Master's thesis

NTNU
Norwegian University of Science and Technology
Faculty of Engineering
Department of Structural Engineering

Emil Neby

Automatic Operational Modal Analysis of a Long-Span Suspension Bridge

Master's thesis in Engineering and ICT
Supervisor: Øyvind Wiig Petersen
Co-supervisor: Ole Andre Øiseth
June 2023

NTNU
Norwegian University of
Science and Technology

Emil Neby

Automatic Operational Modal Analysis of a Long-Span Suspension Bridge

Master's thesis in Engineering and ICT
Supervisor: Øyvind Wiig Petersen
Co-supervisor: Ole Andre Øiseth
June 2023

Norwegian University of Science and Technology
Faculty of Engineering
Department of Structural Engineering

NTNU
Norwegian University of
Science and Technology

Department of Structural Engineering
Faculty of Engineering
NTNU- Norwegian University of Science and Technology

ACCESSIBILITY

OPEN

MASTER THESIS 2023

SUBJECT AREA:	DATE:	NO. OF PAGES:
Structural Dynamics	June 11, 2023	66 + 31

TITLE:

Automatic Operational Modal Analysis of a Long-Span Suspension Bridge

Automatisk operasjonell modalanalyse av en hengebru med langt spenn

BY:

Emil Neby

SUMMARY:
Monitoring of structures is an important task to ensure safety in maintenance and operations, alongside verification of numerical models. Automatic operational modal analysis (AOMA) aims to identify modal parameters, providing the most important information about the global dynamic behaviour of a structure. The Hålogaland bridge is a relatively newly built long-span suspension bridge situated outside Narvik in northern Norway, with a monitoring system recording the motions of the bridge. The main objective of this thesis is to perform AOMA on vibration data from the Hålogaland bridge and investigate how modal parameters are influenced by environmental factors.

First, a framework for AOMA is implemented and assembled of already existing software components. Verification on a numerical example is carried out in order to assess the accuracy of the framework. The verification provided an overall detection rate of 96.4% and an average difference in natural frequency of 0.03%, compared to a numerical model of the example. The results made the framework suitable for further use.

The frequency content of the vibration data from the Hålogaland bridge is investigated in order to make proper adjustments of parameters in the framework. Furthermore, necessary preprocessing is done to prepare the data for modal analysis. AOMA is repeatedly performed on 30-minute time series of vibration data, from a total of one month from February 2022. Modal parameters obtained from a numerical model of the bridge is utilized as comparison to assess the results from the modal analysis. Finally, wind and temperature data from the given period, also recorded by the monitoring system, is analysed to examine if environmental factors influence the modal parameters.

In the frequency range of 0-1 Hz, AOMA is able to repeatedly detect a total of 24 modes of vibration from the bridge deck of the Hålogaland bridge. The framework provided an overall detection rate of 80.2% of the reference modes chosen from the numerical model, with a 2.2% average difference in natural frequency. The average computational time of each time series under investigation was 4.9 seconds. Mode shapes of all the detected modes coincided well with mode shapes obtained from the numerical model. Evidence of dynamic coupling between two modes closely spaced in the frequency domain is present and is most likely the reason for low detection rate of one of these modes.

Negative correlation between temperature and frequency is present for higher order modes, and the correlation is stronger the higher the order. This is indicating that the structure becomes stiffer at lower temperatures and the higher order modes vibrates at higher frequencies. Furthermore, the results show positive correlation between wind speed and estimated damping in the structure, in agreement with buffeting theory. Lower wind speeds reduces the excitation of the bridge, causing more scatter to appear amongst estimated modes located in the lowest frequency.

Finally, AOMA is able to identify more modes of vibrations on the Hålogaland bridge, compared to previously used methods for modal analysis, which supports the further use of AOMA.

RESPONSIBLE TEACHER: Øyvind Wiig Petersen
SUPERVISOR(S): Øyvind Wiig Petersen, Ole Andre Øiseth
CARRIED OUT AT: Department of Structural Engineering, NTNU

Abstract

Monitoring of structures is an important task to ensure safety in maintenance and operations, alongside verification of numerical models. Automatic operational modal analysis (AOMA) aims to identify modal parameters, providing the most important information about the global dynamic behaviour of a structure. The Hålogaland bridge is a relatively newly built long-span suspension bridge situated outside Narvik in northern Norway, with a monitoring system recording the motions of the bridge. The main objective of this thesis is to perform AOMA on vibration data from the Hålogaland bridge and investigate how modal parameters are influenced by environmental factors.

First, a framework for AOMA is implemented and assembled of already existing software components. Verification on a numerical example is carried out in order to assess the accuracy of the framework. The verification provided an overall detection rate of 96.4% and an average difference in natural frequency of 0.03%, compared to a numerical model of the example. The results made the framework suitable for further use.

The frequency content of the vibration data from the Hålogaland bridge is investigated in order to make proper adjustments of parameters in the framework. Furthermore, necessary preprocessing is done to prepare the data for modal analysis. AOMA is repeatedly performed on 30 minute time series of vibration data, from a total of one month from February, 2022. Modal parameters obtained from a numerical model of the bridge is utilized as comparison to assess the results from the modal analysis. Finally, wind and temperature data from the given period, also recorded by the monitoring system, is analysed to examine if environmental factors influence the modal parameters.

In the frequency range of 0-1 Hz, AOMA is able to repeatedly detect a total of 24 modes of vibration from the bridge deck of the Hålogaland bridge. The framework provided an overall detection rate of 80.2% of the reference modes chosen from the numerical model, with a 2.2% average difference in natural frequency. The average computational time of each time series under investigation was 4.9 seconds. Mode shapes of all the detected modes coincided well with mode shapes obtained from the numerical model. Evidence of dynamic coupling between two modes closely spaced in the frequency domain is present and is most likely the reason for low detection rate of one of these modes.

Negative correlation between temperature and frequency is present for higher order modes, and the correlation is stronger the higher the order. This is indicating that the structure becomes stiffer at lower temperatures and the higher order modes vibrates at higher frequencies. Furthermore, the results show positive correlation between wind speed and estimated damping in the structure, in agreement with buffeting theory. Lower wind speeds reduces the excitation of the bridge, causing more scatter to appear amongst estimated modes located in the lowest frequency.

Finally, AOMA is able to identify more modes of vibrations on the Hålogaland bridge, compared to previously used methods for modal analysis, which supports the further use of AOMA.

Sammendrag

Overvåkning av konstruksjoner er viktig for sikker vedlikehold og drift, samt verifikasjon av numeriske modeller. Automatisk operasjonell modalanalyse (AOMA) har som mål å identifisere modale parametere som gir den viktigste informasjonen om den globale dynamiske oppførselen til en konstruksjon. Hålogalandsbrua er en relativt ny hengebru med langt spenn utenfor Narvik i Nord-Norge, med et overvåkningssystem som registrerer bevegelsene til brua. Målet til denne avhandlingen er å utføre AOMA på vibrasjonsdata fra Hålogalandsbrua og undersøke hvordan miljøfaktorer påvirker modale parametere.

Først implementeres et rammeverk for AOMA ved å bruke allerede eksisterende komponenter med programvare. Verifisering på et numerisk eksempel gjennomføres for å vurdere nøyaktigheten til rammeverket. Verifiseringen gir en total deteksjonsrate på 96,4% og en gjennomsnittlig differanse i egenfrekvens på 0,03% sammenlignet med en numerisk modell av eksempelet. Resultatene viser at rammeverket er egnet til videre bruk.

Frekvensinnholdet i vibrasjonsdataene fra Hålogalandsbrua undersøkes for å gjøre korrekte justeringer av parametere i rammeverket. Videre blir nødvendig prosessering utført for å klargjøre dataene for modalanalyse. AOMA gjennomføres gjentatte ganger på 30 minutters tidsserier av vibrasjonsdata fra totalt en måned, fra februar 2022. Modale parametere hentes fra en numerisk modell av brua og brukes som sammenlikningsgrunnlag for å vurdere resultatene fra modalanalysen. Til slutt analyseres vind- og temperaturdata fra den gitte perioden, også registrert av overvåkingssystemet, for å undersøke om miljøfaktorer påvirker de modale parametrene.

I frekvensområdet 0-1 Hz klarer AOMA gjentatte ganger å oppdage totalt 24 bevegelsesmoder fra brudekket på Hålogalandsbrua. Rammeverket ga en total deteksjonsrate på 80,2% av referansemodene fra den numeriske modellen, med en gjennomsnittlig differanse på 2,2% i egenfrekvens. Gjennomsnittlig beregningstid for hver tidsserie var 4,9 sekunder. Modeformer for alle de oppdagede modene stemte godt overens med modeformer fra den numeriske modellen. Indikasjoner på dynamisk kobling mellom to moder som er tett plassert i frekvensdomenet, er til stede og er sannsynligvis årsaken til dårlig deteksjonsrate for en av disse modene.

Det er en negativ korrelasjon mellom temperatur og frekvens for høyere ordens moder, og korrelasjonen blir sterkere jo høyere orden det er snakk om. Dette indikerer at konstruksjonen blir stivere ved lavere temperaturer, og de høyere ordens modene vibrerer med høyere frekvenser. Videre viser resultatene en positiv korrelasjon mellom vindhastighet og estimert demping i konstruksjonen, noe som samsvarer med buffeting-teorien. Lavere vindhastigheter reduserer eksitasjonen av brua, noe som fører til at det oppstår større spredning blant estimerte moder som ligger i den laveste frekvensen.

AOMA identifiserer flere bevegelsesmoder på Hålogalandsbrua sammenlignet med tidligere anvendte metoder for modalanalyse, noe som støtter videre bruk av AOMA.

Preface

This thesis finalizes the Master of Science degree in Engineering and ICT at the Department of Structural Engineering, Norwegian University of Science and Technology (NTNU), in Trondheim. The thesis was written in the spring of 2023.

The work of this thesis has been done under supervision of Ph.D. Øyvind Wiig Petersen at the Department of Structural Engineering. Thank you for the help, I highly appreciate the support I have received during the process of writing this thesis. Also, I would like to thank Øyvind for providing a numerical model of the Hålogaland bridge, which gave valuable contributions to this thesis. Furthermore, I would like to thank Professor Ole Andre Øiseth for co-supervision and for providing time-synchronized data from the monitoring system of the Hålogaland bridge. Also, I would like to thank Ph.D. candidate Anno Christian Dederichs for evaluating my work and providing valuable feedback.

