



(a) Standard modal impact hammer



(b) Extended hammer for submerged cases

Figure 4.5: Modal impact hammers used

4.4 Measurement Grids

For the plate, the impact grid consisted of 80 points distributed across the surface to excite bending and torsional patterns well. The grid was marked on the plate to ensure consistent hits. The same grid was used for both the suspended and submerged conditions [see Figure. 4.1]. The force direction was -Z (towards the plate). The response channel used in the calibration scripts was +Z acceleration at the fixed sensor.

For the propeller, 24 impact points were defined on a single blade [see Figure. 4.6]. The accelerometer was mounted onto the hub of the propeller at point 25 in Figure 4.6. The initial thinking behind this placement was to avoid mass loading effects from the accelerometer on the blade itself. The impact points ran from the root region to the tip of the blade and were placed along lines that sample both the leading and trailing edge regions. Even though only one blade was impacted, the full propeller geometry was present in the measurement UNV file and in the FE model, which means global, near-degenerate "triplet" modes were present and visible in the spectra as closely spaced peaks. Here, the multiple impact points on a singular blade and a single accelerometer

point on the hub later turned out to be an ineffective measurement grid and approach as discussed in Section 4.4.1.

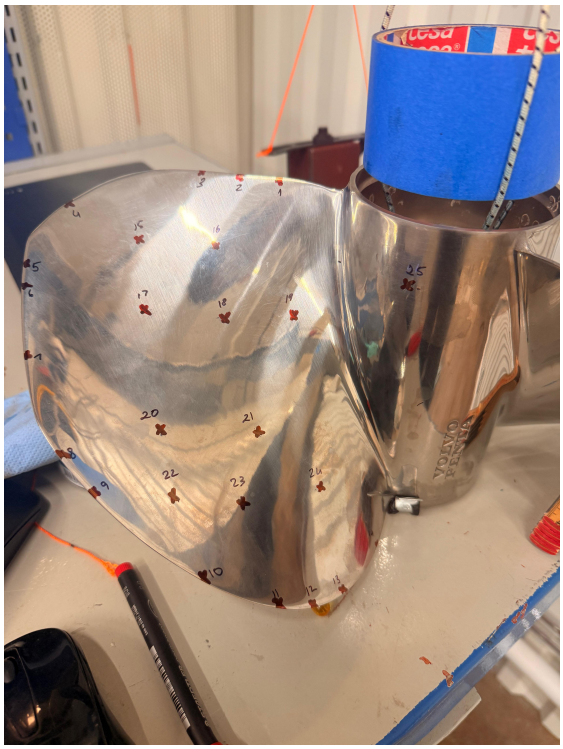


Figure 4.6: Propeller blade excitation points

4.4.1 Limitations of the Propeller Measurement Layout (and Implications for Calibration)

The propeller measurement layout was not ideal for parameter calibration for two main reasons. First, excitations were applied only on a single blade (24 points), while the structure exhibits a cyclical symmetry (see Figure. 4.7) and near-degenerate "triplet" modes. Exciting one blade concentrates the input energy locally and does not excite all the modes of each triplet evenly, which reduces the observability of the global, symmetry based motion across the full propeller. Secondly, the response was measured at a single sensor on the hub. A single response location gives a limited spatial information about blade dominated deformation, so multiple modes that primarily bend or twist the measured blade may have a weak transfer to the hub of the propeller, lowering signal-to-noise ratio in those FRFs, hence effectively reducing the identifiability of material and stiffness parameters.

The current layout can accidentally match peaks that look nearly the same, and make the results sensitive to small misalignments between the FE model and the measurements, and would cause the optimizer to push parameter values to the bounds to try and make up for effects of the uncaptured dynamics. A more calibration friendly layout would have been to have multiple input impacts on all the three blades and include a second response sensor on the measured blade to improve the spatial observability.[Bladh et al., 2001]

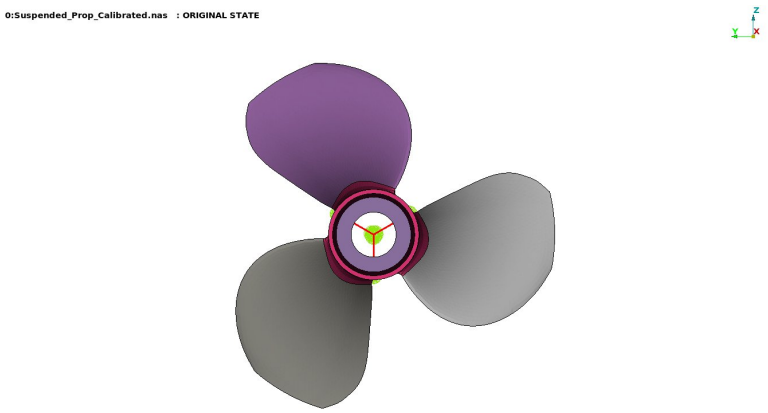


Figure 4.7: Cyclical symmetry of the three-bladed propeller offset by $120°$

4.5 Data Acquisition Settings

All measurements were acquired with the Siemens Simcenter Testlab, with the Seimens SCADAS Mobile as the DAQ device. The sampling frequency, FFT size, and span were chosen so that the usable frequency range covered all modes of interest with acceptable resolution [see Table. 4.2]. For the plate in air and water, modes up to roughly 1 kHz were targeted. For the propeller, the band was chosen to capture the dominant hub/blade global modes and the closely spaced triplets. Force and response windows followed the standard practice (a force window to isolate the impulse and an exponential response window for leakage control). The H1 estimator $H_1(\omega) = G_{xy}/G_{xx}$ was used so that the output noise did not affect the FRFs. Linear averaging over three impacts at each point on the measurement grid reduced the random variability. The coherence $\gamma^2(\omega)$ was checked in real time for each hit. Points with low coherence were re-hit immediately.

	Setting Parameter	**Value**
1.	Bandwidth	4096.00 Hz
2.	Spectral Lines	8192
3.	Resolution	0.50 Hz
4.	Acquisition time	2.00 s

Table 4.2: Scope settings

4.6 Test Procedure

The procedure was the same for all four scenarios. The accelerometer was mounted and wired, and its axes were verified by a quick tap test. The impact grid was marked and photographed for traceability. Before starting the measurement sequence, the DAQ was let to stabilize and the baseline noise levels were checked. At each grid point, three impacts were performed while monitoring the force time trace, the response time trace, and the coherence function. Double hits or false impacts were rejected. A small set of points was re-measured at the end of the session to check repeatability. The channel naming convention used the response channel and axis and the hammer point ID in the UNV block titles so that the automated parsing down the line would be reliable (e.g., Acc:001:+Z/Hammer:010:-Z).

4.7 Data Export and Storage

The UNV file is exported by Testlab for the measurements of each case (for example, `Plate80_Suspended_FRF.unv`, `Plate80_Submerged_FRF.unv`, `Prop_SuspendedFRF.unv`, and `Prop_SubmergedFRF.unv`). The UNV is not modified. A Python script (`00_unv_to_csv.py`) reads the UNV and writes one FRF CSV file per hammer input to the project tree under `01_raw/frf_meas/ (air)` or `01_raw/frf_meas_wet/ (water)`. Each CSV contains the same ascending frequency vector in Hz and the real and imaginary parts of the FRF (accelerance) in SI units. A second Python script (`01_build_sysid_equalize.py`) packs all of these per-input CSVs onto a common frequency grid and writes a single NPZ file pack (`02_proc/HXeq_FRF.npz`) that can then be loaded by NumPy in one line.

5 FE Modeling

This section describes how the finite-element (FE) models were built and solved for the plate and the three-bladed propeller, in both suspended (air) and submerged (water) conditions. It explains the role of each software tool, how the geometry and meshes were prepared, how materials and boundary conditions were applied, h what solver settings were used, and exactly which files were exported for calibration.

5.1 Software Roles in the Modeling Chain

ANSA
ANSA is used strictly for geometry import/cleanup and meshing (no material properties and no loads applied).Additionally, in the submerged cases, the water volume was created in this environment. It outputs a Nastran bulk data file (`.nas`) that contains nodes, solid elements, element sets, and named groups for wetted surfaces. Keeping ANSA limited to meshing makes the flow clear: ANSA = mesh $\rightarrow$ ADINA = physics/solution.

ADINA
ADINA imports the `.nas` meshed geometry file. All materials, boundary conditions, and solver settings are defined here. For submerged cases, ADINA's potential based fluid is coupled to the structure on the wetted interface to form the FSI model. ADINA performs the eigen value extraction and writes a solver ouput file (`.out`) file listing the modal frequencies and an OP2 file containing the node displacement and velocity of the model.

META
META reads ADINA OP2 results and writes one CSV per mode (`modes/mode_###.csv`) with columns `node_id, Dispx, Dispy` in global SI units. META is also used to visualize MAC plots comparing the measurement and FE models.

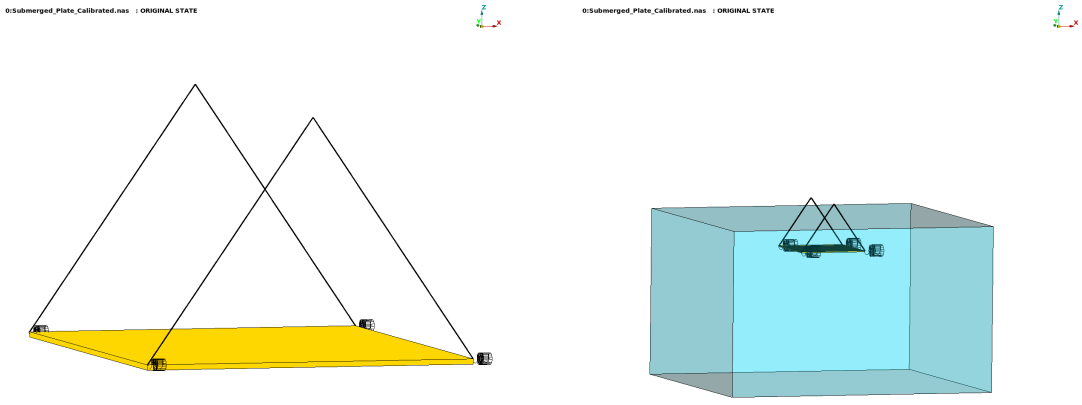
5.2 Geometry and Meshing

5.2.1 Plate

- **Geometry:** The rectangular plate has the dimensions $280 \times 305 \times 4.3$ mm. The water tank is a cuboidal volume with dimensions $1000 \times 1150 \times 700$ mm. The plate is centered horizontally and submerged 130 mm below the surface.
- **Solid modeling:** Both plate and water are modeled as solids. In ADINA, the water block is created and the plate volume is boolean-subtracted from the water to produce the fluid domain with a clean wetted interface.
- **Meshing:**
  - Plate: 3D solid second order tetrahedral elements with global element edge size 4-5 mm and displacement degrees of freedom.
  - Fluid: 3D solid second order tetrahedral mesh, with global element edge length 0.05 m and velocity potential as degree of freedom, and displacements at the fluid structure interaction interface.

Part	Material	Number of elements	Type of element
Plate	Aluminium	173,000	CTetra10
Fluid	Fresh water	277,000	CTetra10
Suspension	Elastic band	4	CBush

Table 5.1: Plate FE model elements information



(a) Suspended plate FE model **(b)** Submerged plate FE model

Figure 5.1: FE models of the suspended and submerged plate

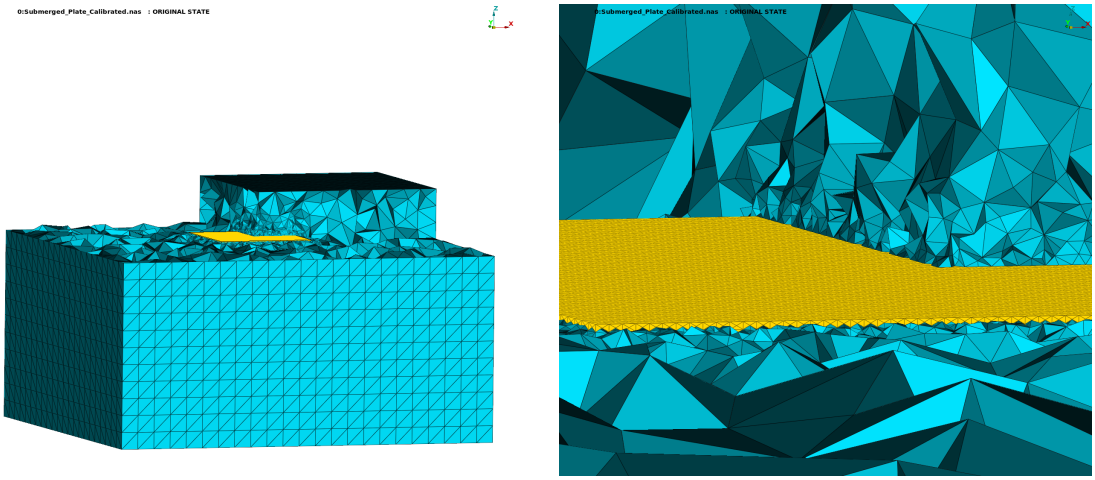


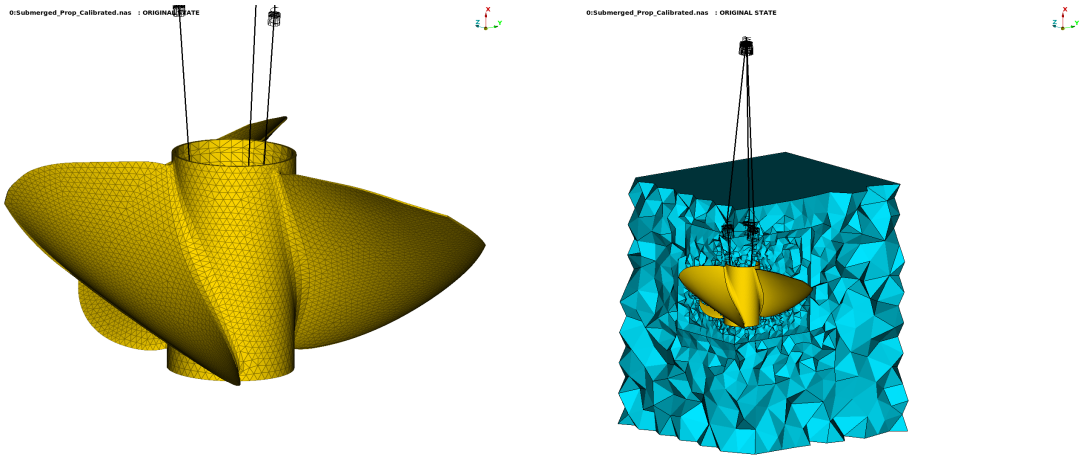
Figure 5.2: Submerged plate in fluid environment

5.2.2 Propeller

- **Geometry:** The three-bladed propeller designed and manufactured by Volvo Penta. Although the measurement grid covered only the 24 points on one blade, the full propeller is modeled so that global triplet modes appear.
- **Meshing:**
  - Propeller: Solid second order tetrahedral elements with displacement degrees of freedom.
  - Fluid: Two volumes of solid second order tetrahedral elements. One near

field cuboidal water volume in contact with the propeller and a far field water volume. The near field fluid mesh is finer than the outer far field fluid box. The fluid elements have velocity potential as degree of freedom, and displacements at the fluid structure interaction interface.