To my close friends and roomates Johan Grøgaard and Niels Semb. Thank you for making a loving home this last year in Trondheim. I appreciate the care and support you have shown me. A special thank you to Johan for reading through my thesis and providing valuable feedback.

I would like to thank my fellow students at the office at Materialteknisk for exchanging knowledge and creating a good social environment during the last year at NTNU.

To all the friends I have met throughout the journey to accomplish my degree, I truly cherish the friendships we have formed. Your presence has undeniably made this period memorable for me.

Lastly, I would like to thank my beloved family for always supporting me.

Emil Neby
Trondheim, Norway
June 11, 2023

Table of Contents

List of Symbols

Latin

Symbols	Explanation
$\mathbf{A}, \mathbf{A}_c$	discrete and continuous time state matrix
$\mathbf{B}, \mathbf{B}_c$	discrete and continuous time input influence matrix
$\bar{\mathbf{B}}$	component of the load vector containing input locations
B	width of bridge deck
$\mathbf{C}$	damping matrix
$\mathbf{C}_a$	acceleration output location matrix
$\mathbf{C}, \mathbf{C}_c$	discrete and continuous time output influence matrix
$\mathbf{C}_d$	displacement output location matrix
$\mathbf{C}_v$	velocity output location matrix
$\tilde{\mathbf{C}}$	modal damping matrix
$\mathbf{D}, \mathbf{D}_c$	discrete and continuous time direct transmission matrix
f	frequency in Hz
f_N	Nyquist frequency
f_s	sampling frequency
$\mathbf{G}$	next state-output covariance matrix
G_{xx}	one-sided auto spectral density function of x
$\hat{G}_{xx}$	estimate of the one-sided auto spectral density function of x
$\mathbf{I}$	identity matrix
i	number of block rows
$\mathbf{K}$	stiffness matrix
$\tilde{\mathbf{K}}$	modal stiffness matrix
l	number of sensor locations at the structure
$\mathbf{M}$	mass matrix
$\tilde{\mathbf{M}}$	modal mass matrix
N_{mod}	number of modes
n	order of Butterworth filter
n_d	number of segments in Welch's method
$\mathbf{O}_i$	observability matrix at time lag i
$\mathbf{p}$	load vector
$\tilde{\mathbf{p}}$	modal load vector
q_i	modal coordinates of mode i
$\mathbf{q}$	modal coordinate vector
$\mathbf{R}_i$	output correlation matrix at time lag i
$\hat{\mathbf{R}}_i$	estimate of the output correlation matrix at time lag i
R_{xx}	auto correlation function of x
R_{xy}	cross-correlation function between x and y
r	detection rate
r	Pearson correlation factor
S	stability of a cluster
S_{xx}	auto spectral density of x
S_{xy}	cross-spectral density between x and y
$\mathbf{s}$	state vector

Symbols	Explanation
$\mathbf{s}_k$	discrete time state vector
$\mathbf{T}_{1\|i}$	block-Toeplitz matrix with i rows and columns
T	period of a time series
t	time
$\mathbf{u}$	component of the load vector describing the time variation of the load
$\mathbf{u}_k$	sampled input vector
u_{Hanning}	the Hanning window
$\mathbf{v}_k$	process noise
$\mathbf{w}_k$	measurement noise
X	the Fourier transform of x
$\mathbf{Y}$	output matrix
$\mathbf{y}$	displacement vector
$\mathbf{y}_i$	modal displacement vector
$\mathbf{y}_k$	sampled output vector

Greek

Symbols	Explanation
α_λ	scaling factor for eigenvalues
α_m	scaling factor for modal assurance criterion
$\mathbf{\Gamma}_i$	reversed controllability matrix at time lag i
Δt	time interval between two samples in a record
$\mathbf{\Lambda}$	diagonal matrix containing discrete time eigenvalues
$\boldsymbol{\lambda}$	continuous time eigenvalue vector
λ	persistence of a cluster
λ_i	continuous time eigenvalue of mode i
ξ	damping ratio
τ	time lag
$\mathbf{\Phi}$	mode shape matrix
$\boldsymbol{\phi}_i$	mode shape vector of mode i
$\mathbf{\Psi}$	matrix containing eigenvectors
ω	frequency in rad/s
ω_c	cut-off frequency
ω_i	natural frequency of mode i

List of Abbreviations

Abbreviations	Explanation
AOMA	Automatic Operational Modal Analysis
API	Application Programming Interface
Cov-SSI	Covariance-driven Stochastic Subspace Identification
DOF	Degree of Freedom
FE	Finite Element
FEM	Finite Element Method
FFT	Fast Fourier Transform
GPS	Global Positioning System
HDBSCAN	Hierarchical Density-Based Spatial Clustering of Applications with Noise
HDF5	Hierarchical Data Format 5
OMA	Operational Modal Analysis
MAC	Modal Assurance Criterion
MDOF	Multiple Degrees of Freedom
MST	Minimum Spanning Tree
PDF	Probability Density Function
PSD	Power Spectral Density
SVD	Singular Value Decomposition
TDMS	Technical Data Management Streaming
ZOH	Zero Order Hold

Chapter 1

Introduction

1.1 Background

Increasing populations and global trade continue to put pressure on public infrastructure, leading to ever-higher demands for better performance. This includes better roads, both for higher volume of traffic, but also for shortening traveling distance. An example is the ongoing ferry free E39 project along Norway's western coast, which yet has several long bridges to be built. Another example is the newly built Hålogaland bridge located outside Narvik, in northern Norway. To ensure the safety of the bridge and to gain knowledge for similar future projects, the bridge has been equipped with sensors to monitor its behaviour.

There are several reasons for monitoring bridges. One is to validate the numerical models and estimates of the motion of the bridge carried out in advance of construction. If new discoveries are made, it would be desirable to make improvements.
Another motivation is to increase the knowledge of slender long span suspension bridges prone to high wind loading. Long span suspension bridges in foreign countries often have multiple lanes in each direction because of a higher general traffic volume. In district Norway traffic are often of a more sparse character, making it necessary to only have one lane in each direction. This makes the bridges slender with different behaviour, creating a demand for more research.
Still, safety and maintenance may be deemed the most important reasons for monitoring structures.

Operational modal analysis (OMA) aims to identify modal parameters of a structure during operation. In this context, modal parameters refers to natural frequency, damping ratio and mode shape. These parameters provides the most important information about the global dynamic behaviour of a structure. OMA entails to perform system identification on vibration data measured by accelerometers on multiple locations on the structure. Covariance-driven stochastic subspace identification (Cov-SSI) is a system identification method relying on a known number of modes to detect in advance. In general, this information is unavailable for big and complex structures. To overcome the problem, Cov-SSI is performed for a range of model orders. Finally, modal estimates are extracted from a stabilization diagram which traditionally is a manual interpretation task done by engineers.

Automatic operational modal analysis (AOMA) is a set of algorithms capable of automatic extract-

ing the modal estimates from the output of Cov-SSI over a range of orders, substituting the last manual interpretation stage of traditional OMA. AOMA relies on different clustering algorithms to identify the desired data points from the stabilization diagram. Motivations behind AOMA is to exclude human error, increase accuracy of the modal analysis and reduce cost by saving time.

AOMA has never before been applied on vibration data from the Hålogaland bridge. Solstad and Onstad [1] performed OMA as a sub task in their thesis, but data from only one period of 30 minutes was utilized for the purpose.

1.2 Problem Formulation

The goal of this study is to perform AOMA on long-term vibration data from the Hålogaland bridge to repeatedly identify its modal parameters. Measurement data from the Hålogaland bridge is provided by the Department of Structural Engineering at NTNU. Two software packages named STRID [2] and KOMA [3] will be utilized to perform the AOMA. The STRID package uses Cov-SSI for system identification and the KOMA package uses hierarchical density-based spatial clustering of applications with noise (HDBSCAN) for clustering and extraction of modal features. Prior to the main analysis of the vibration data from the bridge, a numerical example of a shear frame is generated to verify the accuracy of the framework for AOMA. Furthermore, data from the monitoring system of the bridge is extracted and preprocessed before it will be fed into the AOMA implementation of STRID and KOMA. The results are extracted, post-processed, and visualized for interpretation. Comparison with modal parameters obtained from a finite element (FE) model of the Hålogaland bridge is carried out in order to assess the accuracy of the AOMA framework. Wind and temperature data from the monitoring system of the bridge is also analysed in order to gain knowledge in how these factors influence the AOMA.

To summarize, the main objectives of this thesis are:

- **Implementation and numerical verification of an AOMA framework.**
  Utilize previously developed algorithms implemented in Python [4] to assemble a framework for AOMA. Verify the framework on a numerical example by multiple realizations of vibration data on a shear frame.
- **Apply AOMA on vibration data from the Hålogaland bridge.**
  This includes data exploration and preprocessing before application of AOMA on long-term vibration data from the bridge deck of the Hålogaland bridge to achieve quantitative results. Following comes an evaluation of performance by comparison with modal parameters obtained from a FE-model of the bridge. Finally, there will be a discussion of factors leading to uncertainty in the analysis.
- **Investigate how modal parameters are influenced by environmental factors.**
  Exploration of wind and temperature data and investigate if these environmental factors affects the ability to identify modes. Examine if change in weather patterns influence modal parameters behaviour.

1.3 Limitations

The main limitations of this thesis are:

- The assumption of stationarity applies to the response output in an AOMA context. The stationarity of the vibration data is not explored in detail in this study.
- Usage of FE-model from Abaqus as reference to find mode traces will affect the selection. Still-air modes obtained from the FE-model may deviate from the in-wind modes captured by the monitoring system on the bridge.
- Only one sensor is picked for wind and temperature recordings, respectively. This makes the weather data analysis less robust as one could have extracted averages over multiple sensors along the bridge span.

1.4 Structure of the thesis

The theory related to the algorithms and the general framework for AOMA is applicable for both cases in this thesis, and is therefore presented in one chapter. The numerical example and the Hålogaland bridge are presented in separate chapters, each with its own method, result and discussion part. Thus, the structure of the thesis becomes:

Chapter 2 presents the relevant theory for this thesis. First, theory related to signal processing is presented. Furthermore, theory related structural dynamics, state-space models and system identification is presented. Then the theory behind the chosen clustering algorithm is explained. Finally, a description of the framework for AOMA and the utilized software is given.

Chapter 3 is about the numerical example. First, the methodology is presented in addition to case specific parameters. The results from the simulation is presented before a discussion is given.

Chapter 4 is about the Hålogaland bridge. The methodology is presented, which starts with an introduction of the bridge and a description of the monitoring system. Furthermore, the FE-model of the bridge is presented, before data exploration, preprocessing and application of AOMA is described. The results are presented and finally a discussion is given.

Chapter 5 summarize the thesis with a conclusion and recommendation for further work.

Chapter 2

Theory

2.1 Signal processing

Time series data from sensors requires some techniques for exploration and preprocessing, in order to be suitable for AOMA. This section introduces the main methods utilized for data exploration and preprocessing.

2.1.1 Correlation and spectral density

Wind is considered the main loading of the Hålogaland bridge and can be assumed as a stationary and random process. The assumption of stationarity implies that all the statistical properties of the signal are independent of time [5]. It is important for further analysis to know how a time series correlates, both with itself and with other time series.

The auto correlation function $R_{xx}(\tau)$ of a random process $x(t)$ is defined as:

$$R_{xx}(\tau) = E[x(t)x(t+\tau)] \tag{2.1}$$

The cross-correlation functions, $R_{xy}(\tau)$ and $R_{yx}(\tau)$, between two different stationary random processes $x(t)$ and $y(t)$ are defined as:

$$R_{xy}(\tau) = E[x(t)y(t+\tau)] \tag{2.2}$$

$$R_{yx}(\tau) = E[y(t)x(t+\tau)] \tag{2.3}$$

where t is time, τ is the time lag and $E[*]$ is the expected value operator.

Spectral density functions are utilized to investigate the frequency content of a time series. If the zero value of a random process is normalized so that the mean value of the process is zero, the auto spectral density $S_{xx}(\omega)$ of the process can be obtained by the Fourier transform of its auto correlation function:

$$S_{xx}(\omega) = \frac{1}{2\pi}\int_{-\infty}^{\infty} R_{xx}(\tau)e^{-i\omega\tau}d\tau \tag{2.4}$$

where ω is the angular frequency. Accordingly, the cross-spectral densities, $S_{xy}(\omega)$ and $S_{yx}(\omega)$, of a pair of random processes are defined as the Fourier transforms of their cross-correlation functions respectively:

$$S_{xy}(\omega) = \frac{1}{2\pi}\int_{-\infty}^{\infty} R_{xy}(\tau)e^{-i\omega\tau}d\tau \tag{2.5}$$

$$S_{yx}(\omega) = \frac{1}{2\pi}\int_{-\infty}^{\infty} R_{yx}(\tau)e^{-i\omega\tau}d\tau \tag{2.6}$$

The auto correlation function can be found by averaging over an infinitely long time series:

$$R_{xx}(\tau) = E[x(t)x(t+\tau)] = \frac{1}{T}\int_{-T/2}^{T/2} x(t)x(t+\tau)dt, \quad T \to \infty \tag{2.7}$$

Inserting the expression into the auto spectral density yields:

$$S_{xx}(\omega) = \frac{1}{2\pi}\frac{1}{T}\int_{-\infty}^{\infty}\int_{-T/2}^{T/2} x(t)x(t+\tau)e^{-i\omega\tau}dtd\tau \tag{2.8}$$

The variable substitution $\alpha = t, \beta = t+\tau$ is utilized so that the integration variables becomes $d\alpha = dt, d\beta = d\tau$ and $\tau = \beta - \alpha$ in the exponent of Euler's number e. The double integral can be split into two single integrals:

$$\begin{aligned} S_{xx}(\omega) &= \frac{1}{2\pi}\frac{1}{T}\int_{-\infty}^{\infty}\int_{-\infty}^{\infty} x(\alpha)x(\beta)e^{i\omega\alpha}e^{-i\omega\beta}d\alpha d\beta \\ &= \frac{1}{2\pi}\frac{1}{T}(2\pi)^2 \underbrace{\left(\frac{1}{2\pi}\int_{-\infty}^{\infty} x(\alpha)e^{i\omega\alpha}d\alpha\right)}_{\bar{X}(\omega)} \underbrace{\left(\frac{1}{2\pi}\int_{-\infty}^{\infty} x(\beta)e^{-i\omega\beta}d\beta\right)}_{X(\omega)} = \frac{2\pi}{T}X(\omega)\bar{X}(\omega) \end{aligned} \tag{2.9}$$

where $X(\omega)$ is the Fourier transform of $x(t)$, and $\bar{X}(\omega)$ is its complex conjugate. To convert from rad/sec to Hz one can insert $\omega = \frac{f}{2\pi}$ to obtain:

$$S_{xx}(f) = \frac{1}{T}X(f)\bar{X}(f) \tag{2.10}$$

The one-sided autospectral density functions $G_{xx}(f)$ and $G_{yy}(f)$, where f varies over $(0, \infty)$ only, are defined by:

$$\begin{aligned} G_{xx}(f) &= 2S_{xx}(f) \quad 0 < f < \infty \quad \text{otherwise zero} \\ G_{yy}(f) &= 2S_{yy}(f) \quad 0 < f < \infty \quad \text{otherwise zero} \end{aligned} \tag{2.11}$$

The one-sided autospectral density function expressed by Fourier transforms is obtained by:

$$G_{xx}(f) = \frac{2}{T}|X(f)|^2 \tag{2.12}$$

For practical applications the term power spectral density (PSD) is used for the magnitude of the one-sided spectrum normalized by the frequency resolution.

2.1.2 Welch's method

It is desirable to compute the PSD of the time series to investigate the frequency content of the signal. To compute the exact spectrum is practically impossible since it would require an infinitely long time series without measurement noise. Welch's method is a method to estimate the PSD of a signal.

Welch's method [6] is based on the fast Fourier transform (FFT) [5] of the time series signal. Instead of taking the FFT of a single finite time signal, the signal is partitioned into n_d segments. The segments are partly overlapping with the neighbouring segments. The idea is then to take the FFT of each segment respectively and take the average over all the generated power spectrums to obtain an estimate of the PSD. Welch's method for estimating the PSD, $\hat{G}_{xx}(f)$, is given by:

$$\hat{G}_{xx}(f) = \frac{2}{n_d N \Delta t} \sum_{i=1}^{n_d} |X_i(f)|^2 \tag{2.13}$$

where N is the number of records in each segment, Δt is the time between each record and $X_i(f)$ is the FFT of a segment as a function of frequency f, measured in Hz.

A problem arises when taking the FFT of a time series of finite length where the period of the signal does not coincide with the record length. Discontinuities in frequency content in the borders of the segment allows energy at certain frequencies to spread to nearby frequencies, causing large amplitude errors. This phenomenon is known as leakage [6] and can introduce significant errors in the estimated spectra. To overcome this problem, the data is made periodic by tapering them by an appropriate time window. This suppresses the data in the beginning and end of the segment and removes the discontinuities between the segments. The most commonly used window is the Hanning window [6], which is given by:

$$u_{\text{Hanning}}(t) = \begin{cases} 1 - \cos^2(\frac{\pi t}{T}) & 0 \leq t \leq T \\ 0 & \text{elsewhere} \end{cases} \tag{2.14}$$

Because of the suppression in the beginning and the end of a segment, the use of the Hanning window introduces a loss factor of 3/8 which needs to be rescaled in order to obtain a PSD estimate with correct magnitude.

2.1.3 Low Pass Filtering

When the PSD of the signal is produced, one can investigate the frequency content of the time series and detect the frequency range of interest. It is desirable to remove the frequency content that is not interesting because it could potentially disturb the outcome of the analysis.

The Butterworth [7] filter is a frequently used low pass filter and is given by the amplitude response function:

$$|H(iw)|^2 = \frac{1}{1 + (\frac{\omega}{\omega_c})^{2n}} \tag{2.15}$$

where n is the filter order, ω_c is the cut off frequency and i is the imaginary unit. Figure 2.1 shows the amplitude response function over a range of filter orders with normalized cut-off frequency $\omega_c = 1$ rad/sec. The figure visualizes how amplitudes at lower frequencies are preserved and amplitudes of higher frequencies are removed. The higher the order of the filter, the more significantly the cut off.

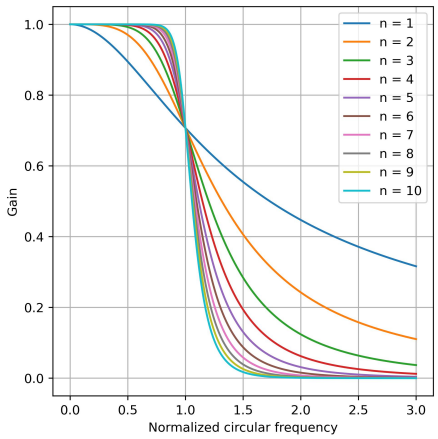


Figure 2.1: Amplitude response function for a set of order numbers.

2.1.4 Downsampling

When excess frequency content is removed by the appropriate low pass filter it is desirable to re-sample the signal to a suitable sampling rate for the analysis. The reason is to reduce the amount of data to analyze, while still being able to preserve the frequency content of interest. Too high of a sampling rate would simply lead to redundant data and increase computational time. Too low sampling rate can introduce aliasing, which can lead to misinterpretation of the frequency content of the signal. Figure 2.2 shows a simple example of aliasing.

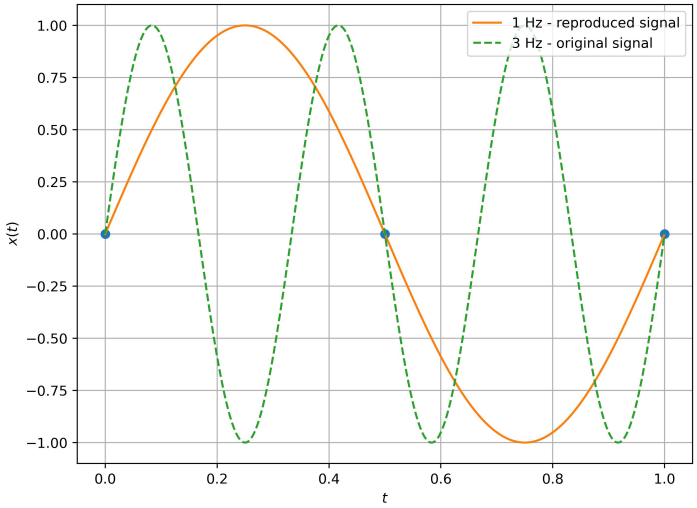


Figure 2.2: Illustration of aliasing. Original signal of 3 Hz recorded with a sampling frequency of 2 Hz is only able to reproduce a 1 Hz signal.