For submerged cases, acoustic wavelength in water is $\lambda = c/f$ (with $c \approx 1500$ m/s). In the bands used here (≤ 1 kHz), a coarse fluid mesh is therefore acceptable away from the interface. Refinement is kept near the structure to stabilize the coupling and better capture near-field gradients. This also helps in bring down compute times later during solving.



(a) Propeller suspended **(b)** Propeller submerged

Figure 5.3: FE models of the suspended and submerged propeller

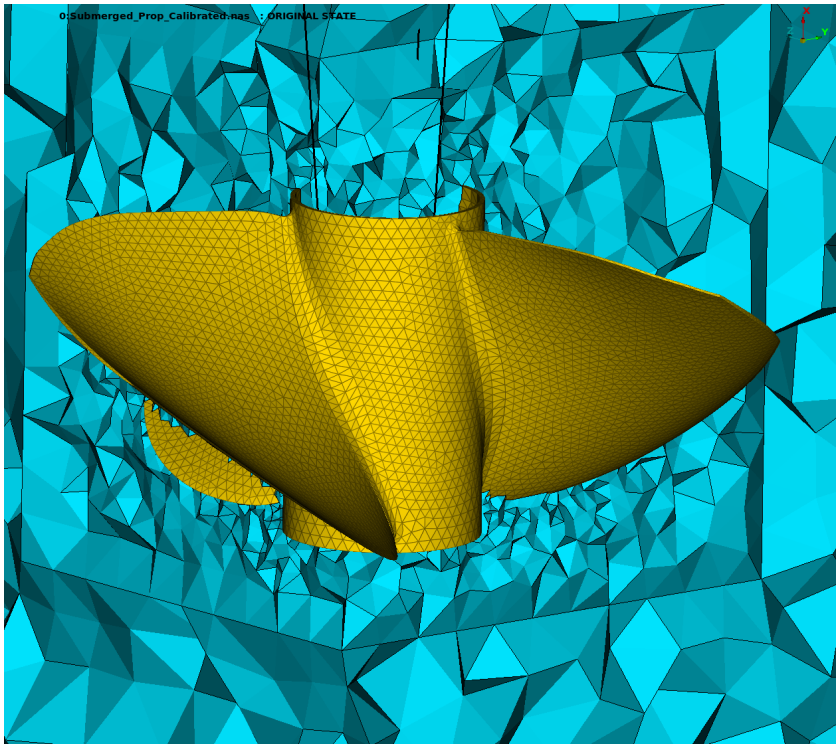


Figure 5.4: Submerged propeller in fluid environment

5.3 Boundary Conditions and Suspension Modeling

5.3.1 Suspended (air)

Elastic bands in the test are represented in the FE model by linear springs connected to the structure at the suspension points. The effective vertical stiffness denoted as k_{susp}, is later included in calibration for the plate as the parameter sensitivity is non-negligible. For both the cases of plate and propeller, suspension points are fixed (fixity) in the model and a mass-proportional load is introduced on the entire setup as gravity.

5.3.2 Submerged (water)

Free surface at the open top of the tank has a pressure release boundary (p = 0). The common surface between structure and fluid is used for FSI coupling in ADINA.In both cases of plate and propeller submerged in water, the suspension points are fixed (as in suspended case), and mass-proportional load is applied on the entire system as gravity (downwards). There are numerous low-frequency sloshing modes when the model incorporates the free fluid surface. These are, however, separable from the required higher frequency flexible modes for the plate and propeller. The frequency solution is therefore solved with minimum frequencies for each case.

5.4 Modal Analysis Settings

This project uses Lanczos iteration in ADINA to compute real, undamped modes for both the purely structural models (suspended cases) and the coupled structure–acoustic models (submerged cases).

ADINA solves the generalized symmetric eigenvalue problem

$$K\phi_k = \omega_k^2 M\phi_k$$

where K and M are the assembled global stiffness and mass matrices of the model in the configuration of interest:

- Suspended (air): K and M come from the structural mesh only, including the elastic suspension representation.
- Submerged (water): K and M are coupled structure–acoustic matrices. The acoustic domain contributes compressibility (stiffness-like) and fluid inertia (mass-like) terms through the wetted interface.

ADINA hence returns eigenpairs $\{\omega_k, \phi_k\}$ (rad/s and corresponding mode shapes) over the requested frequency range or mode count.

5.5 Outputs for Calibration

Each FE case (nominal and +/- delta parameter case) is exported to a folder structure under `03_fe/op2_csv/<case>/..`:

- **Frequencies:** From ADINA's `.out` file the modal frequencies (`mode_id, freq_hz`) are parsed and stored in `frequencies.csv` for each FE case.
- **Mode shapes:** A subfolder `modes/` containing one CSV per mode used in the band is extracted from the OP2 output from ADINA using META. Each mode shape file has columns `node_id, Dispx, Dispy, Dispz` (global coordinates) and represent the mode shapes of the structure.

- **Nominal and +/- delta cases:** The same two items above are produced for the nominal parameters case and the +/- delta parameters case used to compute the finite difference gradients (e.g. E_prop_plus, E_Plate_plus).

6 Calibration Methodology

This section documents how the FE model is calibrated to the measured data for all the articles (plate and propeller, in both suspended and submerged conditions). The intention is not only to the list steps, but to also explain each decision taken: which physical parameters are identified, which signals are trusted, how sensitivities are computed, how some ambiguous mode pairs are handled, and what changes were made after early failures (OP2 parsing, UNV frequency-grid drift, triplet clustering, and sparse sensing at the propeller hub).

6.1 Purpose, Scope, and Design Choices

The calibration process adjusts a small set of physical material parameters so that the FE model's dynamic response matches the experimental measurement data. Modal frequencies are treated as the primary, robust target because they relate cleanly to stiffness and mass. They are also less sensitive to unknown gains and damping. Frequency-response (FRF) magnitudes are used second, mainly to check that the shape of the resonant peaks are consistent. To avoid combining damping and scale with the physical parameters, the model is evaluated with a fixed/"equalized" modal damping ratio ζ_0 and a single global gain G when comparing FRF magnitudes. This follows the approach Abrahamsson proposed: damping is equalized so that the calibration emphasizes stiffness- and mass-controlled quantities, while one gain absorbs measurement-chain scaling [Abrahamsson and Kammer, 2015].

6.2 Data Inputs and Outputs

On the measurement side, the pipeline produces one CSV per input (impact) with a strictly ascending frequency vector and complex accelerance data. These are packed into a single NPZ file that contains a frequency grid and the 3-D array of FRFs. On the FE side, each analysis case (nominal and each +/- delta parameter case) provides a `frequencies.csv` with `mode_id, freq_hz` and a set of mode-shape CSVs with one file per mode, listing `node_id, Dispx, Dispy, Dispz` in global SI units. A YAML file is used to as a helper file for the two sides(measurement and FE) by specifying paths, the calibration frequency band, the equalized damping value, pairing gates for MAC and frequency proximity, and the parameter block with nominal values, upper and lower bound values of the parameter, and finite-difference steps. The calibration scripts write a reduced model NPZ and a quality control JSON file(for pairing quality).

6.3 Measurement Damping Equalization

The equalization step does not numerically modify the FRFs. Instead, it standardizes them into a single, consistent pack and defines how they will be compared. All input FRFs are checked for strictly ascending frequency axes; tiny numeric differences from export are normalized by sorting and snapping the frequency grid to eight decimals. Files that still disagree beyond a very small absolute tolerance are either rejected or interpolated explicitly so that all inputs share an identical grid. The "equalization" then occurs at comparison time: the model is always evaluated with the fixed damping ζ_0. When FRF magnitudes are compared, a single global gain is fitted so that the optimizer

is not likely to misuse the physical parameters to compensate for amplitude scaling.

6.4 Reduced Model From FE

To make calibration fast and transparent, the FE model is replaced by a diagonal "modal surrogate". The nominal modal line is defined by

$$\omega_{0k} = 2\pi f_{0k}$$

and

$$K_0 = diag(\omega_{0k}^2)$$

For each parameter p_i, a diagonal sensitivity G_i is computed per mode from the delta runs. When forward finite differences are adequate,

$$g_{ik} = (\omega_{k,+}^2 - \omega_{0k}^2)/\Delta p_i$$

When the spectra are clustered (propeller triplets or close FSI-coupled modes), central differences are preferred.

$$g_{ik} = (\omega_{k,+}^2 - \omega_{k,-}^2)/2\Delta p_i$$

The reduced stiffness along the modal axes is then approximated by

$$K(p) \approx K_0 + \sum_i (p_i - p_{0,i})G_i \tag{6.1}$$

This reduced model works because it assumes each mode of vibration keeps its unique character even after small adjustments. We confirm this by matching the original and altered mode shapes with a strict comparison tool (MAC), to make sure they don't deviate too much.

6.5 Pairing With MAC and Gates

Every +/- Delta parameter set of modes are paired to the nominal set by maximizing modal assurance criterion(MAC) and solving a Hungarian assignment on the cost $1 - MAC$. Two safeguards prevent obvious wrong matches: a minimum MAC threshold (tightened or relaxed per article) and, where helpful, a frequency proximity gate that limits how far a mode may lie from the nominal mode value. If a delta case contains fewer modes in the calibration band than the nominal case, as happened in some submerged runs, the method records the matched subset and builds the sensitivity only on the modes that could be paired. The pairing step produces a small QC(quality control) file that reports the number of matched modes, the mean and minimum MAC, and any warnings. This QC file is checked before proceeding to the next steps.

6.6 FRF Comparison with Equalized Damping and One Global Gain

This project compares the FE model to the measurements in the frequency domain using frequency response functions (FRFs). The FRF used here is accelerance (acceleration per unit force). Instead of trying to estimate many hard-to-know quantities (exact damping in each mode, exact sensor/hammer gains for every input), the comparison is set up so that those effects do not drive the calibration. Two simple ideas make this work:

First, the damping is fixed in the model:
Every mode in the reduced model is given the same small, constant damping ratio ζ_0 (the “equalized” damping). That keeps all resonant peaks narrow in a controlled way and avoids mixing “damping fitting” with the parameters we actually care about (like elastic modulus or density).

Second, using just one overall gain for amplitudes:
Because the setup has one response channel (one accelerometer) and many hammer inputs, it is not required to estimate a separate input gain for each impact. Instead, a single scale factor G is used to line up the overall level of the model FRFs with the measured FRFs. At every evaluation of the residual, that single G is found in closed form (least squares in the log-magnitude scale), so the optimizer does not waste iterations trying to get a trivial scale mismatch.

With those choices, the model FRF is just a sum of modal terms. In the reduced mode, each mode acts like a simple second-order resonator. Its "effective stiffness" on the modal axis is

$$K_{kk} \approx K_{0,kk} + \sum_i (p_i - p_{0,i}) G_{i,kk} \tag{6.2}$$

where $K_{0,kk}$ is the nominal value and $G_{i,kk}$ is the sensitivity from the finite difference step for parameter p_i.
Using the equalized damping ζ_0, the modal accelerance FRF at angular frequency ω becomes [Abrahamsson and Kammer, 2015]

$$H_{FE}(\omega) = -\omega^2 \sum_{k=1}^{n_m} \frac{C_k U_k}{K_{kk}(p) - \omega^2 + 2j\zeta_0 \omega_{0k} \omega} \tag{6.3}$$

In our setup (one response, many inputs), the direction factors C_k and U_k are not identified separately. They are combined into one global gain G.

The comparison to data is done on log-magnitude (dB). For each frequency bin, the code computes $log_{10}|H_{meas}|$ and $log_{10}|GH_{FE}|$, takes their difference, and stacks these values across frequency and inputs to form the residual vector for the optimizer.

This design has three practical benefits:

- It keeps the fit focused on stiffness and mass (what the parameters control), not on damping or unknown amplitude scales.
- It is stable with sparse sensing (single output) and many inputs, because the global gain G handles scale differences.
- It works cleanly with the diagonal reduced model, since only per-mode terms are needed and the ζ_0 term is simple and fixed.

6.7 What is Minimized and in What Order

The calibration is run in two steps so the residual minimization problem stays stable. Stage A works on frequency-only. It pairs measured peak frequencies with the model’s modal frequencies from the reduced model and minimizes the squared differences inside the chosen band. This step is the main one, because modal frequencies respond cleanly to changes in stiffness and mass and are not confused by unknown damping

or gains. Peaks are picked once from the measured FRFs (using standard prominence and spacing rules), and the same list is reused so that the optimizer does not have to chase different peaks each time. When there are clusters (for example, triplets on the propeller), the pairing uses MAC frequency gates and if needed, applies small weights so that the most reliable mode of a cluster drives the fit.

Stage B is optional and checks FRF log-magnitude. After Stage A aligns the peak locations, Stage B compares $log_{10}|H_{meas}|$ with $log_{10}|GH_{FE}|$ over the same band, where the global gain G is removed in closed form at each call. Stage B is useful for catching undesired level mismatches, but it is not used to pull parameters if the sensing is sparse or the peaks are ambiguous. In this project Stage B is a qualitative check for the propeller and a helpful confirmation for the plate.

6.8 Choosing Parameters

Each parameter in the configuration YAML has a nominal value, bounds, and a finite-difference (FD) step. Bounds are chosen so that the values remain physically reasonable. The FD step is chosen so the change in modal frequencies is small but visible (about 0.5–2%). If the step is too small, the sensitivity is noisy. If it is too large, the model leaves the local linear region and the MAC pairing may swap modes, especially in clusters. For isolated modes, forward differences are enough. For clustered spectra (e.g., propeller triplets), central differences are safer and more accurate.

6.9 Optimizer Logic

The optimizer is a bounded nonlinear least-squares method. The code adjusts a few model parameters so that the model agrees with the measurements, while never letting any parameter go outside the physical bounds set in the YAML. The optimizer evaluates residual, measures how large that residual is (the cost), and then proposes new parameters that should make the cost smaller.

The optimizer stays within the bounds from the YAML on every step. If a parameter tries to move beyond its allowed range, the bound stops it. Bound hits indicate that if a parameter keeps pushing to a bound while the cost does not improve, that parameter is probably not observable with the current data or band. In that case it is better to fix that parameter and focus on the ones that actually change the residual.

Two numbers help decide whether the run is healthy. The first is the cost. It should go down as the optimizer iterates. A steady decrease means the residual is informative and the parameters are doing useful work. The second is the per-parameter gradient norm. A large gradient means that parameter still has affects the residual. A very small gradient means the data cannot "see" it in the current setup.

If the cost barely changes or the optimizer stops after very few evaluations, that usually points to one of three issues. The band may be too wide or include messy clusters. Narrowing the band to clean peaks makes the problem easier. The pairing between model and measurement may be weak. Checking the MAC QC and tightening MAC or frequency gates prevents wrong matches. The finite-difference steps used to build sensitivities may be poorly chosen. Steps that are too small create noisy gradients, and steps that are too large can flip pairings. Near clustered spectra (such as triplets on a propeller), using central differences for elastic-type parameters often makes the gradients more reliable.