In order to avoid aliasing it is necessary to record the signal at a sampling rate of at least twice the highest frequency content of the original signal. This is also known as the Nyquist frequency [6]:

$$f_N = \frac{f_s}{2} \tag{2.16}$$

where f_s is the sampling frequency and f_N is the highest obtainable frequency in the recorded signal.

The Nyquist frequency sets the boundary for the lowest sampling rate to downsample a signal to, i.e. twice the rate of the highest frequency content of interest. The easiest way to downsample a signal is to create a new signal by picking selected samples with even distribution. For exemple, when downsampling a signal to half the sampling frequency, just pick every other sample in the original sample.

2.2 Structural Dynamics

2.2.1 Modal analysis

The Hålogaland bridge is a complex structure with multiple degrees of freedom (MDOF). For a MDOF system the the second order differential equation of motion in matrix notation is given by:

$$\mathbf{M}\ddot{\mathbf{y}} + \mathbf{C}\dot{\mathbf{y}} + \mathbf{K}\mathbf{y} = \mathbf{p}(t) \tag{2.17}$$

where $\mathbf{M}$, $\mathbf{C}$ and $\mathbf{K}$ denotes mass, damping and stiffness matrices respectively and $\mathbf{p}$ is the load vector. $\mathbf{y}$, $\dot{\mathbf{y}}$ and $\ddot{\mathbf{y}}$ denotes displacement, velocity and acceleration vectors respectively, containing

all degrees of freedom.

A MDOF system contains N_{mod} modes corresponding to the number of degrees of freedom (DOF). A modal expansion of the displacement vector $\mathbf{y}$ can be written on the form

$$\mathbf{y}(x,t) = \sum_{i=1}^{N_{mod}} \mathbf{y}_i(x,t) = \sum_{i=1}^{N_{mod}} \boldsymbol{\phi}_i(x) q_i(t) = \boldsymbol{\Phi}(x)\mathbf{q}(t) \tag{2.18}$$

where q_i are modal coordinates and $\boldsymbol{\phi}_i$ are the mode shape vectors assembled in the mode shape matrix $\boldsymbol{\Phi}$. Equation (2.18) expresses the classical mode displacement superposition method [8] because the total response of the system $\mathbf{y}$ is the sum of the modal responses $\mathbf{y}_i$.

The undamped natural frequencies and mode shapes of the system can be found by solving the eigenvalue problem:

$$(\mathbf{K} - \omega_i^2 \mathbf{M})\boldsymbol{\phi}_i = 0 \tag{2.19}$$

where ω_i is the natural frequency corresponding to mode shape $\boldsymbol{\phi}_i$.

Substituting equation (2.18) for $\mathbf{y}$ into equation (2.17) and pre-multiplying with $\boldsymbol{\Phi}^T$ leads to the uncoupled modal equation of motion:

$$\tilde{\mathbf{M}}\ddot{\mathbf{q}} + \tilde{\mathbf{C}}\dot{\mathbf{q}} + \tilde{\mathbf{K}}\mathbf{q} = \tilde{\mathbf{p}}(t) \tag{2.20}$$

where $\tilde{\mathbf{M}}$, $\tilde{\mathbf{C}}$ and $\tilde{\mathbf{K}}$ denotes the modal mass, damping and stiffness matrices respectively, and $\tilde{\mathbf{p}}$ is the modal load vector. The equations are uncoupled given the assumption of classical damping [8], where the natural modes of vibration are identical to those of the associated undamped system, and can be solved separately for each mode.

2.3 State-space models

2.3.1 Continuous-time state-space model

The second order differential equation given in equation (2.17) can be converted into a set of first order differential equations by the use of state-space models, which defines the state equation and the observation equation.

The state equation is derived from equation (2.17) by factorizing the load vector $\mathbf{p}(t)$ into the matrix $\bar{\mathbf{B}}$ containing the input locations of the load, and the vector $\mathbf{u}(t)$ describing the time variation of the load. Inserting into (2.17) the equation writes:

$$\mathbf{M}\ddot{\mathbf{y}} + \mathbf{C}\dot{\mathbf{y}} + \mathbf{K}\mathbf{y} = \bar{\mathbf{B}}\mathbf{u}(t) \tag{2.21}$$

which divided by the mass matrix, can be rewritten to

$$\ddot{\mathbf{y}} + \mathbf{M}^{-1}\mathbf{C}\dot{\mathbf{y}} + \mathbf{M}^{-1}\mathbf{K}\mathbf{y} = \mathbf{M}^{-1}\bar{\mathbf{B}}\mathbf{u}(t) \tag{2.22}$$

The state vector is defined:

$$\mathbf{s}(t) = \begin{bmatrix} \mathbf{y}(t) \\ \dot{\mathbf{y}}(t) \end{bmatrix} \tag{2.23}$$

By use of the identity $\mathbf{M}\dot{\mathbf{y}}(t) = \mathbf{M}\dot{\mathbf{y}}(t)$ and substituting the state vector into equation (2.22) to obtain the first order system of equations:

$$\dot{\mathbf{s}}(t) = \begin{bmatrix} -\mathbf{M}^{-1}\mathbf{K} & -\mathbf{M}^{-1}\mathbf{C} \\ \mathbf{0} & \mathbf{I} \end{bmatrix} \mathbf{s}(t) + \begin{bmatrix} -\mathbf{M}^{-1}\bar{\mathbf{B}} \\ \mathbf{0} \end{bmatrix} \mathbf{u}(t) \tag{2.24}$$

where $\mathbf{I}$ is the identity matrix, which all elements of the diagonal reads one and zero elsewhere. The state matrix $\mathbf{A}_c$ and the input influence matrix $\mathbf{B}_c$ are defined as:

$$\mathbf{A}_c = \begin{bmatrix} -\mathbf{M}^{-1}\mathbf{C} & -\mathbf{M}^{-1}\mathbf{K} \\ \mathbf{I} & \mathbf{0} \end{bmatrix} \tag{2.25}$$

$$\mathbf{B}_c = \begin{bmatrix} -\mathbf{M}^{-1}\bar{\mathbf{B}} \\ \mathbf{0} \end{bmatrix} \tag{2.26}$$

where the subscript c denotes continuous time. The state equation becomes:

$$\dot{\mathbf{s}}(t) = \mathbf{A}_c\mathbf{s}(t) + \mathbf{B}_c\mathbf{u}(t) \tag{2.27}$$

Assuming that the response of the structure are measured at l sensor locations, the observation equation can be written as:

$$\mathbf{y}_l(t) = \mathbf{C}_a\ddot{\mathbf{y}}(t) + \mathbf{C}_v\dot{\mathbf{y}}(t) + \mathbf{C}_d\mathbf{y}(t) \tag{2.28}$$

where $\mathbf{C}$ is the output location matrix, subscript a, v and d denotes acceleration, velocity and displacement respectively. Substituting equation (2.22) into (2.28) gives:

$$\mathbf{y}_l(t) = (\mathbf{C}_v - \mathbf{C}_a\mathbf{M}^{-1}\mathbf{C})\dot{\mathbf{y}}(t) + (\mathbf{C}_d - \mathbf{C}_a\mathbf{M}^{-1}\mathbf{K})\mathbf{y}(t) + (\mathbf{C}_a\mathbf{M}^{-1}\bar{\mathbf{B}})\mathbf{u}(t) \tag{2.29}$$

By introducing the output influence matrix $\mathbf{C}_c$ and the direct transmission matrix $\mathbf{D}_c$ defined as:

$$\mathbf{C}_c = \begin{bmatrix} \mathbf{C}_d - \mathbf{C}_a\mathbf{M}^{-1}\mathbf{K} & \mathbf{C}_v - \mathbf{C}_a\mathbf{M}^{-1}\mathbf{C} \end{bmatrix} \tag{2.30}$$

$$\mathbf{D}_c = \begin{bmatrix} \mathbf{C}_a\mathbf{M}^{-1}\bar{\mathbf{B}} \end{bmatrix} \tag{2.31}$$

the observation equation is rewritten to:

$$\mathbf{y}_l(t) = \mathbf{C}_c\mathbf{s}(t) + \mathbf{D}_c\mathbf{u}(t) \tag{2.32}$$

The state equation (2.27) and observation equation (2.32) defines the continuous-time state-space model.

2.3.2 Discrete-time state-space model

Real measurements on structures provide data measured in discrete time, hence a conversion to discrete time is necessary. A question arises on how to handle the data in between two timestamps, as a discrete model can not capture the continuity between two samples. The assumption of Zero Order Hold (ZOH) [6] states that the input is piece-wise constant over the sampling period. This assumption leads to the following relation between continuous-time matrices and discrete-time matrices:

$$\mathbf{A} = e^{\mathbf{A}_c\Delta t} \tag{2.33}$$

$$\mathbf{B} = (\mathbf{A} - \mathbf{I})\mathbf{A}_c^{-1}\mathbf{B}_c \tag{2.34}$$

$$\mathbf{C} = \mathbf{C}_c \tag{2.35}$$

$$\mathbf{D} = \mathbf{D}_c \tag{2.36}$$

The discrete-time state-space model then becomes:

$$\mathbf{s}_{k+1} = \mathbf{A}\mathbf{s}_k + \mathbf{B}\mathbf{u}_k \tag{2.37}$$

$$\mathbf{y}_k = \mathbf{C}\mathbf{s}_k + \mathbf{D}\mathbf{u}_k \tag{2.38}$$

where $\mathbf{A}$ is the discrete state matrix, $\mathbf{B}$ is the discrete input matrix, $\mathbf{C}$ is the discrete output influence matrix and $\mathbf{D}$ is the direct transmission matrix. $\mathbf{s}_k$ is the discrete-time state vector, $\mathbf{u}_k$ and $\mathbf{y}_k$ are sampled input and sampled output respectively.

2.3.3 Discrete-time stochastic state-space model

The input vector $\mathbf{u}_k$ in equation (2.37) and (2.38) is deterministic, which entails the discrete-time state-space model being deterministic. To account for the random characteristic of experimental input data, stochastic components needs to be included. When including stochastic components, $\mathbf{w}_k$ and $\mathbf{v}_k$, we obtain the discrete-time combined deterministic-stochastic state-space model:

$$\mathbf{s}_{k+1} = \mathbf{A}\mathbf{s}_k + \mathbf{B}\mathbf{u}_k + \mathbf{w}_k \tag{2.39}$$

$$\mathbf{y}_k = \mathbf{C}\mathbf{s}_k + \mathbf{D}\mathbf{u}_k + \mathbf{v}_k \tag{2.40}$$

where $\mathbf{w}_k$ and $\mathbf{v}_k$ is process noise and measurement noise due to model inaccuracies and sensor inaccuracies, respectively.

In the application of OMA, the deterministic input loads are unknown since they are not measured. Thus, measured system response is assumed generated by the stochastic processes only, which causes products of the deterministic input load $\mathbf{u}_k$ to be cancelled out. The following discrete-time stochastic state-space model is obtained:

$$\mathbf{s}_{k+1} = \mathbf{A}\mathbf{s}_k + \mathbf{w}_k \tag{2.41}$$

$$\mathbf{y}_k = \mathbf{C}\mathbf{s}_k + \mathbf{v}_k \tag{2.42}$$

The next state of the system $\mathbf{s}_{k+1}$ is related to the current state $\mathbf{s}_k$ through the state matrix $\mathbf{A}$ and process noise $\mathbf{w}_k$. The output $\mathbf{y}_k$ of the system is related to the current state $\mathbf{s}_k$ through the output influence matrix $\mathbf{C}$ and the measurement noise $\mathbf{v}_k$.

The objective of a stochastic state-space model is to determine the order n (number of DOFs) of the unknown system and a realization of $\mathbf{A}$ and $\mathbf{C}$ from a large number of measurements from the output $\mathbf{y}_k$. The process noise $\mathbf{w}_k$ and measurement noise $\mathbf{v}_k$ are both immeasurable, so they are assumed to be stationary white noise processes with zero mean and covariance matrices given by:

$$E\left[\begin{Bmatrix}\mathbf{w}_p\\ \mathbf{v}_p\end{Bmatrix}\begin{Bmatrix}\mathbf{w}_q^T & \mathbf{v}_q^T\end{Bmatrix}\right] = \begin{cases}\begin{bmatrix}\mathbf{Q}^{ww} & \mathbf{S}^{wv}\\ (\mathbf{S}^{wv})^T & \mathbf{R}^{vv}\end{bmatrix} & p = q\\ \mathbf{0} & p \neq q\end{cases} \tag{2.43}$$

where p and q are two arbitrary time instants. The output response $\mathbf{y}_k$ in the state-space model consequently becomes a zero mean Gaussian process, which output covariance matrices is given by:

$$\mathbf{R}_i = E[\mathbf{y}_{k+i}\mathbf{y}_k^T] \tag{2.44}$$

$\mathbf{R}_i$ contains all the information to describe the process. An estimated state-space model characterized by correct covariance may be defined, that is able to describe the statistical properties of the system response. Such a model is called a covariance equivalent model.

The state $\mathbf{s}_k$ is also a zero mean Gaussian process with covariance given by:

$$\mathbf{\Sigma} = E[\mathbf{s}_k\mathbf{s}_k^T] \tag{2.45}$$

where $\mathbf{\Sigma}$ is the large Greek letter sigma and must not be confused by the summation operator $\sum$. The current state $\mathbf{s}_k$ is uncorrelated with both process noise $\mathbf{w}_k$ and measurement noise $\mathbf{v}_k$:

$$E[\mathbf{s}_k\mathbf{w}_k^T] = \mathbf{0} \tag{2.46}$$

$$E[\mathbf{s}_k\mathbf{v}_k^T] = \mathbf{0} \tag{2.47}$$

It may be shown that the previous stated assumptions about noise terms, system state and response, leads to the following relations:

$$\mathbf{\Sigma} = \mathbf{A}\mathbf{\Sigma}\mathbf{A}^T + \mathbf{Q}^{ww} \tag{2.48}$$

$$\mathbf{R}_0 = \mathbf{C}\mathbf{\Sigma}\mathbf{C}^T + \mathbf{R}^{vv} \tag{2.49}$$

$$\mathbf{G} = \mathbf{A}\mathbf{\Sigma}\mathbf{C}^T + \mathbf{S}^{wv} \tag{2.50}$$

$$\mathbf{R}_i = \mathbf{C}\mathbf{A}^{i-1}\mathbf{G} \tag{2.51}$$

Where $\mathbf{G}$ is the next state-output covariance matrix defined as:

$$\mathbf{G} = E[\mathbf{s}_{k+1}\mathbf{y}_k^T] \tag{2.52}$$

which is describing the covariance between the response of the system $\mathbf{y}_k$ and the next state $\mathbf{s}_{k+1}$. The property of the output covariance sequence $\mathbf{R}_i$ is important because it can be directly estimated from measured data. Its decomposition allows estimation of the state-space matrices and the solution of the system identification problem. There are several methods available to perform such estimations, this thesis utilizes Covariance-driven Stochastic Subspace Identification (Cov-SSI).

2.4 Operational Modal Analysis

2.4.1 Covariance-driven Stochastic Subspace Identification

Cov-SSI aims to solve the problem of identifying a stochastic state-space model from output-only data. The output from l different channels, or sensors, can be collected in a matrix:

$$\mathbf{Y} = \begin{bmatrix} \mathbf{y}_0^T \\ \mathbf{y}_1^T \\ \vdots \\ \mathbf{y}_l^T \end{bmatrix} \tag{2.53}$$

where $\mathbf{y}_j, j \in (0, l)$ has the length N, which is number of samples in each channel. The estimate of the output correlation matrix at time lag i of a finite number of data records are given by:

$$\hat{\mathbf{R}}_i = \frac{1}{N-i}\mathbf{Y}_{(1:N-i)}\mathbf{Y}_{(i:N)}^T \tag{2.54}$$

The estimated correlations at different time lags can be collected into the block-Toeplitz matrix:

$$\mathbf{T}_{1|i} = \begin{bmatrix} \hat{\mathbf{R}}_i & \hat{\mathbf{R}}_{i-1} & \dots & \hat{\mathbf{R}}_1 \\ \hat{\mathbf{R}}_{i+1} & \hat{\mathbf{R}}_i & \dots & \hat{\mathbf{R}}_2 \\ \vdots & \vdots & \ddots & \vdots \\ \hat{\mathbf{R}}_{2i-1} & \hat{\mathbf{R}}_{2i-2} & \dots & \hat{\mathbf{R}}_i \end{bmatrix} \tag{2.55}$$

Where the maximum number of block rows i in the matrix is limited by the system order n and the size $l \times l$ of the correlation matrix $\hat{\mathbf{R}}_i$:

$$i \times l \geq n \tag{2.56}$$

By utilizing the relation in equation (2.51) one can create a decomposition of the block-Toeplitz matrix:

$$\mathbf{T}_{1|i} = \mathbf{O}_i \mathbf{\Gamma}_i \tag{2.57}$$

where the $\mathbf{O}_i$ is the observability matrix:

$$\mathbf{O}_i = \begin{bmatrix} \mathbf{C} \\ \mathbf{CA} \\ \mathbf{CA}^2 \\ \vdots \\ \mathbf{CA}^{i-1} \end{bmatrix} \tag{2.58}$$

and $\mathbf{\Gamma}_i$ is the reversed controllability matrix:

$$\mathbf{\Gamma}_i = \begin{bmatrix} \mathbf{A}^{i-1}\mathbf{G} & \dots & \mathbf{AG} & \mathbf{G} \end{bmatrix} \tag{2.59}$$

Performing a Singular Value Decomposition (SVD) [9] on the block-Toepltiz matrix leads to:

$$\mathbf{T}_{1|i} = \begin{bmatrix} \mathbf{U}_1 & \mathbf{U}_2 \end{bmatrix} \begin{bmatrix} \mathbf{\Sigma}_1 & \mathbf{0} \\ \mathbf{0} & \mathbf{0} \end{bmatrix} \begin{bmatrix} \mathbf{V}_1^T \\ \mathbf{V}_2^T \end{bmatrix} = \mathbf{U}_1 \mathbf{\Sigma}_1 \mathbf{V}_1^T \tag{2.60}$$

were $\mathbf{\Sigma}_1$ holds the non-zero singular values in decreasing order. It can be shown that from equation (2.60) one can obtain a new expression for the observability matrix and the reversed controllability matrix:

$$\mathbf{O}_i = \mathbf{U}_1 \mathbf{\Sigma}_1^{1/2} \tag{2.61}$$

$$\mathbf{\Gamma}_i = \mathbf{\Sigma}_1^{1/2} \mathbf{V}_1^T \tag{2.62}$$

It can be seen from equation (2.58) that the output influence matrix $\mathbf{C}$ corresponds to the first block row of $\mathbf{O}_i$ and from equation (2.59) that the next state-output covariance matrix $\mathbf{G}$ corresponds to the last column of $\mathbf{\Gamma}_i$. The state matrix $\mathbf{A}$ can be computed based on the decomposition property

of the one-lag shifted block-Toeplitz matrix:

$$\mathbf{T}_{2|i+1} = \begin{bmatrix} \hat{\mathbf{R}}_{i+1} & \hat{\mathbf{R}}_i & \dots & \hat{\mathbf{R}}_2 \\ \hat{\mathbf{R}}_{i+2} & \hat{\mathbf{R}}_{i+1} & \dots & \hat{\mathbf{R}}_3 \\ \vdots & \vdots & \ddots & \vdots \\ \hat{\mathbf{R}}_{2i} & \hat{\mathbf{R}}_{2i-1} & \dots & \hat{\mathbf{R}}_{i+1} \end{bmatrix} = \mathbf{O}_i \mathbf{A} \boldsymbol{\Gamma}_i \tag{2.63}$$

Inserting equation (2.61) and (2.62) into (2.63), and solving for $\mathbf{A}$, one obtains the following expression for the state matrix:

$$\mathbf{A} = \mathbf{O}_i^+ \mathbf{T}_{2|i+1} \boldsymbol{\Gamma}_i^+ = \boldsymbol{\Sigma}_1^{-1/2} \mathbf{U}_1^T \mathbf{T}_{2|i+1} \mathbf{V}_1 \boldsymbol{\Sigma}_1^{-1/2} \tag{2.64}$$

where the $^+$ operator denotes the pseudo-inverse of a matrix.