When the optimizer runs with FRF log-magnitude residuals, a single global gain is solved in closed form at each evaluation so that unknown sensor or hammer scaling does not bias the physical parameters. This keeps the comparison focused on stiffness and mass effects (peak locations and overall shape), rather than on absolute level.

6.10 Quality Control Files

Two small quality control files make checks quick. The pairing QC JSON (from Step 02) shows for each parameter, how many modes were matched and the MAC mean/min under the current gates. If matched counts are low or MAC minimum is poor, the sensitivities are not reliable and the finite difference(FD) step or gates should be adjusted. The calibration summary (from Step 03) shows nominal and best values, cost, nfev, gradient norms, and FRF RMSE dB if Stage B was used. If a parameter is at a bound or has an near-zero gradient, it is likely that it is not observable with the current setup and should be fixed or removed next time.

6.11 Problems Met

Parsing ADINA OP2 failed on some runs, so META CSV exports were adopted for mode shapes and a dedicated `.out` frequency parser was written. UNV FRFs sometimes had slightly different frequency grids. Strict sorting and snapping solved most cases, and a controlled interpolation resolved the rest. The propeller's triplets and sparse hub sensing made pairing unstable until MAC gates, partial matching, and central differences were added. We also learned not to lean on FRF magnitude fitting with one output.

7 Results

This chapter reports what was obtained from the four calibration cases, using the pipeline described earlier. Each case is written in terms of what data went in (bands and signals), which parameters were identified (with bounds and finite-difference steps), what matched in the MAC-based pairing, and what changed after calibration.

7.1 Suspended Plate (Air)

7.1.1 Inputs

The suspended plate was measured with a roving hammer at 80 impact points and one accelerometer at a fixed location. The Z-axis accelerance FRFs (acceleration/force) were exported to CSVs and consolidated into a single equalized NPZ with a common frequency grid. The calibration band covered the modes in the desired frequency band of interest. The FE side provided the nominal modal set and one Δ set per parameter for finite differences.

7.1.2 Parameter, Bounds and Steps

The plate calibration targeted structural parameters plus the suspension stiffness. Poisson's ratio was treated as fixed. Thickness was allowed but, in practice, it did not move.

Parameter	Name (units)	Nominal	Min	Max	step_fd (Δ)	FD type
E_{plate}	Elastic modulus (Pa)	6.10e10	5.0e10	7.5e10	+1.2e9 (2%)	forward
ρ_{plate}	Density (kg/m^3)	2700	2600	2800	+27(1%)	forward
t_{plate}	Thickness (m)	0.0043	0.0041	0.0045	+5e-5(1.2%)	forward
k_{susp}	Suspension stiffness (N/m)	35.5	10	200	+5	forward

Table 7.1: Parameter nominal values, bounds, finite difference steps, and FD type

Parameter	Name (units)	Best (after)
E_{plate}	Elastic modulus (Pa)	5.996e10
ρ_{plate}	Density (kg/m$\hat{3}$)	2712.6
t_{plate}	Thickness (m)	0.0043
k_{susp}	Suspension stiffness (N/m)	91.47

Table 7.2: Parameter calibrated values

7.1.3 Pairing Quality (MAC)

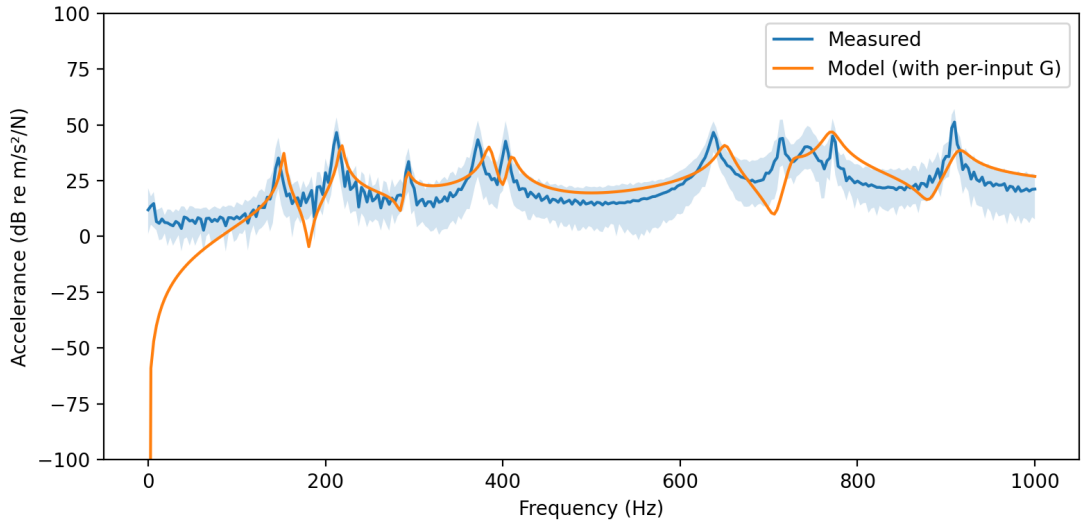
Mode pairing between nominal and perturbed sets was stable for the plate case. The MAC values used to build sensitivities were high for almost all modes. Representative values from the measurement-to-model MAC check after calibration were mostly between 0.82 and 0.96, with two outliers near 0.73.
After calibration, the per-mode frequency errors fell below about 3% across the modes of interest. The biggest movement came from k_{susp}, which set the low modes correctly. The elastic modulus and density moved only slightly and stayed within their bounds.

Thickness did not change. FRF overlays (with one global gain) showed that the resonant peaks aligned well in frequency and had sensible levels.

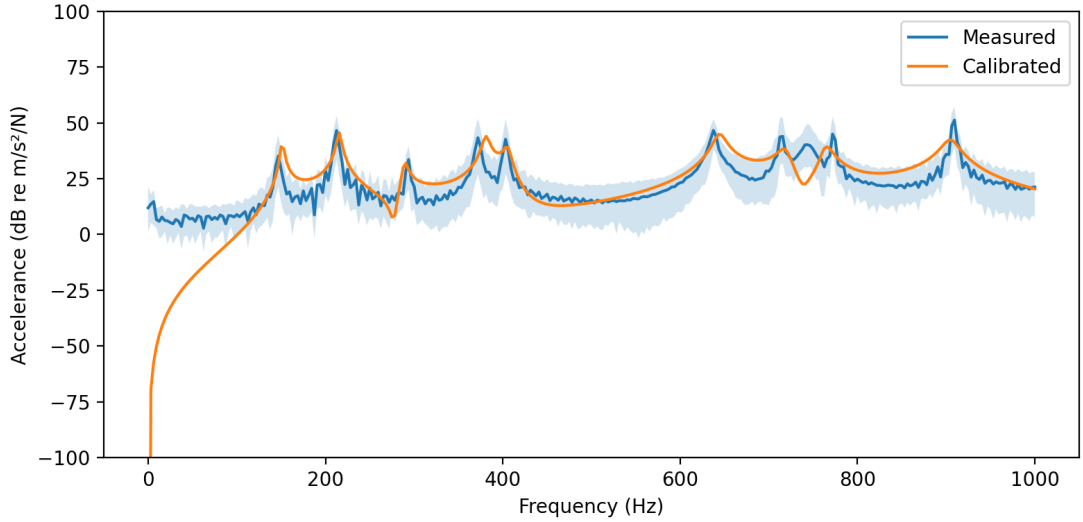
Figures 7.1a and 7.1b show the FRF overlays between the measured and FE nominal modes and FE calibrated modes respectively. As seen in Figure 7.1a, the peaks for the suspended plate agree quite well even before the calibration process, which can be attributed to the simplicity of the case. But the peaks between the modeled and measured FRF are seen to be drifting apart at higher frequencies. After calibration of the parameters(according to Table 7.2), a clear shift in the peaks of the FE models towards the measured peaks can be observed(as seen in Figure. 7.1b). The mode at around 740 Hz seem to have a phase shift between the measured and FE models. This effect is also seen in the swapped mode positions seen in the MAC heatmap plots of the measured vs FE model(Figures. 7.2 and 7.3). In the MAC plots, we see a clear improvement between the nominal and calibrated figures and it is seen that the even the swapped positions are better correlated after calibration. Table 7.3 shows the percentage frequency difference between the measured modes and the calibrated FE modes. The frequency differences between the paired modes can be seen to be within a range of -3% to 1.4% and the MAC correlation between the shapes range from 0.82 to 0.99. This can be classified as a very good result in terms of the calibration effort.

FE modes	Measured mode	MAC	Freq diff%	FE mode freq	Measured mode freq
1	1	0.988196	-3.04616	151.24	146.633
2	2	0.896504	-1.45084	215.53	212.403
3	3	0.977199	1.43592	289.051	293.262
4	4	0.980036	-2.35317	380.848	371.886
5	5	0.897743	-0.580866	405.599	403.243
6	6	0.918325	-0.963082	644.389	638.183
7	7	0.880871	-0.632653	718.719	714.172
8	8	0.833024	1.36929	761.725	772.3
9	9	0.820313	-2.49486	763.248	744.206
10	10	0.945664	0.416695	904.78	908.566

Table 7.3: Calibrated modes vs measured modes MAC and frequency difference values for the suspended plate



(a) FRF overlay of the measured and nominal values



(b) FRF overlay of the measured and calibrated values

Figure 7.1: FRF overlays for the suspended plate scenario

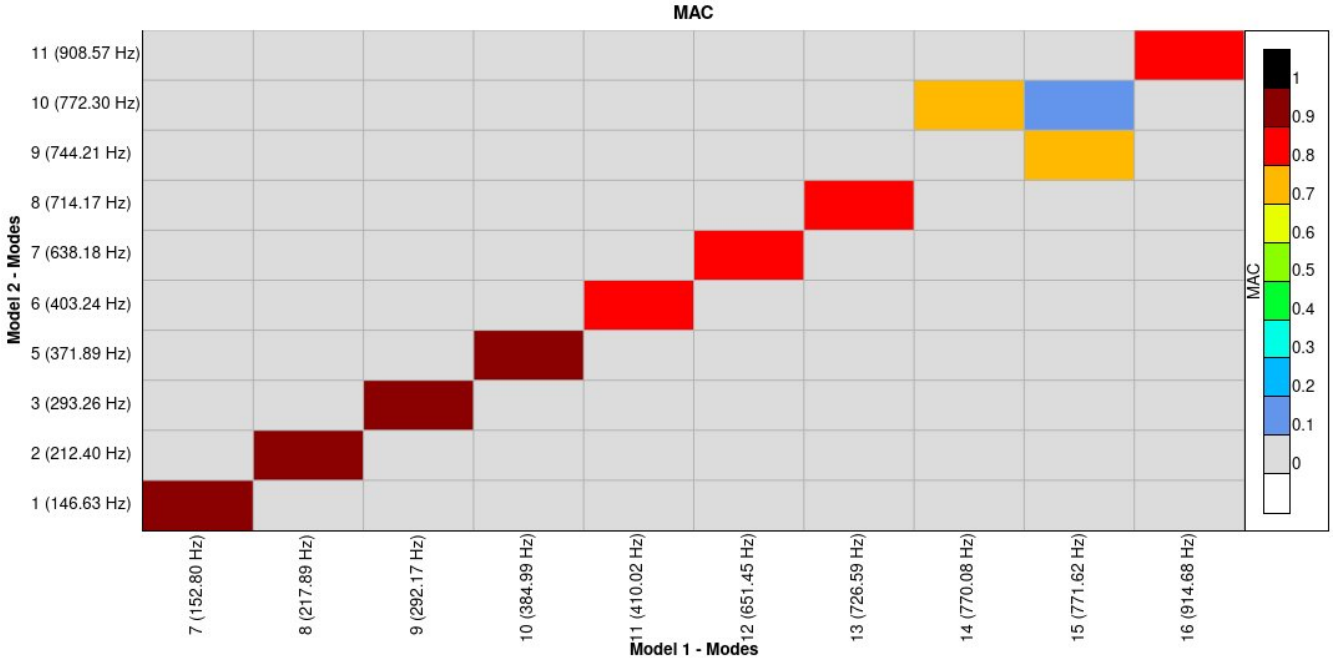


Figure 7.2: Suspended plate MAC plot before calibration (x-axis: FE modes, y-axis: measured modes)

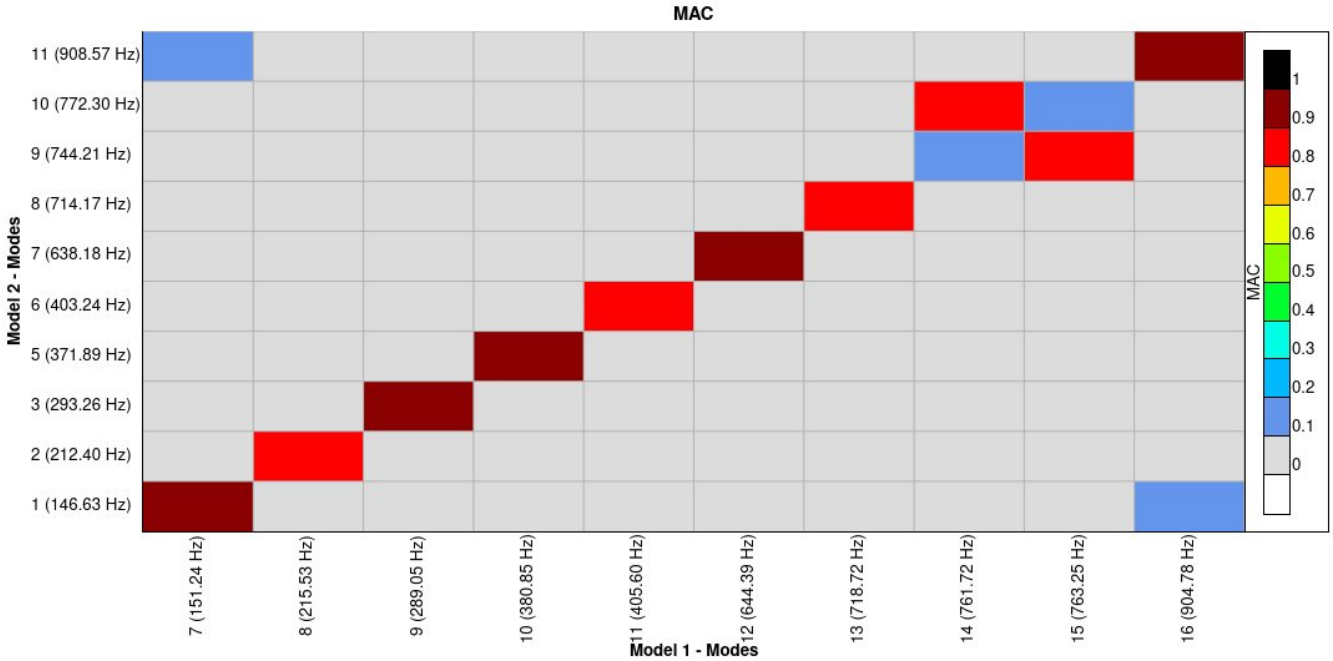


Figure 7.3: Suspended plate MAC plot after calibration (x-axis: FE modes, y-axis: measured modes)

7.2 Submerged Plate (Water)

7.2.1 Inputs

The same plate was measured while fully submerged. The roving hammer layout and single accelerometer were unchanged. The FRFs were parsed to CSV and consolidated

as before. The band was limited to the range where peaks were clear and pairing remained stable under added-mass effects. Structural parameters values were set to the calibrated values of the suspended-case. Only the fluid parameters were used for the calibration process. This was done to clearly observe the effects of mass-loading and radiation damping caused by the fluid.

7.2.2 Parameter, Bounds and Steps

The fluid was modeled as potential-bsed fluid with density and bulk modulus. The nominal sound speed was set using $\sqrt{K_f/\rho_f}$.

Parameter	Name (units)	Nominal	Min	Max	step_fd (Δ)	FD type
c_f	Sound speed (m/s)	1500	1400	1600	+75 (5%)	forward
ρ_f	Density (kg/m$\hat{3}$)	1000	900	1100	+50(5%)	forward
k_{susp}	Suspension stiffness (N/m)	35.5	10	200	+5	forward

Table 7.4: Parameter nominal values, bounds, finite difference steps, and FD type

Parameter	Name (units)	Best (after)
c_f	Sound speed (m/s)	1502.06
ρ_f	Density (kg/m$\hat{3}$)	1004.68
k_{susp}	Suspension stiffness (N/m)	91.46

Table 7.5: Parameter calibrated values

7.2.3 Pairing Quality (MAC)

The MAC pairing was very good between the sound-speed delta set(MAC mean $\approx$ 0.99, MAC min high) and nominal modes. Pairing for the density delta set was weaker in places because shifting many modes together squeezes the band and can hide peaks in comparison with nominal modes. The QC file reflected lower matched counts in some runs. These points were taken into account when interpreting gradients.

The fitted ρ_f moved by roughly 5% relative to nominal, and c moved within a narrow range consistent with the selected band. The calibrated model reproduced the submerged peaks in frequency reasonably well, and the FRF overlays were acceptable once the single gain was applied. Where pairing was weak for ρ_f, those modes were downweighted or excluded from the frequency residual to avoid biasing the result.

Figures 7.4 and 7.5 show the MAC heatmap for the mode shape comparisons between the measured and FE nominal modes and calibrated modes respectively. It is evident from Figure 7.4, that with the calibrated structural parameter values, the corresponding mode shapes line up between the measured and nominal FE instances, but there are clearly effects from the fluid mass loading. This is seen to be causing a lower MAC value between 0.2 and 0.4. After the calibration, as seen in Figure 7.5, the MAC correlation between the measured and calibrated FE modes are much higher. The MAC values of the calibrated modes are all above 0.91. This is a significant improvement over the uncalibrated model. This indicates the significance of the fluid properties' effect on a submerged structure, and how it affects the mode shapes of a structure. In table 7.6, we

can see that the percentage frequency difference between the measured and calibrated FE modes ranges between -4.8% to -0.74%. The MAC values range from 0.91 to 0.99. This is an indication that the mode shapes are almost perfectly correlated between the measured and calibrated FE modes.

FE modes	Measured mode	MAC	Freq diff%	FE mode freq	Measured mode freq
1	1	0.983351	-4.88882	63.3916	60.2925
2	2	0.979696	-2.97389	91.6881	88.9614
3	3	0.985281	-3.57996	172.879	166.69
4	4	0.991545	-2.58315	186.439	181.623
5	5	0.984482	-2.31913	288.989	282.287
6	6	0.981788	-2.77771	343.449	333.909
7	7	0.967178	-0.744212	351.779	349.161
8	8	0.970467	-3.22586	374.473	362.393
9	9	0.975861	-1.40087	433.23	427.161
10	10	0.965037	-2.68183	585.831	570.12
11	11	0.962363	-3.02485	610.376	591.913
12	12	0.914137	-1.3838	748.949	738.585
13	13	0.928103	-2.19201	834.486	816.194

Table 7.6: Calibrated modes vs measured modes MAC and frequency difference values for the submerged plate

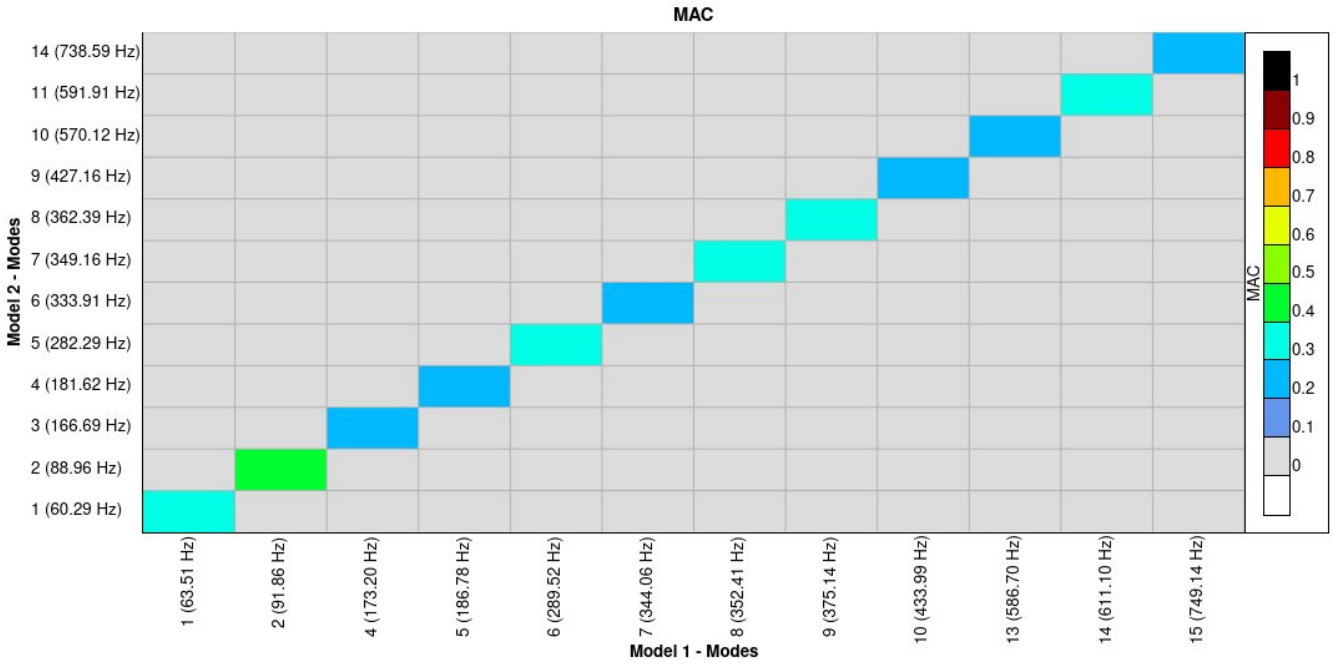


Figure 7.4: Submerged plate MAC plot before calibration (x-axis: FE modes, y-axis: measured modes)

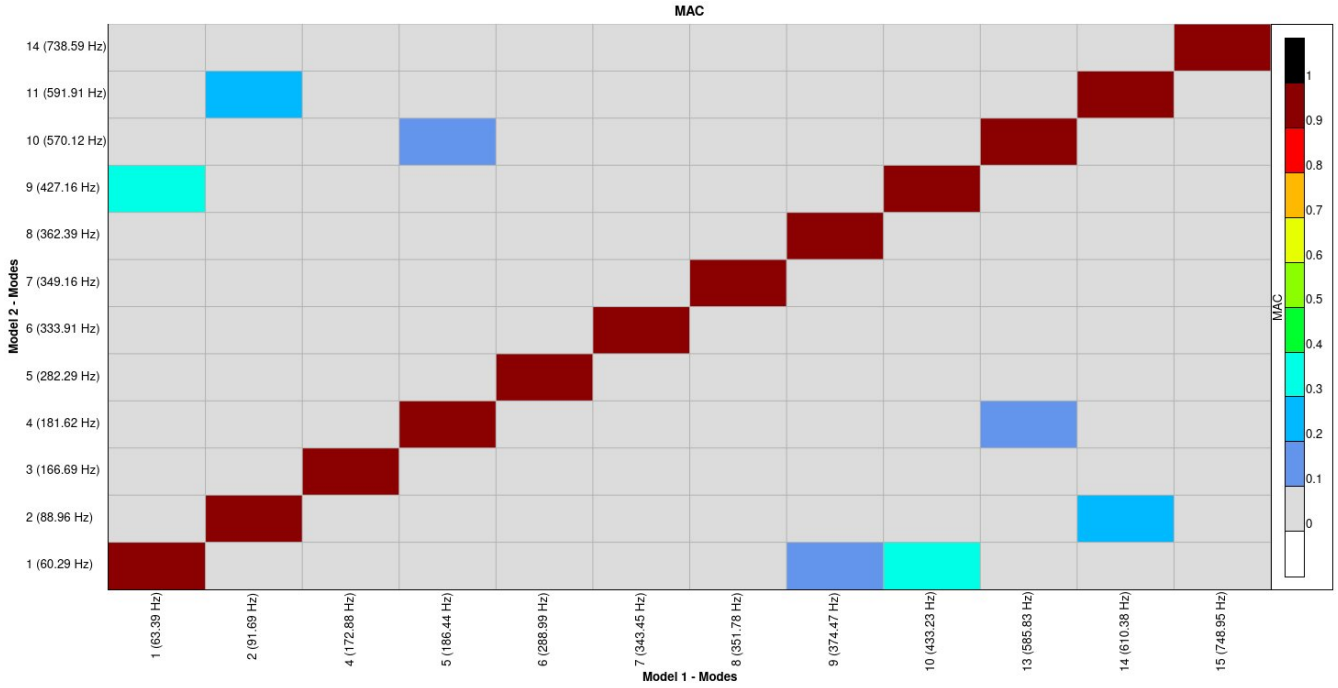


Figure 7.5: Submerged Plate MAC plot after calibration(x-axis: FE modes, y-axis: measured modes)

7.3 Suspended Propeller

7.3.1 Inputs

The propeller was measured with impacts on a single blade and one accelerometer near the hub. This layout is sparse. It favors modes that are well "seen" at the hub and can hide other members of the triplet clusters. The FRFs were parsed for the chosen axis (+Z or +X at the hub) and packed as before. The band was set to avoid the lowest rigid-body motion and the highest crowded region. The reduced model was built with central differences for the elastic modulus of the propeller in an attempt to stabilize triplets modes.

7.3.2 Parameters, Bounds and Steps

The material of the propeller was treated as uniform. The sleeve/bushing parameters were tested but dropped later because their gradients were repeatedly near zero with the present sensor layout. The suspension stiffness represents the combined elasticity of the supports used in test.

Parameter	Name (units)	Nominal	Min	Max	step_fd (Δ)	FD type
E_{prop}	Elastic modulus (Pa)	1.9e11	1.6e11	2.2e11	$\pm$5.7e9 ($\pm$3%)	central
ρ_{prop}	Density (kg/m$\hat{3}$)	8000	7400	8600	+240(3%)	forward
k_{susp}	Suspension stiffness (N/m)	1000	200	5000	+100 (10%)	forward

Table 7.7: Parameter nominal values, bounds, finite difference steps, and FD type

Parameter	Name (units)	Best (after)
E_{prop}	Elastic modulus (Pa)	2.192e11
ρ_f	Density (kg/m$\hat{3}$)	8200.6
k_{susp}	Suspension stiffness (N/m)	1000

Table 7.8: Parameter calibrated values

7.3.3 Pairing Quality (MAC)

Triplet clusters were present as expected from the threefold cyclical symmetry. With one hub channel and impacts on just one blade, the strongest member of each triplet was clear, but the other two were sometimes weak. The pairing therefore used MAC gates and allowed partial matching. For sensitivity building, only paired modes with acceptable MAC were kept; central differences were used for E_{prop} to reduce bias near clustered peaks.

Frequency-only calibration aligned the dominant peaks reasonably. However, with the current measurement layout, density often carried most of the leverage, while the elastic modulus had a weaker gradient. In early FRF-based checks, the log-magnitude residual could remain large even when the frequency residual improved, which is consistent with the sparse sensing at the hub. The final judgement relied on the frequency residual and on the pairing QC. But there isn't as much of an improvement after calibration as hoped, mainly due to a poor measurement layout.

Figures 7.6a and 7.6b show the FRF overlays between the measured and FE nominal and calibrated values respectively. In these figures, the double peaks consistent with one axisymmetric family mode and a doublet mode pair at the same frequency from the whirling family are clearly visible. It is important to note that the placement of these measured peaks in the frequency domain are unreliable due to the measurement grid and causes the FE model's peaks to mismatch despite calibration efforts. It is evident from these figures that the optimization process is trying to match unreal peaks and has ended up hitting the bounds for material parameters in trying to do so as seen in Table 7.11. Figures 7.7 and 7.8 show the MAC heatmap between the measured modes and FE nominal and calibrated modes respectively. The triplet modes are clearly seen in these figures. We see two mode shapes at almost the same frequency along with one mode at slightly different frequency but still matching the same measured mode shape. These cause the triplet blocks seen in the MAC heatmaps. A small improvement can be observed between the MAC plots of the nominal and calibrated modes but this is still an ineffective calibration in terms of the false measurement data that the optimizer is chasing.

FE modes	Measured mode	MAC	Freq diff%	FE mode freq	Measured mode freq
1	1	0.623617	-2.49734	502.72	495.606
2	2	0.611536	15.4414	502.74	601.149
3	1	0.609636	-15.2925	578.52	495.606
3	2	0.86666	2.67322	578.52	601.149
4	3	0.733091	-0.0788464	1078.92	1089.85
4	4	0.751443	2.84679	1078.95	1122.67
6	3	0.697347	-13.792	1249.90	1089.85
6	4	0.709506	-2.70974	1472.71	1448.36
8	6	0.710967	-0.800693	1472.75	1476.78
9	6	0.782945	-0.830008	1473.15	1476.78
10	7	0.624191	-2.32942	1669.10	1647.82
10	8	0.751774	-0.462915	1686.40	1679.31
11	8	0.693158	-1.49114	1786.43	1679.31

Table 7.9: Calibrated modes vs measured modes MAC and frequency difference values for the suspended propeller

In table 7.9 shows the MAC values ranging from 0.6 to 0.87, with an average of 0.72 after calibration. The percentage frequency difference ranges between -15% to 15%, and these large frequency differences are explicitly seen for the FE modes that are MAC paired to a higher measured mode, indicating that there exist similar mode shapes which failed to be captured due to the measurement layout used.