Modal parameters can be found by an eigenvalue decomposition [6] of the state matrix $\mathbf{A}$:

$$\mathbf{A} = \boldsymbol{\Psi} \boldsymbol{\Lambda} \boldsymbol{\Psi}^{-1} \tag{2.65}$$

where $\boldsymbol{\Psi}$ contains the eigenvectors of $\mathbf{A}$, and $\boldsymbol{\Lambda}$ is a diagonal matrix containing the eigenvalues in the discrete time domain. Using equation (2.33) to obtain the continuous time eigenvalue vector $\boldsymbol{\lambda}$:

$$\boldsymbol{\lambda} = \frac{1}{\Delta t} \ln(\boldsymbol{\Lambda}) \tag{2.66}$$

Utilizing the assumption for underdamped systems [8] that $\xi < 1$, let $\lambda_i, i \in (0, n)$ be the eigenvalue of mode i where n is the system order, then λ_i is given by:

$$\lambda_i = \omega_i \left(-\xi_i \pm i\sqrt{1 - {\xi_i}^2} \right) \tag{2.67}$$

which comes as complex conjugate pairs and leads to the well known dynamic properties [10]:

$$\omega_i = |\lambda_i| \tag{2.68}$$

$$f_i = \frac{\omega_i}{2\pi} \tag{2.69}$$

$$\xi_i = \frac{\mathrm{Re}(\lambda_i)}{|\lambda_i|} \tag{2.70}$$

where ω_i is the natural frequency of mode i in rad/sec, f_i is the natural frequency of mode i in Hz and ξ_i is the damping ratio of mode i. The operator Re(*) obtains the real part of a complex number. The mode shape matrix $\boldsymbol{\Phi}$ is found from:

$$\boldsymbol{\Phi} = \mathbf{C} \boldsymbol{\Psi} \tag{2.71}$$

2.4.2 Reference-based stochastic subspace identification

For complex structures with extensive monitoring systems and many sensors, the output vector $\mathbf{y}_k$ can quickly become large. Since the output correlation matrix $\hat{\mathbf{R}}_i$ is a quadratic function of the output vector, the computational cost becomes high. Reference-based stochastic subspace identification avoids computation of all covariances, where the key is the projection of the row space of the future outputs into the row space of the past outputs. Reference sensors are chosen among the already existing sensors, so that they are able to capture global mode shapes of the structure. That is, candidates for the reference outputs are these sensors where it is expected that all modes of vibration are present in the measured data [11].

Now, let there be a set of reference outputs $\mathbf{y}_k^{\text{ref}}$ that is a subset of $\mathbf{y}_k$:

$$\mathbf{y}_k = \begin{bmatrix} \mathbf{y}_k^{\text{ref}} \\ \mathbf{y}_k^{\sim\text{ref}} \end{bmatrix} \tag{2.72}$$

where $\mathbf{y}_k^{\sim\text{ref}}$ is the non-reference outputs. The output covariance matrices between all outputs and the references becomes:

$$\mathbf{R}_i^{\text{ref}} = E[\mathbf{y}_{k+i}{\mathbf{y}_k^{\text{ref}}}^T] \tag{2.73}$$

and the next state-reference output covariance matrix becomes:

$$\mathbf{G}_i^{\text{ref}} = E[\mathbf{s}_{k+i}{\mathbf{y}_k^{\text{ref}}}^T] \tag{2.74}$$

Equation (2.51) leads to:

$$\mathbf{R}_i^{\text{ref}} = \mathbf{C}\mathbf{A}^{i-1}\mathbf{G}^{\text{ref}} \tag{2.75}$$

It can be shown that the observability matrix (2.58) becomes unchanged and that the reference reversed controllability matrix becomes:

$$\mathbf{\Gamma}_i^{\text{ref}} = \begin{bmatrix} \mathbf{A}^{i-1}\mathbf{G}^{\text{ref}} & \dots & \mathbf{A}\mathbf{G} & \mathbf{G}^{\text{ref}} \end{bmatrix} \tag{2.76}$$

Inserting all the estimated reference covariances into the block-Toeplitz matrix (2.55) yields the following decomposition:

$$\mathbf{T}_i^{\text{ref}} = \mathbf{O}_i\mathbf{\Gamma}_i^{\text{ref}} \tag{2.77}$$

It can be shown that by applying a SVD equivalent to (2.60) to (2.77), the observability matrix $\mathbf{O}_i$ in (2.61) becomes unchanged and the referenced reversed controllability matrix becomes:

$$\mathbf{\Gamma}_i^{\text{ref}} = \mathbf{\Sigma}_1^{1/2}\mathbf{V}_1^T \tag{2.78}$$

Once $\mathbf{O}_i$ and $\mathbf{\Gamma}_i^{\text{ref}}$ are known, it is possible to obtain a solution to the system identification problem. The state matrix $\mathbf{A}$ is once again computed from a decomposition of the shifted block-Toeplitz matrix:

$$\mathbf{T}_{2|i+1}^{\text{ref}} = \mathbf{O}_i \mathbf{A} \mathbf{\Gamma}_i^{\text{ref}} \tag{2.79}$$

solving for $\mathbf{A}$ and inserting equation (2.61) and (2.78):

$$\mathbf{A} = \mathbf{O}_i^{+} \mathbf{T}_{2|i+1}^{\text{ref}} \mathbf{\Gamma}_i^{\text{ref}^{+}} = \mathbf{\Sigma}_1^{-1/2} \mathbf{U}_1^T \mathbf{T}_{2|i+1}^{\text{ref}} \mathbf{V}_1 \mathbf{\Sigma}_1^{-1/2} \tag{2.80}$$

Modal parameters are found by equation (2.65) and dynamic properties according to the procedures described in the end of section 2.4.1.

As mentioned in the introduction, system identification is in general performed for a range of model orders, establishing a large set of poles. These poles can either be pure mathematical solutions to the eigenvalue problem, represent noise in the system or be actual physical modes of the system. Techniques to distinguish between physical modes and spurious poles will be presented in the next section.

2.5 Clustering Algorithms

Clustering is the task of grouping similar objects together into clusters, based on information only found in the data [12]. Hence, clustering is a subset of unsupervised machine learning algorithms which do not rely on labeled data. There are many categories of clustering algorithms, the most important are hierarchical clustering, centroid-based clustering, distribution-based clustering, density-based clustering and grid-based clustering. HDBSCAN is a newer development that combines algorithms from multiple categories. In an AOMA context, HDBSCAN is utilized to distinguish between physical modes and spurious poles established by system identification methods.

Before the algorithm is explained, it will be useful to establish some basic terms from graph theory. A graph consists of a set of vertices that may be connected together by a set of edges. In an undirected graph, the set of edges does not have any direction. A weighted graph is a graph for which each edge has an associated weight. An undirected graph is connected if every vertex is reachable from all other vertices. Connected components is a set of vertices in a graph that are linked to each other by edges. A graph with no simple cycles is acyclic. A free tree is a connected, acyclic, undirected graph. A rooted tree is a free tree in which one of the vertices is distinguished from the others, and this vertex is the root. Vertices of a rooted tree is often referred to as nodes. Consider a node x in a rooted tree. Any node y on the unique simple path from the root node to x is an ancestor of x. If y is an ancestor of x, then x is a descendant of y. If the last edge on a simple path from the root node to x is the edge between x and y, then y is the parent of x, and x is the child of y. A node with no children is a leaf node [13]. Figure 2.3 shows an example of a weighted rooted tree.

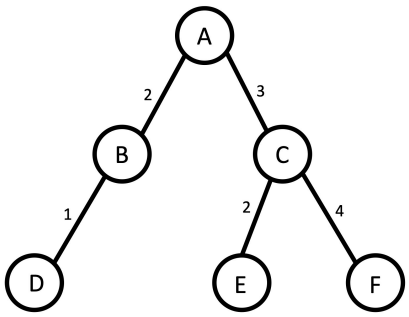


Figure 2.3: A weighted rooted tree. Node A is the root, node D, E and F are leaf nodes. Node B is a child of A and the parent of D. A is the ancestor of all the other nodes. F is a descendant of A and C. The weight of the edge between A and B is 2.

2.5.1 HDBSCAN

Hierarchical Density-Based Spatial Clustering of Applications with Noise (HDBSCAN) is a clustering algorithm developed by Campello, Moulavi and Sander, of which original paper [14] was published in 2015. The HDBSCAN is an assembly of many well known algorithms in the world of data science and is quite comprehensive in its entirety. This section only gives a brief overview of how the algorithm works and is inspired by the hdbscan python implementation documentation [15]. For a more thorough explanation, see the original paper [14].

The HDBSCAN algorithm can be broken into a series of 5 main steps:

1. Local density approximation
2. Build a Minimum spanning tree
3. Construct a cluster hierarchy
4. Condense the cluster hierarchy
5. Extract stable clusters

Local density approximation

The basic intuition when detecting clusters is to find the islands of higher density in a "sea" of sparser noise. The assumption of noise is important because real data is messy and has outliers, which also applies for the poles produced from system identification methods. A single noise data point can act as a bridge between two high density islands, which would be unfortunate. It is desirable to have an algorithm that is robust against noise, so it will only connect dense areas.

To be able to connect dense points into clusters, an estimate of density is necessary to distinguish high density points from sparse points. The core distance $\text{core}_k(x)$ of point x is defined as the distance from x to its k-th nearest neighbour in euclidean space. The mutual reachability distance is defined as:

$$d_{mreach-k}(a,b) = \max\{\text{core}_k(a), \text{core}_k(b), d(a,b)\} \tag{2.81}$$

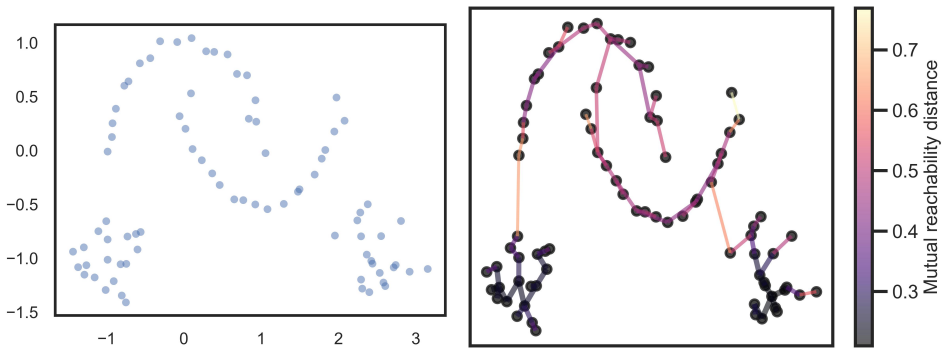
where $d(a, b)$ is the euclidean distance between a and b. According to this metric, points that are dense (with a low core distance) maintain their relative distance, while points that are more sparse are repelled and must remain a least their core distance away from any other point. The mutual reachability distance states that in order to connect two points together, they do not only have to be close in euclidean space, the two points also have to be dense. The mutual reachability distance is dependent on the parameter k, which means that a larger value of k causes more points becoming sparse.