7.4 Submerged Propeller

7.4.1 Inputs

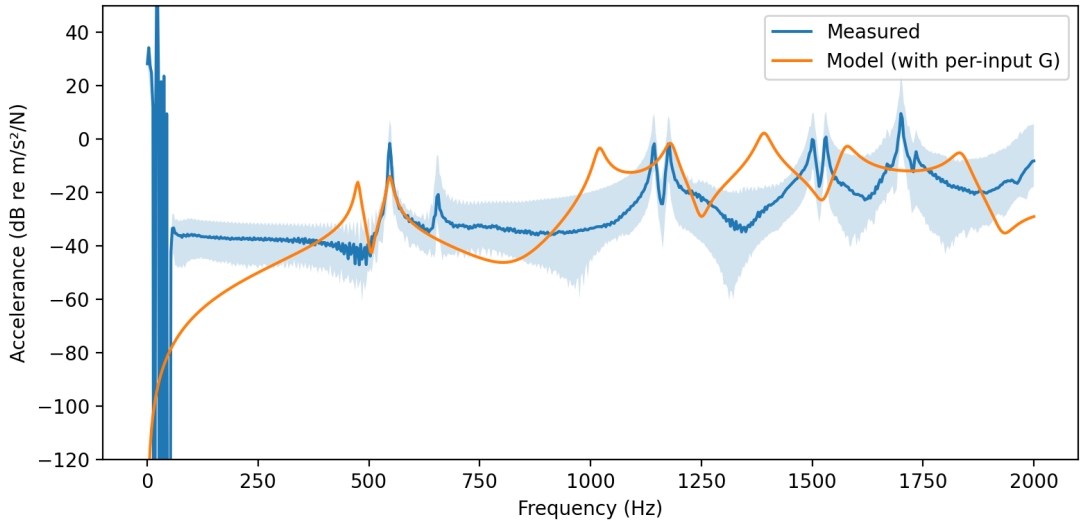
The same propeller was measured while submerged. The impact layout and the single hub accelerometer remained the same, which makes the problem harder: the added mass from the fluid shifts many modes together, and triplets remain. The FRFs were parsed and consolidated as before, and the band was chosen where peaks were still separable and pairing remained stable. The reduced model included fluid coupling through the FE solver.

7.4.2 Parameters, Bounds and Steps

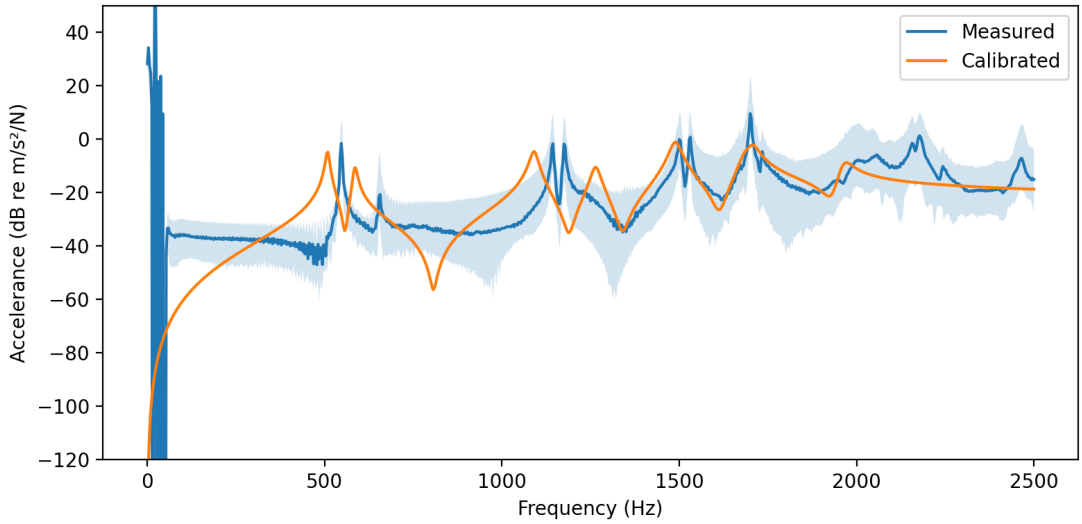
In the submerged case, the structural parameters were maintained at the calibrated suspended-case values. Only fluid parameters were used as parameters for calibration[Table 7.10].

Parameter	Name (units)	Nominal	Min	Max	step_fd ($\triangle$)	FD type
c_f	Sound speed (m/s)	1500	1400	1600	+75 (5%)	forward
ρ_f	Density (kg/m$\hat{3}$)	1000	900	1050	+50(5%)	forward

Table 7.10: Parameter nominal values, bounds, finite difference steps, and FD type



(a) FRF overlay of the measured and nominal values



(b) FRF overlay of the measured and calibrated values

Figure 7.6: FRF overlays for the suspended propeller scenario

Parameter	Name (units)	Best (after)
c_f	Sound speed (m/s)	1500
ρ_f	Density (kg/m$\hat{3}$)	1050

Table 7.11: Parameter calibrated values

7.4.3 Pairing Quality (MAC)

Since the fluid shifts many modes together, matched counts can drop if the band is too wide. The QC JSON should be checked after each reduced-model build. If MAC minima fall or matched counts shrink, the band should be narrowed or the pairing gates tightened. Where triplets appear, the strongest member should be kept for the frequency residual, and the others left for qualitative checks.

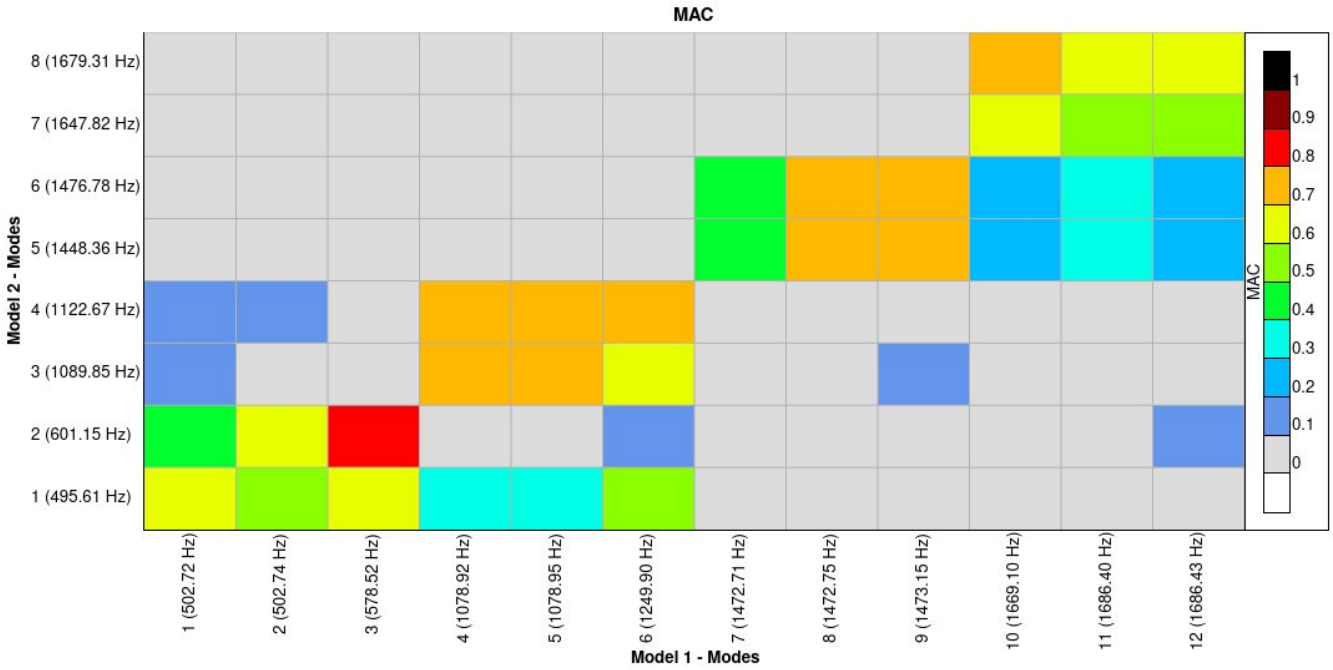


Figure 7.7: Suspended propeller MAC plot before calibration (x-axis: FE modes, y-axis: measured modes)

After calibration, the fluid parameter are as listed in table 7.11. It is clearly evident that, like the suspended propeller case, the optimizer is hitting bounds to compensate for the poor measurement data. From the MAC heatmaps(Figures 7.9 and 7.10), similar to the suspended propeller case, the triplet blocks are observed. But interestingly it is only until a certain frequency that these triplets are exhibited. It is possible that a non-uniform mass loading from the surrounding fluid causes a breakdown in the cyclical symmetry effects in the propeller. It remains hard to say given the measurement layout inadequacy.

From table 7.12, the MAC values for the paired modes are seen to be in the range of 0.61 to 0.85. The percentage frequency differences between the measured and the calibrated FE modes range between -2.36% to 7.5%.

FE modes	Measured mode	MAC	Freq diff%	FE mode freq	Measured mode freq
5	3	0.778475	5.47189	654.42	692.302
7	6	0.687001	7.53811	792.955	857.602
8	5	0.678831	-0.675958	825.787	820.205
8	7	0.833175	6.22227	825.787	880.579
9	5	0.618792	-2.36758	840.095	820.205
9	7	0.852679	4.59743	840.095	880.579
10	7	0.768794	3.90209	846.218	880.579
12	8	0.769085	4.2656	1018.48	1063.86
14	9	0.759742	4.35754	1224.96	1280.77
15	10	0.822003	3.10277	1360.03	1403.58

Table 7.12: Calibrated modes vs measured modes MAC and frequency difference values for the submerged propeller

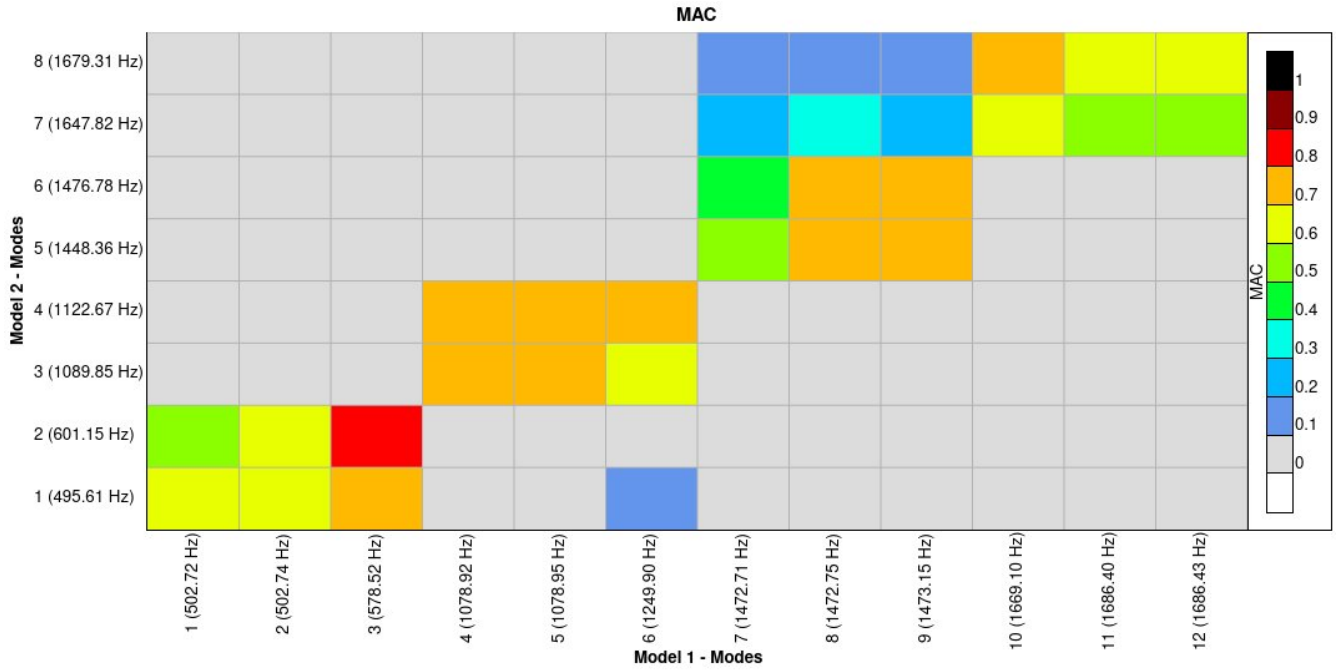


Figure 7.8: Suspended propeller MAC plot after calibration (x-axis: FE modes, y-axis: measured modes)

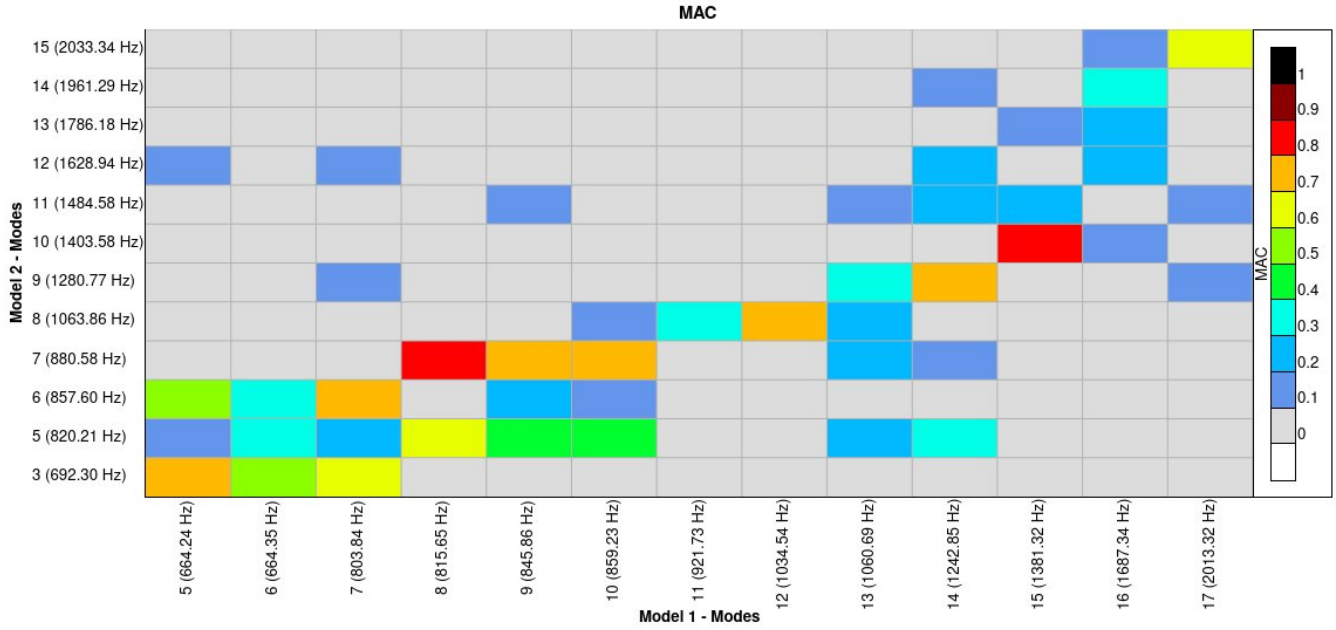


Figure 7.9: Submerged propeller MAC plot before calibration

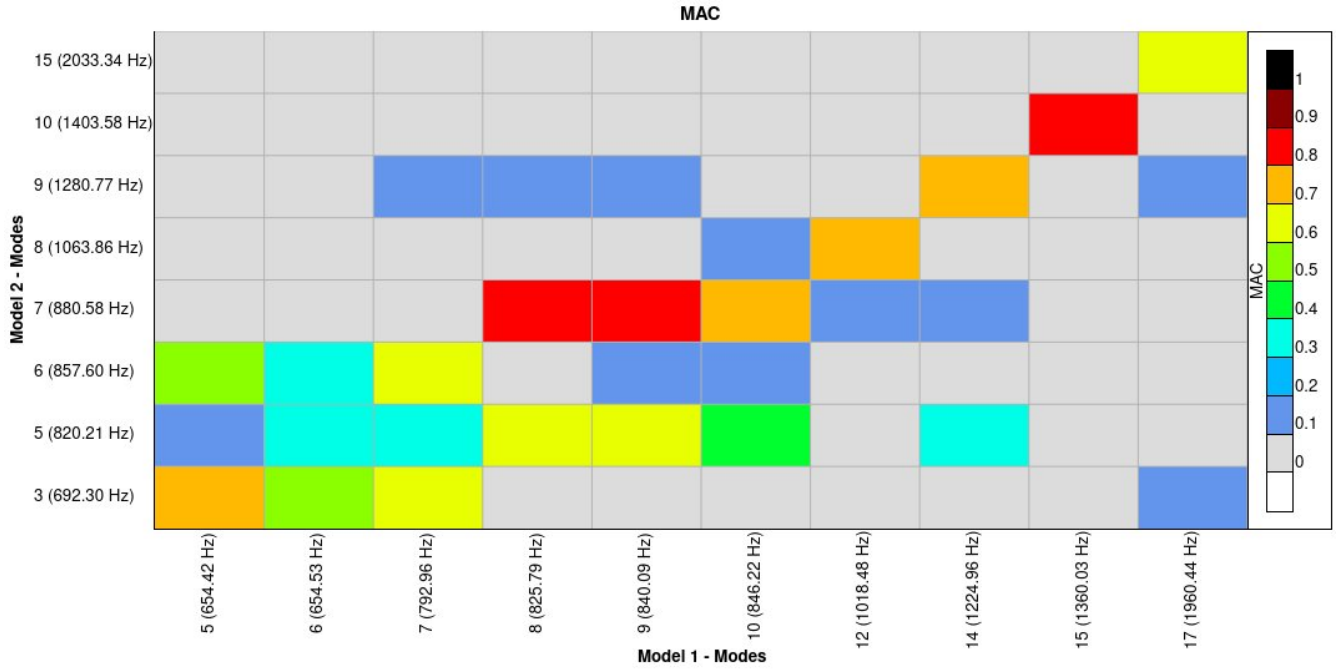


Figure 7.10: Submerged propeller MAC plot after calibration

8 Discussion

The plate suspended in air served as a clean validation case. With the equalized FRF approach [Abrahamsson and Kammer, 2015] and the reduced modal model, the plate's natural frequencies after calibration were within 3% of the measurements, and the MAC values were generally high (between 0.82 to 0.99). This confirms that the overall pipeline: UNV→CSV, measured FRF packing, reduced model construction, mode pairing with MAC, and least-squares fitting in the log-magnitude domain works as intended. The damping equalization also stabilized the comparison between the model and measurement by controlling peak sharpness.

The same pipeline transferred on to the submerged plate with minimal changes. Keeping the parameters fixed from the air case and calibrating only the fluid parameters gave plausible values, and the FRFs followed the main resonance peaks in the frequency band of interest. This showed that the combined FSI + calibration workflow is feasible and produces accurate results.

The suspended propeller in air was harder. The three-bladed geometry introduced near-degenerate triplet modes. Small assymetries in the geometry, mass, or boundary conditions split these triplets in ways that are hard to track across parameter changes, given the measurement methodology adopted. This complicates both mode pairing and gradient estimation in addition to the unrealistic residual values. In several runs, the optimizer pushed parameters to their bounds and the residual stayed high, as was expected. This is a classic sign of low identifiability. The measured data did not constrain those parameters well for the chosen output(single accelerometer) and input patter(hammer on one blade only).

In terms of the FE modeling choices, the second order tetrahedral elements in both structure and fluid gave robust meshes on complex geometry and better bending and wave accuracy than linear tets, at a manageable DOF count. This helped the modal results stabilize with moderate refinement and gradient steps (Δp). The Lanczos method for modal extraction on ADINA was a good fit for our large, sparse eigenproblems. It consistently returned the requested number of modes inside the frequency band, which helped later during pairing. The damping equalization simplified the comparison. We matched peak locations (via $K(p) = \omega^2$) and prevented unrealistic, very sharp peaks from dominating the residual.

When it comes to the sensitivity and gradients, the forward-difference gradients $\partial K/\partial p$ from the "_plus" runs were stable for the plate. On the propeller, the gradients for some parameters were very small over the used band. So, where possible, central differences were used to improve stability(using both "_plus" and "_minus" runs with matched mode counts). However, this doubled the solves per parameter and still depends on reliable pairing.

The FRF residuals are computed in the log-magnitude domain and can be dominated by wide dynamic range regions if the gains are not handled. Including a global gain inside the residual solved this. However, the overall difference remained high for the propeller, which can suggest that the measurement set does not excite/observe sufficiently for the needed mode families. This is not a coding or math issue at the end of the day, it is a measurement data coverage issue.

9 Conclusions

This work built and validated a complete, reproducible pipeline for calibrating FE models with measured FRFs, from meshing and coupled fluid-structure modal analysis to data handling, reduced model, pairing and optimization. On the suspended plate in air, the method produced very small frequency differences and high MAC values between the measured and calibrated FE modes, showing that the approach is sound when the structure is simple and the measurement is well controlled. On the submerged plate, the same method, with fluid parameters produced a very good calibration with very small frequency differences and high MAC values. It produced realistic fluid properties and tracked main resonances, demonstrating that the equalized FRF calibration extends to FSI cases.

On the propeller, the results were mixed. The pipeline executed as expected, but the combination of triplet modes, limited excitation/observation, and low parameter sensitivity led to high residuals and parameters drifting to bounds. The main conclusion is that the method is viable, but the data required must be richer for a rotating, cyclic structure. With better measurement coverage(more outputs and multiple blades excited), the same calibration procedure should become well conditioned for the propeller too.

10 Future Work

10.1 Measurement Plan Improvements for the Propeller

- **More outputs**: Use at least three accelerometers. One on the hub and one near the mid-span of two different blades. This should observe the different families of modes and reduce ambiguity.
- **Excite multiple blades**: Using the same roving hammer method, excite points on all three blades and not just one. This breaks symmetry and helps separate the triplet modes cleanly in the data.

10.2 Model and Parameter Set

- **Selectors from geometry**: Automate mapping of measurement coordinates to FE nodes, so U(inputs) and C(output) match the measured coordinates without the requirement of a manual coordinate feed. This should improve amplitude matching and reduce the optimizer's reliance on global gain.
- **Central differences by default**: If the runtime allows, include both "_plus" and "_minus" sets for all active parameters to improve gradient accuracy.
- **Mode dependent damping**: Replace the single ζ_0 with a frequency-dependent or modal-dependent damping if the data supports it.

10.3 Robust Pairing and Identifiability

- **Triplet handling**: Alternatively, modes can be grouped by MAC clusters within a small frequency gate and pair groups rather than single modes when the structure is cyclic. Restrict cross-group swaps but allow in-group permutations.
- **Bounds**: A solution to discourage edge solutions and bound hits when the data is weak can be explored

10.4 Tooling and Automation

- **Single code pipeline**: The option of all the steps of calibration codes(01, 02, 03) to be wrapped into a single program can be explored.
- **More format support**: Although the META exported CSVs work well, a tested OP2 reader for ADINA solutions would streamline processes.
- **FE model delta steps**: Automating the creation of parameter delta step solutions of the FE model through input file edits and rewrites based on a reference nominal parameter file can be explored.

Bibliography

Abrahamsson, T., & Kammer, D. (2015). Finite element model calibration using frequency responses with damping equalization. *Mechanical Systems and Signal Processing*, *60*, 372–389.

Bladh, R., Castanier, M. P., & Pierre, C. (2001). Component-mode-based reduced order modeling for mistuned bladed disks. *Journal of Engineering for Gas Turbines and Power*, *123*(1), 89–99.

Ewins, D. (2000). *Modal testing: Theory, practice and application*. Research Studies Press.

Richter, T. (2017). *Fluid-structure interactions*. Springer.

Appendix A: Python Codes

10.5 00_unv_to_csv.py

```
#!/usr/bin/env python3
"""
Strict UNV FRF exporter for titles:  FRF for
 ↪  Acc:NNN:+Z/Hammer:MMM:-Z  (no spaces around '/')

Run:
    %run scripts/00_unv_to_csv_titles.py
"""

from __future__ import annotations
from pathlib import Path
from typing import List, Tuple, Optional, Dict
import sys, re
import numpy as np
import pandas as pd

# -------- patterns --------
NUM = r'[+\-]?(?:\d+(?:\.\d*)?|\.\d+)(?:[eEdD][+\-]?\d+)?'
RE_NUM = re.compile(NUM)
RE_TITLE =
 ↪  re.compile(r'FRF\s+for\s+Acc:(\d+):\+Z/Hammer:(\d+):-Z',
 ↪  re.I)
RE_HDR =
 ↪  re.compile(rf'^\s*5\s+(\d+)\s+\d+\s+({NUM})\s+({NUM})\s+({NUM})\s*$',
 ↪  re.I)

def fnum(s: str) -> float:
    return float(s.replace('D','E').replace('d','e'))

def read_lines(p: Path) -> List[str]:
    txt = p.read_text(errors="ignore")
    # normalize EOL to \n
    return txt.replace('\r\n', '\n').replace('\r',
     ↪  '\n').splitlines()

def find_title_indices(lines: List[str]) ->
 ↪  List[Tuple[int,int,int]]:
    """Return list of (line_idx, out_id, in_id) for each
     ↪  matching title."""
    hits = []
    for i, L in enumerate(lines):
        m = RE_TITLE.search(L)
        if m:
            hits.append((i, int(m.group(1)), int(m.group(2))))
    return hits
```

```
def parse_block(lines: List[str], start_idx: int, next_idx:
↪  int) -> Optional[Dict]:

    # title ids
    mt = RE_TITLE.search(lines[start_idx])
    if not mt:
        return None
    out_id = int(mt.group(1)); in_id = int(mt.group(2))

    # search header anywhere in the block (after the title)
    npts = None; f0 = None; df = None
    for j in range(start_idx+1, next_idx):
        m = RE_HDR.match(lines[j].strip())
        if m:
            npts = int(m.group(1)); f0 = fnum(m.group(2)); df =
            ↪  fnum(m.group(3))
            break

    # collect ALL numeric tokens after the title to the next
    ↪  title
    tokens: List[float] = []
    for j in range(start_idx+1, next_idx):
        parts = RE_NUM.findall(lines[j])
        if parts:
            # avoid swallowing the integers from the header
            ↪  itself (we already read header if present)
            if RE_HDR.match(lines[j].strip()):
                continue
            # convert all numeric tokens on the line
            tokens.extend(fnum(p) for p in parts)

    if not tokens:
        return None

    # Decide payload mode
    if npts is not None and f0 is not None and df is not None:
        # Expect pairs: Re, Im repeating. Tolerate extra
        ↪  trailing numbers.
        need = 2 * npts
        if len(tokens) < need:
            # Sometimes numbers are split in blocks; still try
            ↪  with what we have (min)
            need = (len(tokens) // 2) * 2
        if need < 2:
            return None
        re_arr = np.array(tokens[0:need:2], dtype=float)
        im_arr = np.array(tokens[1:need:2], dtype=float)
        n = min(re_arr.size, im_arr.size)
        re_arr = re_arr[:n]; im_arr = im_arr[:n]
        f = f0 + df * np.arange(n, dtype=float)
        H = re_arr + 1j * im_arr
```

```
        else:
            # Triplets: f, Re, Im repeating
            # Make length a multiple of 3
            ntrip = (len(tokens) // 3)
            if ntrip == 0:
                return None
            use = 3 * ntrip
            arr = np.array(tokens[:use], dtype=float).reshape(-1,
             ↪  3)
            f = arr[:, 0]; H = arr[:, 1] + 1j * arr[:, 2]

        # ensure monotone increasing f
        if f.size and not np.all(np.diff(f) > 0):
            idx = np.argsort(f); f = f[idx]; H = H[idx]

        return dict(out_id=out_id, in_id=in_id, freq=f, H=H)

    def pick_master_grid(records: List[Dict]) -> np.ndarray:
        lo = max(r["freq"][0] for r in records if r["freq"].size)
        hi = min(r["freq"][-1] for r in records if r["freq"].size)
        if hi <= lo:
            raise RuntimeError("No common frequency overlap across
             ↪  selected blocks.")
        # choose df from the first record (median df)
        d = np.diff(records[0]["freq"])
        d = d[(d > 0) & np.isfinite(d)]
        df = float(np.median(d)) if d.size else
         ↪  float(records[0]["freq"][1]-records[0]["freq"][0])
        n  = int(np.floor((hi - lo)/df)) + 1
        f  = lo + df * np.arange(n, dtype=float)
        if f[-1] > hi:
            f = f[:-1]
        return f

    def interp_complex(f_src: np.ndarray, H_src: np.ndarray, f_tgt:
     ↪  np.ndarray) -> np.ndarray:
        mask = (f_src >= f_tgt[0]) & (f_src <= f_tgt[-1])
        if np.count_nonzero(mask) < 4:
            return np.full_like(f_tgt, np.nan, dtype=np.complex128)
        f = f_src[mask]
        Hr = H_src.real[mask]; Hi = H_src.imag[mask]
        Hr_i = np.interp(f_tgt, f, Hr); Hi_i = np.interp(f_tgt, f,
         ↪  Hi)
        return Hr_i + 1j*Hi_i

    def main():
        if len(sys.argv) < 3:
            print("Usage: 00_unv_to_csv_title_Zm_FINAL.py
             ↪  <UNV_or_TXT_path> <OUT_DIR>")
            sys.exit(2)
```

```
p_in = Path(sys.argv[1]); out_dir = Path(sys.argv[2])
out_dir.mkdir(parents=True, exist_ok=True)

lines = read_lines(p_in)
hits = find_title_indices(lines)
if not hits:
    # help debug
    with open(out_dir / "debug_first_lines.txt", "w",
     ↪  encoding="utf-8") as w:
        for i in range(min(300, len(lines))):
            w.write(lines[i] + "\n")
    raise RuntimeError("No title lines matched 'FRF for
     ↪  Acc:NNN:+Z/Hammer:MMM:-Z'.")

# make list of (start, next) boundaries
starts = [h[0] for h in hits]
bounds: List[Tuple[int,int]] = []
for i, s in enumerate(starts):
    e = starts[i+1] if i+1 < len(starts) else len(lines)
    bounds.append((s, e))

# parse each block
recs: List[Dict] = []
for (s, e) in bounds:
    r = parse_block(lines, s, e)
    if r and r["freq"].size and r["H"].size:
        recs.append(r)

if not recs:
    with open(out_dir / "titles_seen.txt", "w",
     ↪  encoding="utf-8") as w:
        for s, _ in bounds:
            w.write(lines[s] + "\n")
    raise RuntimeError("Titles were found but no usable
     ↪  numeric payloads parsed. See titles_seen.txt.")

# build common grid and export
f_master = pick_master_grid(recs)
recs.sort(key=lambda r: (r["in_id"], r["out_id"]))
exported = 0
for r in recs:
    Hi = interp_complex(r["freq"], r["H"], f_master)
    if np.isnan(Hi.real).any():
        continue
    name =
     ↪  f"H_Acc+Z_{r['out_id']:03d}__Imp-Z_{r['in_id']:03d}.csv"
    pd.DataFrame({"f_hz": f_master, "re": Hi.real, "im":
     ↪  Hi.imag}).to_csv(out_dir / name, index=False)
    exported += 1

print(f"Exported {exported} FRF CSVs to {out_dir}")
```

```
if __name__ == "__main__":
    main()
```