Build a minimum spanning tree

After having defined the density of each point in the data set, the next stage is to find the islands of dense data. Density is relative, so it will vary across different islands. Conceptually, the data can be viewed as a weighted graph, where the vertices represent the data points and the edges represent the mutual reachability distance between pairs of points, with the weight of each edge being equal to this distance.

Consider a high threshold value that is gradually being decreased. As edges with weights above the threshold are eliminated, the graph will progressively break down into connected components. At different threshold levels, there will be a hierarchy of connected components ranging from fully connected to fully disconnected.

In order to obtain such a concept at a computational feasible cost, HDBSCAN creates a minimum spanning tree (MST) efficiently by Prim's algorithm [13]. This involves creating a weighted rooted tree by adding one edge at the time, always adding the lowest remaining edge that connects the tree to a vertex not yet included. Figure 2.4 shows the MST of a spatial data set, based on the mutual reachability distance of the data.



(a) Spatial data (b) Minimum Spanning Tree

Figure 2.4: Spatial data plot and corresponding MST based on mutual reachability distance.

Construct a cluster hierarchy

To obtain a hierarchy of connected components from the MST, a single linkage clustering [12] is performed. The essence of this stage is to sort the edges of the tree by distance in ascending order and iterate through, for each edge, create a new merged cluster. The result can be visualized as a dendogram shown in figure 2.5.

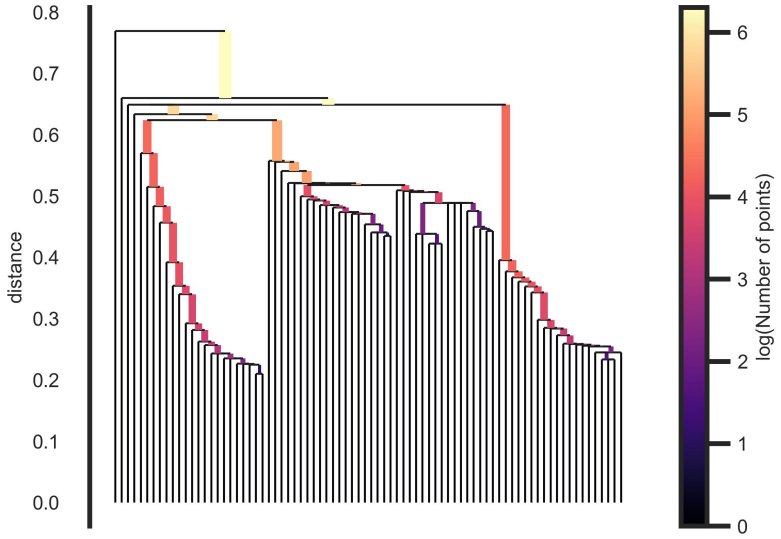


Figure 2.5: Dendogram

A flat cluster is traditionally obtained by drawing a horizontal line through the dendogram, selecting the clusters that the line cuts through. The level at which the line cuts the dendogram corresponds to a mutual reachability distance, i.e. a single density level that is fixed for all clusters. In practice, it would be more reasonable to have different density levels for each cluster, generalized as variable density clusters.

Condense the cluster hierarchy

The dendogram in figure 2.5 quickly becomes complicated and difficult to manage, especially when applied on large amounts of data. Thus, selecting variable level cuts through a dendogram becomes rather unpractical. As a next step in the procedure, HDBSCAN creates a condensed tree with more data attached to each node. The width of each column in the new tree represents the number of data points that is included in the cluster. The algorithm takes the parameter "minimum cluster size" as input. When a split in the dendogram corresponds to less than minimum cluster size number of points falling out of the cluster, the column in the condensed tree simply becomes narrower. On the other hand, when a split occurs where at least minimum cluster size number of points fall out of the cluster, a true split happens in the condensed tree. The result becomes more clear and easier to handle, visualized in figure 2.6.

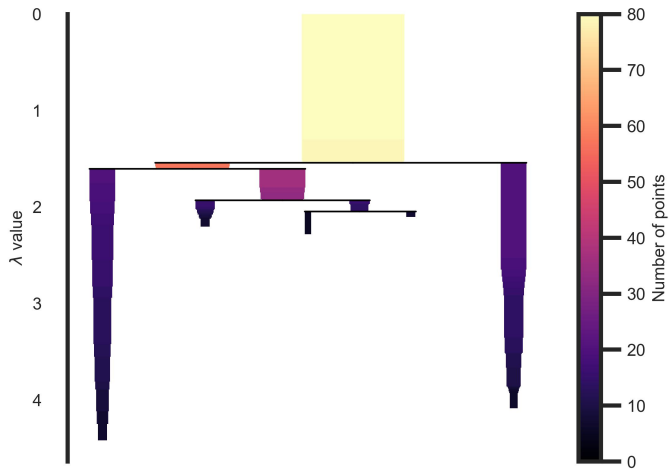


Figure 2.6: Condensed tree

Extract stable clusters

The remaining task is to select clusters for the final flat clustering. HDBSCAN picks the columns with the largest area, bound by the criteria that it is illegal to select a cluster that is a descendant of a cluster already selected. This is formalized as follows.

The persistence of a cluster is given by the value lambda defined as:

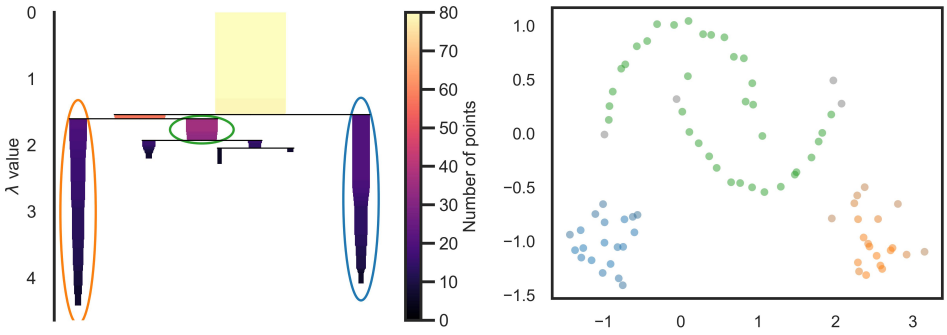
$$\lambda = \frac{1}{\text{distance}} \tag{2.82}$$

For each cluster, let λ_0 and λ_1 be the lambda value when the parent cluster divides and the cluster becomes its own cluster, and the lambda value when the cluster splits into smaller clusters. For each point in a given cluster, let $\lambda_p \in [\lambda_0, \lambda_1]$ be the value at which the point falls out of the cluster. The stability S of a cluster C_i is given as:

$$S(C_i) = \sum_{p \in C_i} (\lambda_p - \lambda_0) \tag{2.83}$$

In order to select a cluster, first declare all leaf nodes as clusters. Next, traverse the tree in reverse topological order, that is starting at the leaf nodes and move towards the root node. If the sum of the stabilities of the child clusters exceeds the stability of the current cluster, set the cluster's stability to be the sum of the child stabilities. Conversely, if the cluster's stability is higher than the sum of its children, declare it a selected cluster and deselect all of its descendants. Upon reaching the root node, the set of currently selected clusters is considered the flat clustering.

The selected clusters and the resulting labeled data set is visualized in figure 2.7.



(a) Selected clusters in a condensed tree (b) Clustered spatial data

Figure 2.7: Selected clusters in a condensed tree (a), and corresponding labeled data (b).

2.6 Automatic Operational Modal Analysis

The procedures outlined in this section is an assembly of the previous presented theory and is tested on a numerical example before applied on real measurement data from the Hålogaland bridge. A general description is given in this section and case specific details are given in section 3.1 and 4.1, respectively.

The automatic operational modal analysis can be broken down to three main steps for each time series under investigation:

1. OMA algorithm by Cov-SSI
2. Stabilization analysis
3. Clustering analysis by HDBSCAN

As convention, poles are all mathematical eigenvalues and modes are considered the true modes of the system. All calculations in this section and application in section 3.1 and 4.1 are conducted in Python version 3.10.8 [4], for details see Appendix.

2.6.1 Cov-SSI

Reference-based Covariance-driven Stochastic Subspace Identification, which underlying theory is described in section 2.4, is conducted using the STRID [2] package developed by Frøseth and Guddal. The implementation is based on Peeters and De Roeck [11] and Van Overschee and De Moor [16].

A two dimensional matrix of acceleration data with size `[number_of_channels x number_of_records]` and its sampling rate is passed along into the function for system identification. Reference indexes is also passed in, which is the index of the reference channels, corresponding to indices of the rows in the acceleration matrix.

Hyperparameters for the algorithm is the number of block rows i and the number of orders to perform the system identification for. Yang et al. [17] proposes as a rule of thumb, that the number of block rows i in the Block-Toeplitz matrix has a lower limit according to:

$$i \geq \frac{f_s}{f_{\min}} \tag{2.84}$$

where f_s is the sampling rate of the signal and $f_{\min}$ is the assumed lowest natural frequency of the system. The vibration mode of the lowest natural frequency of the Hålogaland bridge is approximately at 0.05 Hz, which yields $i \geq 40$ according to equation (2.84). The interpretation of this relation is that the block-Toeplitz matrix should be able to capture at least one period of the auto correlation function of the lowest order mode, i.e. the mode with the longest natural period. To ensure this is achieved, the number of block rows is set to $i = 50$. The number of orders to perform system identification for needs to be at least the number of DOFs of the system, hereby overestimating the order of the system is common. Still, too many orders will create more mathematical poles to assess subsequently and increase the computational time of the system identification itself. Since model order is generally unknown, common practice is to do a heavy overestimation.

Lastly, one has to decide on the length of the time series passed into the system identification. An underlying assumption for the theory is that the system is excited by a stationary, stochastic white noise loading process. Under this assumption, an infinitely long time series would be the optimal choice to make a better representation of the stochastic process. In practice this is impossible, but there appear to be other arguments for further limiting the time series. The main loading of the Hålogaland bridge is wind, which surely is not stationary. It turns out that slow changes from one type of weather to another can, by a long time series, be captured as very low frequent dynamics in the system. Hence, moving averages can appear in the recorded signal. The goal becomes to have a time series long enough to create a good statistical foundation for the system identification, but at the same time avoid change in weather systems to be captured. Previous experience and advice from supervisor suggest to have time series length between 10 and 30 minutes.