10.6 01_build_sysid_equalize.py

```
#!/usr/bin/env python3
"""
01_build_sysid_equalize_wet.py
Build (pass-through) an "equalized" FRF pack HXeq_FRF_wet.npz
 ↪  for the submerged plate case.

Inputs (choose ONE):
A) NPZ with:
   - f_hz : (nf,) float
   - H    : (ny, nu, nf) complex   (or H_re/H_im with same
    ↪  shape)
   - (optional) in_names, out_names : list[str]

B) Folder of CSVs (one per input) with columns:
   - f_hz (or freq_hz), and re/im (or Hre/Him)
   Typical submerged filenames: H_Acc+Z_1__ImpZ_pt###.csv

Output:
  02_proc/HXeq_FRF_wet.npz  with keys: f_hz, Hx_eq, in_names,
   ↪  out_names

Run:
    %run scripts/01_build_sysid_equalize_plate.py
"""

from pathlib import Path
from typing import Optional, List, Tuple
import argparse
import re
import glob
import os
import numpy as np

# -------- Paths --------
ROOT = Path(__file__).resolve().parents[1]
OUT  = ROOT / "02_proc" / "HXeq_FRF_wet.npz"
OUT.parent.mkdir(parents=True, exist_ok=True)

# -------- Helpers --------
def load_from_npz(path: Path) -> Tuple[np.ndarray, np.ndarray,
 ↪  List[str], List[str]]:
    d = np.load(path, allow_pickle=True)
    f = d.get("f_hz") or d.get("freq_hz") or d.get("f")
    if f is None:
        raise RuntimeError("NPZ missing frequency vector
         ↪  (f_hz/freq_hz/f).")
```

```
    H = d.get("H")
    if H is None:
        Hre = d.get("H_re"); Him = d.get("H_im")
        if Hre is None or Him is None:
            raise RuntimeError("NPZ must contain H (complex) or
            ↪  H_re/H_im.")
        H = Hre + 1j * Him
    in_names  = list((d.get("in_names") or d.get("u_names") or
    ↪  []))
    out_names = list((d.get("out_names") or d.get("y_names") or
    ↪  []))
    return np.asarray(f, float), np.asarray(H, complex),
    ↪  in_names, out_names

def _read_csv(path: Path) -> Tuple[np.ndarray, np.ndarray]:
    import pandas as pd
    df = pd.read_csv(path)
    # frequency column
    if "f_hz" in df.columns:
        f = df["f_hz"].to_numpy(float)
    elif "freq_hz" in df.columns:
        f = df["freq_hz"].to_numpy(float)
    else:
        raise RuntimeError(f"{path}: missing frequency column
        ↪  (f_hz or freq_hz)")
    # FRF columns (complex)
    if {"re","im"}.issubset(df.columns):
        Hc = df["re"].to_numpy(float) +
        ↪  1j*df["im"].to_numpy(float)
    elif {"Hre","Him"}.issubset(df.columns):
        Hc = df["Hre"].to_numpy(float) +
        ↪  1j*df["Him"].to_numpy(float)
    else:
        raise RuntimeError(f"{path}: missing FRF columns (re/im
        ↪  or Hre/Him)")
    return f, Hc

def _glob_inputs(csv_dir: Path, axis_tag: str) -> List[str]:
    # Be tolerant to underscore variants
    patt_variants = [
        csv_dir / f"H_Acc{axis_tag}_???__Imp-Z_???.csv",
        csv_dir / f"H_Acc{axis_tag}_1__ImpZ_pt*.csv",   # double
        ↪  underscore
        csv_dir / f"H_Acc{axis_tag}_1_ImpZ_pt*.csv",    # single
        ↪  underscore
        csv_dir / f"H_Acc{axis_tag}_1__Imp_pt*.csv",
        csv_dir / f"H_Acc{axis_tag}_1_Imp_pt*.csv",
        csv_dir / "input_*.csv",
    ]
    files = []
    for patt in patt_variants:
```

```
        files.extend(glob.glob(os.fspath(patt)))
    files = sorted(set(files))
    if not files:
        raise RuntimeError(f"No CSV files found under {csv_dir}
        ↪  for axis {axis_tag}")
    # Sort by pt###
    def pt_index(p: str):
        m = re.search(r"pt(\d+)", os.path.basename(p),
        ↪  re.IGNORECASE)
        return int(m.group(1)) if m else 10**9
    files.sort(key=pt_index)
    return files

def _ensure_sorted(f: np.ndarray, H: np.ndarray) ->
↪  Tuple[np.ndarray, np.ndarray]:
    if np.all(np.diff(f) > 0):
        return f, H
    idx = np.argsort(f)
    return f[idx], H[idx]

def _snap8(x: np.ndarray) -> np.ndarray:
    # snap to 8 decimals to kill CSV round-off differences
    return np.round(x.astype(float), 8)

def equalize_pass_through(H: np.ndarray, target_zeta: float =
↪  0.01) -> np.ndarray:
    # Keep the exact flow from the suspended script: do nothing
    ↪  here.
    return H

def load_from_csv_folder(csv_dir: Path, ny: int, nu:
↪  Optional[int], axis_tag: str = "+Z"):
    files = _glob_inputs(csv_dir, axis_tag)
    if nu is not None and len(files) != nu:
        print(f"[WARN] Found {len(files)} CSVs, expected {nu}.
        ↪  Proceeding with found files.")

    f_ref_snapped = None
    cols = []
    in_names = []
    out_names = ["Acc{axis_tag}_1"] * ny

    for j, p in enumerate(files):
        f, Hj = _read_csv(Path(p))
        f, Hj = _ensure_sorted(f, Hj)
        f_snap = _snap8(f)

        if f_ref_snapped is None:
            f_ref_snapped = f_snap  # store snapped reference
            ref_name = os.path.basename(p)
        else:
```

```
            if f_snap.size != f_ref_snapped.size:
                raise RuntimeError(f"Frequency size mismatch in
                ↪ {p}: {f_snap.size} vs ref
                ↪ {f_ref_snapped.size}")
            # allow tiny diffs via allclose; otherwise fail if
            ↪ >1e-4 Hz
            if not np.allclose(f_snap, f_ref_snapped,
            ↪ rtol=1e-9, atol=1e-6):
                max_abs = float(np.max(np.abs(f_snap -
                ↪ f_ref_snapped)))
                if max_abs > 1e-4:
                    raise RuntimeError(
                        f"Frequency grid mismatch in {p} (max
                        ↪ |f|={max_abs:.6g} Hz) "
                        f"vs reference {ref_name}. If needed we
                        ↪ can enable interpolation."
                    )
                else:
                    # tiny residual differences after
                    ↪ snapping--accept and proceed
                    pass

        cols.append(Hj)
        m = re.search(r"(Imp[Zz]?_?pt\d+|pt\d+)",
        ↪ os.path.basename(p))
        in_names.append(m.group(1) if m else
        ↪ f"input_{j+1:03d}")

    nin = len(cols)
    nf  = f_ref_snapped.size
    H = np.zeros((ny, nin, nf), dtype=complex)
    for j, Hj in enumerate(cols):
        if Hj.size != nf:
            # if any file differed by 1-2 rows purely due to
            ↪ sorting/headers (rare), trim/pad conservatively
            if Hj.size > nf:
                Hj = Hj[:nf]
            else:
                pad = np.zeros(nf, dtype=complex);
                ↪ pad[:Hj.size] = Hj; Hj = pad
        H[0, j, :] = Hj

    return f_ref_snapped, H, in_names, out_names

# -------- Main --------
def main():
    ap = argparse.ArgumentParser()
    ap.add_argument("--from_npz", type=str, help="Path to input
    ↪ FRF NPZ (optional)")
    ap.add_argument("--csv_dir", type=str, default=str(ROOT /
    ↪ "01_raw" / "frf_meas_wet"),
```

```
                    help="Folder of per-input CSVs (default:
                     ↪  01_raw/frf_meas_wet)")
    ap.add_argument("--ny", type=int, default=1)
    ap.add_argument("--nu", type=int, default=80, help="Number
     ↪  of inputs (default 80)")
    ap.add_argument("--axis", type=str, default="+Z",
     ↪  help="Accelerometer axis tag in filenames (default
     ↪  +Z)")
    ap.add_argument("--target_zeta", type=float, default=0.01,
     ↪  help="(kept for parity; pass-through)")
    args = ap.parse_args()

    if args.from_npz:
        f, H, in_names, out_names =
         ↪  load_from_npz(Path(args.from_npz))
    else:
        f, H, in_names, out_names =
         ↪  load_from_csv_folder(Path(args.csv_dir),
         ↪  ny=args.ny, nu=args.nu, axis_tag=args.axis)

    Hx_eq = equalize_pass_through(H,
     ↪  target_zeta=args.target_zeta)

    np.savez(OUT, f_hz=f, Hx_eq=Hx_eq,
             in_names=np.array(in_names, dtype=object),
             out_names=np.array(out_names, dtype=object))
    print(f"[OK] wrote {OUT}  | nin={H.shape[1]},
     ↪  nf={H.shape[2]}")

if __name__ == "__main__":
    main()



"""
import pandas as pd, numpy as np, glob, os
files =
 ↪  sorted(glob.glob(os.path.join("01_raw","frf_meas_wet","H_Acc+Z_1__ImpZ_pt
f0 = None
for p in files:
    df = pd.read_csv(p)
    col = "f_hz" if "f_hz" in df.columns else "freq_hz"
    f = df[col].to_numpy(float)
    if not np.all(np.diff(f) > 0):
        print("[WARN] not strictly ascending:",
         ↪  os.path.basename(p))
        f = f[np.argsort(f)]
    f = np.round(f, 8)
    if f0 is None:
        f0 = f; print("Ref grid ->", os.path.basename(p), "nf
         ↪  =", f.size)
```

```
    else:
        if f.size != f0.size:
            print("[ERR] size mismatch:", os.path.basename(p),
             ↪  f.size, "vs", f0.size)
        else:
            d = np.max(np.abs(f - f0))
            if d > 1e-6:
                print("[DIFF]", os.path.basename(p), "max |f|
                 ↪  =", d)
"""
```

10.7 02_build_reduction_from_op2.py

```
#!/usr/bin/env python3
"""
02_build_reduction_from_op2_prop.py
Builds reduced modal model (K0, Gs, U, C, w0, zeta0) from CSV
 ↪  exports,
and now also stores the nominal parameter vector so downstream
 ↪  tools
can use (p - p0) consistently.

Run (Spyder/IPython):
    %run scripts/02_build_reduction_from_op2_plate_Final.py
"""

from pathlib import Path
import json, yaml, numpy as np, pandas as pd
from scipy.optimize import linear_sum_assignment

ROOT = Path(__file__).resolve().parents[1]
CFG  = ROOT / "calib_config_wet.yaml"

# ---------- helpers ----------
def load_yaml(p: Path):
    with open(p, "r", encoding="utf-8") as f:
        return yaml.safe_load(f)

def read_freqs(csv_path: Path) -> np.ndarray:
    df = pd.read_csv(csv_path)
    lc = {c.lower(): c for c in df.columns}
    col = lc.get("freq_hz") or lc.get("f_hz") or
     ↪  lc.get("frequency_hz") or lc.get("f")
    if not col:
        raise RuntimeError(f"{csv_path}: missing frequency
         ↪  column (freq_hz/f_hz/f)")
    f = df[col].to_numpy(float).reshape(-1)
    if not np.all(np.diff(f) >= 0):
        f = f[np.argsort(f)]
    return f
```

```
def read_modes(modes_dir: Path) -> np.ndarray:
    files = sorted(modes_dir.glob("mode_*.csv"))
    if not files:
        raise RuntimeError(f"No mode_*.csv in {modes_dir}")
    V = []
    for p in files:
        df = pd.read_csv(p)
        lc = {c.lower(): c for c in df.columns}
        cx = lc.get("dispx") or lc.get("ux") or lc.get("x")
        cy = lc.get("dispy") or lc.get("uy") or lc.get("y")
        cz = lc.get("dispz") or lc.get("uz") or lc.get("z")
        if not (cx and cy and cz):
            raise RuntimeError(f"{p}: need Dispx/Dispy/Dispz
            ↪  (or ux/uy/uz)")
        v = df[[cx, cy, cz]].to_numpy(float)
        v = v / (np.linalg.norm(v) + 1e-16)
        V.append(v.ravel())
    return np.stack(V, axis=1)  # (3N, nmode)

def node_index_map(first_mode_csv: Path) -> dict:
    df0 = pd.read_csv(first_mode_csv)
    lc = {c.lower(): c for c in df0.columns}
    c_id = lc.get("node_id") or lc.get("id") or lc.get("nid")
    if not c_id:
        raise RuntimeError(f"{first_mode_csv}: need node_id
        ↪  column")
    node_ids = df0[c_id].to_numpy(int)
    return {int(nid): 3*i for i, nid in enumerate(node_ids)}  #
    ↪  base row for X; +2 for Z

def mac(A: np.ndarray, B: np.ndarray) -> np.ndarray:
    num = np.abs(A.conj().T @ B)**2
    den = (np.sum(np.abs(A)**2, axis=0)[:,None] *
    ↪  np.sum(np.abs(B)**2, axis=0)[None,:] + 1e-16)
    return num / den

def pair_map(Phi_nom: np.ndarray, Phi_var: np.ndarray,
↪  mac_min_keep: float):
    M = mac(Phi_nom, Phi_var)
    r, c = linear_sum_assignment(1.0 - M)
    macs = M[r, c]
    keep = macs >= mac_min_keep
    return {int(ri): int(ci) for ri,ci in zip(r[keep],
    ↪  c[keep])}, macs[keep]

def selectors_from_yaml(modes_dir: Path, Phi0: np.ndarray,
↪  yaml_path: Path):
    nm = Phi0.shape[1]
    if not yaml_path or not yaml_path.exists():
        print("[INFO] selectors YAML not found; using
        ↪  U=C=ones.")
```

```
        return np.ones((nm,1)), np.ones((1,nm))
    sel =
    ↪  yaml.safe_load(yaml_path.read_text()).get("selectors",
    ↪  {})
    hammer_nodes = [int(x) for x in sel.get("hammer_nodes",
    ↪  [])]
    accel_node   = sel.get("accel_node", None)
    first_mode = sorted(modes_dir.glob("mode_*.csv"))[0]
    base = node_index_map(first_mode)
    def zrow(nid): return base[int(nid)] + 2
    # C from accel Z (or mean hammer Z)
    if accel_node is not None:
        Cz = Phi0[zrow(accel_node), :].astype(float)
    elif hammer_nodes:
        Cz = np.mean(Phi0[[zrow(n) for n in hammer_nodes], :],
        ↪  axis=0).astype(float)
    else:
        Cz = np.ones(nm, float)
    # U as RMS over hammer Z (or ones)
    if hammer_nodes:
        Uz = np.sqrt(np.mean(Phi0[[zrow(n) for n in
        ↪  hammer_nodes], :]**2, axis=0)).astype(float)
    else:
        Uz = np.ones(nm, float)
    C = (Cz / (np.linalg.norm(Cz)+1e-16)).reshape(1,-1)
    U = (Uz / (np.linalg.norm(Uz)+1e-16)).reshape(-1,1)
    return U, C

# ---------- main ----------
def main():
    cfg = load_yaml(CFG)
    paths = cfg["paths"]
    fe_root = ROOT / paths["fe_csv_root"]
    out_npz = ROOT / paths["reduced_npz"]
    out_npz.parent.mkdir(parents=True, exist_ok=True)

    zeta0        = float(cfg.get("zeta0_equalized", 0.01))
    mac_min_ok   = float(cfg.get("mac_min_ok", 0.80))
    mac_min_keep = float(cfg.get("mac_min_keep", 0.60))
    band         = cfg.get("band_hz", None)
    selectors_yml = paths.get("selectors_yaml")
    selectors_yml = (ROOT / selectors_yml) if selectors_yml
    ↪  else None

    # Nominal
    f0   = read_freqs(fe_root/"nominal"/"frequencies.csv")
    Phi0 = read_modes(fe_root/"nominal"/"modes")
    if band:
        sel0 = (f0>=band[0]) & (f0<=band[1]); f0, Phi0 =
        ↪  f0[sel0], Phi0[:,sel0]
    w0 = 2*np.pi*f0
```

```
K0 = (w0**2).astype(float)
U, C = selectors_from_yaml(fe_root/"nominal"/"modes", Phi0,
↪  selectors_yml)

# Parameter order & p0 from YAML
param_order = [k for k in cfg["params"].keys() if "step_fd"
↪  in cfg["params"][k]]
p0_vals     = [float(cfg["params"][k]["nom"]) for k in
↪  param_order]

# Gradients (central if *_minus exists)
Gs = []
qc = {}
for pname in param_order:
    dp = float(cfg["params"][pname]["step_fd"])
    plus_dir = fe_root / f"{pname}_plus"
    if not plus_dir.exists():
        print(f"[WARN] skipping {pname}: no {plus_dir}")
        continue
    fP, PhiP = read_freqs(plus_dir/"frequencies.csv"),
    ↪  read_modes(plus_dir/"modes")
    if band:
        selP = (fP>=band[0]) & (fP<=band[1]); fP, PhiP =
        ↪  fP[selP], PhiP[:,selP]
    mapP, macsP = pair_map(Phi0, PhiP, mac_min_keep)
    wP = 2*np.pi*fP
    gdiag = np.zeros_like(K0)

    minus_dir = fe_root / f"{pname}_minus"
    if minus_dir.exists():
        fM, PhiM = read_freqs(minus_dir/"frequencies.csv"),
        ↪  read_modes(minus_dir/"modes")
        if band:
            selM = (fM>=band[0]) & (fM<=band[1]); fM, PhiM
            ↪  = fM[selM], PhiM[:,selM]
        mapM, macsM = pair_map(Phi0, PhiM, mac_min_keep)
        wM = 2*np.pi*fM
        common = sorted(set(mapP) & set(mapM))
        if not common:
            print(f"[WARN] {pname}: no common +/-; using
            ↪  one-sided plus")
            for r in mapP:
                gdiag[r] = (wP[mapP[r]]**2 - w0[r]**2) / dp
        else:
            for r in common:
                gdiag[r] = (wP[mapP[r]]**2 -
                ↪  wM[mapM[r]]**2) / (2.0*dp)
        qc[pname] = {
            "plus":  {"matched": len(mapP), "mac_mean":
            ↪  float(macsP.mean()) if macsP.size else 0.0,
```

```
                        "mac_min": float(macsP.min())  if
                         ↪  macsP.size else 0.0,
                        "nm_nom": int(Phi0.shape[1]),
                         ↪  "nm_var": int(PhiP.shape[1])},
                "minus": None if not minus_dir.exists() else
                        {"matched": len(mapM), "mac_mean":
                         ↪  float(macsM.mean()) if macsM.size
                         ↪  else 0.0,
                         "mac_min": float(macsM.min())  if
                          ↪  macsM.size else 0.0,
                         "nm_nom": int(Phi0.shape[1]),
                          ↪  "nm_var": int(PhiM.shape[1])},
            }
        else:
            for r in mapP:
                gdiag[r] = (wP[mapP[r]]**2 - w0[r]**2) / dp
            qc[pname] = {
                "plus": {"matched": len(mapP), "mac_mean":
                 ↪  float(macsP.mean()) if macsP.size else 0.0,
                        "mac_min": float(macsP.min())  if
                         ↪  macsP.size else 0.0,
                        "nm_nom": int(Phi0.shape[1]),
                         ↪  "nm_var": int(PhiP.shape[1])},
                "minus": None,
            }

        if (qc[pname]["plus"]["mac_min"] or 1) and
         ↪  qc[pname]["plus"]["mac_min"] < mac_min_ok:
            print(f"[WARN] {pname}: low MAC_min
             ↪  (plus={qc[pname]['plus']['mac_min']:.3f})")
        Gs.append(np.diag(gdiag))

    if not Gs:
        raise RuntimeError("No gradients built; check *_plus
         ↪  folders and YAML params[*].step_fd")

    # Save reduced model with p0
    np.savez(out_npz,
             K0=K0, Gs=np.array(Gs, dtype=object),
             U=U, C=C, w0=w0,
              ↪  zeta0=float(cfg["zeta0_equalized"]),
             param_names=np.array(param_order, dtype=object),
             p0_names=np.array(param_order, dtype=object),
             p0_values=np.array(p0_vals, dtype=float))
    with open(out_npz.with_suffix(".qc.json"), "w",
     ↪  encoding="utf-8") as f:
        json.dump(qc, f, indent=2)

    print(f"[OK] wrote {out_npz} | params={param_order}")
    print(f"[QC]  {out_npz.with_suffix('.qc.json')}")
```

```
if __name__ == "__main__":
    main()
```

10.8 03_calibrate.py

```
#!/usr/bin/env python3
"""
03_calibrate_prop.py
Single-stage FRF least-squares calibration (no staging).
- Reads measured FRFs (NPZ with f_hz + Hx_eq or H)
- Reads reduced model (K0, Gs, U, C, w0, zeta0,
 ↪  p0_names/p0_values)
- Minimizes log-magnitude residual with per-input global gains.

Run:
    %run scripts/03_calibrate_plate_Final.py
"""

from pathlib import Path
import json, yaml, numpy as np
from scipy.optimize import least_squares

ROOT = Path(__file__).resolve().parents[1]
CFG  = ROOT / "calib_config_wet.yaml"

# -------- loaders --------
def load_yaml(p: Path):
    with open(p, "r", encoding="utf-8") as f:
        return yaml.safe_load(f)

def load_frfs(npz_path: Path, inputs_used="all"):
    D = np.load(npz_path, allow_pickle=True)
    fk = next((k for k in ("f_hz","freq_hz","f") if k in
     ↪  D.files), None)
    if fk is None: raise RuntimeError("FRF NPZ missing
     ↪  f_hz/freq_hz/f")
    f = np.asarray(D[fk], float).ravel()
    if "Hx_eq" in D.files:
        H = D["Hx_eq"]
    elif "H" in D.files:
        H = D["H"]
    elif "H_re" in D.files and "H_im" in D.files:
        H = D["H_re"] + 1j*D["H_im"]
    else:
        raise RuntimeError("FRF NPZ must contain Hx_eq or H or
         ↪  H_re/H_im")
    H = np.asarray(H)
    if H.ndim == 2: H = H[None, :, :]
    if H.ndim != 3: raise RuntimeError(f"Unexpected H shape
     ↪  {H.shape}")
    if H.shape[-1] != f.size: raise RuntimeError("FRF length
     ↪  mismatch")
```

```
    if isinstance(inputs_used, str) and inputs_used.lower() ==
    ↪  "all":
        return f, H
    idx = (np.asarray(inputs_used, int) - 1)
    return f, H[:, idx, :]

def load_reduced(npz_path: Path):
    D  = np.load(npz_path, allow_pickle=True)
    K0 = np.asarray(D["K0"], float).reshape(-1)
    w0 = np.asarray(D["w0"], float).reshape(-1)
    z0 = float(D["zeta0"])
    # gradients as diagonal vectors
    Gs = []
    for Gi in list(D["Gs"]):
        Gi = np.asarray(Gi, float)
        g  = np.diag(Gi) if Gi.ndim == 2 else Gi.reshape(-1)
        Gs.append(g.astype(float))
    U  = np.asarray(D["U"], float).reshape(-1)
    C  = np.asarray(D["C"], float).reshape(-1)
    # nominal parameters
    p0_map = {}
    if "p0_names" in D.files and "p0_values" in D.files:
        names = [str(x) for x in D["p0_names"]]
        vals  = [float(x) for x in D["p0_values"]]
        p0_map = {n:v for n,v in zip(names, vals)}
    # order hint
    pnames = [str(x) for x in D.get("param_names", [])]
    return dict(K0=K0, Gs=Gs, U=U, C=C, w0=w0, zeta0=z0,
                p0_map=p0_map, param_names=pnames)

# -------- model & residual --------
def assemble_Kdiag(mdl, p_names, p_vals):
    Kdiag = mdl["K0"].copy()
    p0_map = mdl.get("p0_map", {})
    for i, name in enumerate(p_names[:len(mdl["Gs"])]):
        dp = float(p_vals[i] - p0_map.get(name, 0.0))
        Kdiag += dp * mdl["Gs"][i]
    return Kdiag

def synthesize_frf(f_hz, mdl, p_names, p_vals):
    w = 2*np.pi*np.asarray(f_hz, float)
    Kdiag = assemble_Kdiag(mdl, p_names, p_vals)
    Z  = (Kdiag[None,:] - w[:,None]**2) +
    ↪  2j*mdl["zeta0"]*mdl["w0"][None,:]*w[:,None]
    CU = mdl["C"].reshape(-1) * mdl["U"].reshape(-1)
    H  = -(w**2) * np.sum(CU[None,:] / Z, axis=1)  # (nf,)
    return H

def residual_logmag(p_vec, p_names, f, H_meas, mdl,
 ↪  compress="mean"):
    Hmod = synthesize_frf(f, mdl, p_names, p_vec)
```

```
    ny, nu, nf = H_meas.shape
    Hmod_rep = np.repeat(Hmod[None,None,:], nu, axis=1)  #
    ↪  (1,nu,nf)

    # per-input global gain (log domain mean match)
    eps = 1e-18
    log_meas = np.log10(np.maximum(np.abs(H_meas), eps))
    log_mod  = np.log10(np.maximum(np.abs(Hmod_rep), eps))
    g_log = (log_meas - log_mod).mean(axis=2, keepdims=True)  #
    ↪  (1,nu,1)
    Hmod_rep = (10.0**g_log) * Hmod_rep

    if compress == "mean":
        Hm  = H_meas.mean(axis=1, keepdims=True)
        Hmd = Hmod_rep.mean(axis=1, keepdims=True)
    elif compress == "first":
        Hm, Hmd = H_meas[:,:1,:], Hmod_rep[:,:1,:]
    else:
        Hm, Hmd = H_meas, Hmod_rep

    R = np.log10(np.maximum(np.abs(Hm),  eps)) \
      - np.log10(np.maximum(np.abs(Hmd), eps))
    return R.ravel()

# -------- main --------
def main():
    cfg   = load_yaml(CFG)
    paths = cfg["paths"]
    inputs_used = cfg.get("inputs_used", "all")

    f_all, H_all = load_frfs(ROOT/paths["frf_npz"],
    ↪  inputs_used)
    mdl          = load_reduced(ROOT/paths["reduced_npz"])

    # frequency band
    if cfg.get("band_hz"):
        b0, b1 = cfg["band_hz"]
    else:
        b0, b1 = float(f_all.min()), float(f_all.max())
    sel = (f_all >= b0) & (f_all <= b1)
    f, H = f_all[sel], H_all[:,:,sel]

    # active parameters (keep YAML order if present in reduced)
    all_params = cfg["params"]
    p_names = [k for k in (mdl["param_names"] or
    ↪  list(all_params.keys()))
               if k in all_params and
                ↪  bool(all_params[k].get("active", True))]
    if not p_names:
        raise RuntimeError("No active parameters; set
        ↪  params[*].active: true in YAML")
```

```
# p0 and bounds from YAML, but residual subtracts reduced p0_map
p0 = np.array([all_params[k]["nom"] for k in p_names], float)
lb = np.array([all_params[k]["min"] for k in p_names], float)
ub = np.array([all_params[k]["max"] for k in p_names], float)

def fun(x): return residual_logmag(x, p_names, f, H, mdl, compress=cfg.get("compress_inputs","mean"))

# check residual at start
r0 = fun(p0)
if not np.all(np.isfinite(r0)):
    raise RuntimeError("Non-finite residuals at nominal; check inputs/band.")

res = least_squares(fun, x0=p0, bounds=(lb,ub), method="trf",
                    max_nfev=int(cfg.get("max_nfev", 800)),
                    ftol=1e-9, xtol=1e-9, gtol=1e-9)
p_best = res.x

# summarize
results_dir = ROOT/paths["results_dir"]
results_dir.mkdir(parents=True, exist_ok=True)
best_path   = results_dir/"best_params_prop.json"
summ_path   = results_dir/"summary_prop.txt"

best = {k: float(v) for k,v in zip(p_names, p_best)}
with open(best_path, "w", encoding="utf-8") as f:
    json.dump(best, f, indent=2)

m = H[:, :1, :].size if cfg.get("compress_inputs","mean") in ("mean","first") else H.size
rmse_db = 20.0 * np.sqrt(res.cost / max(1, m))

with open(summ_path, "w", encoding="utf-8") as f:
    f.write("=== Calibration Summary (prop) ===\n")
    f.write(f"band_hz = [{b0:.3f}, {b1:.3f}]\n\n")
    for k, v0, vb in zip(p_names, p0, p_best):
        f.write(f"{k:12s}: nominal={v0:.6g}  best={vb:.6g}  delta={vb-v0:.6g}\n")
    f.write(f"\nstatus: success={res.success}, nfev={res.nfev}\n")
    f.write(f"message: {res.message}\n")
    f.write(f"cost={res.cost:.3e}, Residual RMSE_dB ~ {rmse_db:.2f} dB\n")
```

```
    print(f"[OK] wrote {best_path} and {summ_path}")
    print(f"[INFO] cost={res.cost:.3e}, nfev={res.nfev},
     ↪  RMSE_dB~{rmse_db:.2f}")

if __name__ == "__main__":
    main()
```

DEPARTMENT OF ARCHITECTURE AND
CIVIL ENGINEERING
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2026
www.chalmers.se



CHALMERS
UNIVERSITY OF TECHNOLOGY