Cov-SSI establishes a large set of poles over the model orders which the algorithm is run for. That is, for each order, the solution to the system identification problem is found and poles are established from an eigenvalue decomposition of the state matrix $\mathbf{A}$ as in equation (2.65). The output can be visualized in a stabilization diagram shown in figure 2.8.

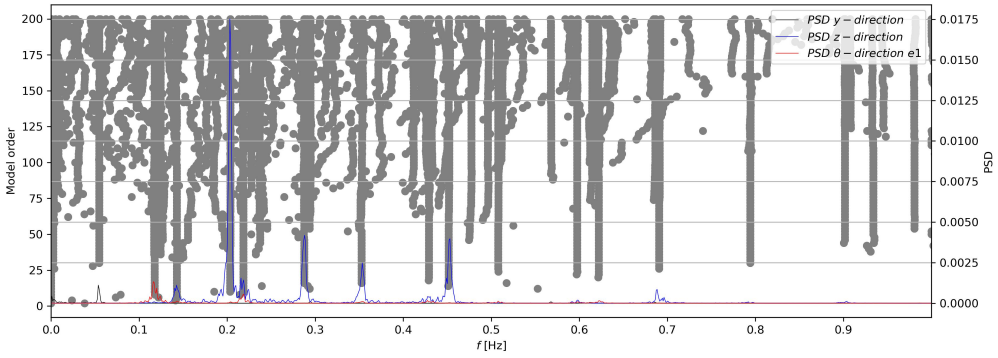


Figure 2.8: Stabilization diagram created by Cov-SSI

Two types of patterns are observable in the stabilization diagram. Clean vertical lines are assumed to represent physical system poles. Scattered and messy poles represents spurious mathematical poles.

2.6.2 Stabilization analysis

After Cov-SSI has established poles for various orders, a stabilization analysis is performed in order to reduce the number of spurious poles, thus cleaning the stabilization diagram. Stabilization analysis is conducted using the KOMA [3] package developed by Kvåle and the relevant function is based on Kvåle et al. [18] and Kvåle and Øiseth [19]. Poles obtained for a particular order are compared to the nearest pole from lower orders, using a modal indicator. For this study, the modal indicator is the relative difference in natural frequency. Afterwards, poles that meet the specified stability criterion are considered stable and preserved, while those that do not meet the criterion are discarded. Additionally, these criteria may also need to be met for a certain number of preceding orders s (referred to as the stability level) prior to the current order under investigation, thereby further promoting pole stability.

The stability criterion utilized in this study, is the relative difference in frequency, the relative difference in damping and the modal assurance criterion (MAC) [6], which is a measure of similarity between two mode shapes and is defined as:

$$\mathrm{MAC}_{i,j} = \frac{|\phi_i^T \phi_j^*|^2}{(\phi_i^T \phi_i^*)(\phi_j^T \phi_j^*)} \tag{2.85}$$

where ϕ is the eigenvector of a mode. It is defined such that MAC = 1 indicates two perfectly equal mode shapes, while MAC = 0 is as different as it gets, that is, the two mode shapes are orthogonal to each other. The values of the stability criterion for this study is presented in table 2.1.

Table 2.1: Stabilization criterion

Parameter	Value
Frequency	0.2
Damping	0.2
MAC	0.5
s	1

The result after the stabilization analysis is a more cleaned stabilization diagram shown in figure 2.9. Preserved poles are labeled in blue, while discarded poles are grey.

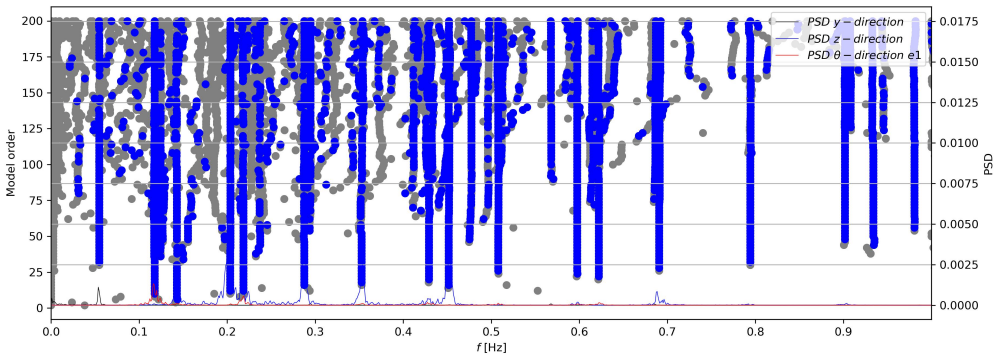


Figure 2.9: Stabilization diagram after stabilization analysis. Preserved poles are labeled in blue, while discarded poles are grey.

2.6.3 HDBSCAN

The remaining task of the modal analysis is the automatic selection of poles for modal feature extraction. The HDBSCAN algorithm, described in section 2.5.1, is utilized for the purpose, conducted by the KOMA package where relevant functions relies on the hdbscan library [20] for Python.

The HDBSCAN algorithm requires a spatial representation of the data to cluster. The poles created from Cov-SSI has three attributes; natural frequency, damping ratio and mode shapes. These attributes, or features, are utilized to design a spatial distance measure between the points. The distance d_{ij} between two points i and j is defined from paper of Kvåle and Øiseth [19] as:

$$d_{ij} = \sqrt{\alpha_\lambda^2 (d_{ij,Re(\lambda)}^2 + d_{ij,Im(\lambda)}^2) + (\alpha_m d_{ij,m})^2} + d_{ij,\infty} \tag{2.86}$$

where $d_{ij,Re(\lambda)}$ and $d_{ij,Im(\lambda)}$ is the difference between the real and imaginary part of the eigenvalues, respectively, and $d_{ij,m} = 1 - \text{MAC}(\phi_i, \phi_j)$. α_λ and α_m are scaling factors for eigenvalues and MAC respectively, enabling different weighting of the two parameters. $d_{ij,\infty}$ is the hard stop distance, which normally takes the value of zero, but is designed to have infinite value if two points have the same order o and therefore should not be clustered together. The construction of the distance matrix is also implemented in the KOMA package.

The most important parameters for HDBSCAN is minimum cluster size and minimum samlpes. Minimum cluster size is simply the smallest size grouping to be considered a cluster. Increasing this parameter will have the effect of increasing the cluster size and reducing the total number of clusters, merging some together. Minimum samples can be understood as a way to control how conservative the clustering is going to be. Increasing this parameter leads to more conservative clustering, where more poles are declared as noise and clusters are confined to increasingly denser regions. Both parameters are dependent on the size of the candidates for each cluster, which in a modal analysis context will be dependent on the number of orders the system identification is run for. In practice, the number of orders decides how tall the stabilization diagram will be, and consequently how many poles that may be stacked on top of each other. The latter decides the number of candidates for a cluster. Hence, the number of orders the system identification is run for should be taken in consideration when deciding minimum cluster size and minimum samples.

HDBSCAN provides a a probability p of each pole belonging to its designated cluster, simply a normalization of λ_p given in section 2.5.1. KOMA enables the user to set a probability threshold that the poles needs to surpass in order to belong to a cluster when extracting the clusters. The hyperparameters utilized for HDBSCAN is summarized in table 2.2.

Table 2.2: Hyperparameters HDBSCAN

Parameter	Value
Minimum cluster size	50^{*}, 25^{**}
Minimum samples	20^{*}, 10^{**}
Probability threshold	0.99
α_λ	1.0
α_m	1.0

* For Hålogaland bridge only.
** For numerical example only.

The output of HDBSCAN and the identified clusters among the stable poles are visualized in a stabilization diagram in figure 2.10, each cluster with its own color labeling. Each cluster represents a physical mode in the system and modal features (natural frequency, damping and mode shape) are extracted as the median value of the participants within a cluster.

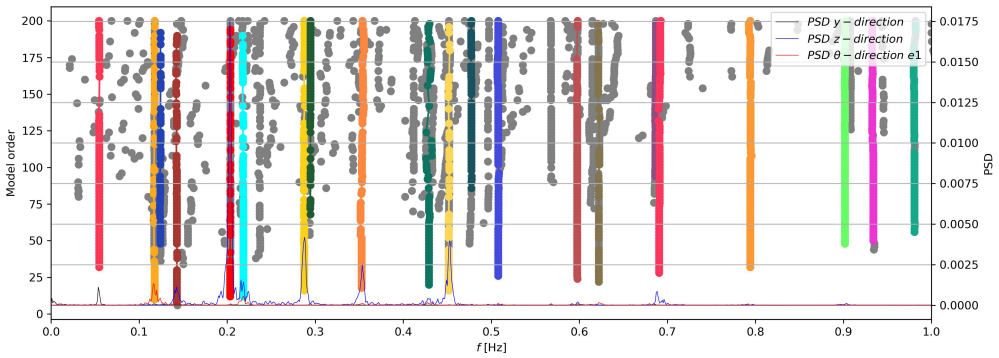


Figure 2.10: Stabilization diagram with a separate color for each cluster. Grey poles are remainders from the stabilization analysis considered noise by HDBSCAN.

Chapter 3

Numerical example

3.1 Method

In order to assure the accuracy and verify the framework for AOMA, a numerical example is carried out. The goal of the numerical example is not to mimic the dynamics of the Hålogaland bridge in order to find the best parameters for the algorithms, but rather to ensure that the framework for AOMA provides credible results in general. Random vibration data for a 9 storey planar share frame, visualized in figure 3.1, is generated with python code from the STRID package. The shear frame is designed to have vibration modes in the range 0-1 Hz, with the lowest order mode at approximately 0.08 Hz. 100 realizations are created in order to give a trace of the different modes. The AOMA procedure outlined in section 2.6 with the same hyperparameters, is carried out for each realization of vibration data to obtain modal features. The number of block rows is set to the same as the Hålogaland bridge, $i = 50$. The number of DOFs is severely reduced from the Hålogaland bridge, thus the number of orders for Cov-SSI to be performed for can also be reduced, and is set to 50. That is, system identification is run for every order from 1 to 50 inclusive. Length of the time series passed into Cov-SSI was set to 100 times the length of the longest natural period in order to meet the time series length indicated in section 2.6.1, which corresponds to approximately 20 minutes. An example of time series data from one channel of a typical realization is shown in figure 3.2.

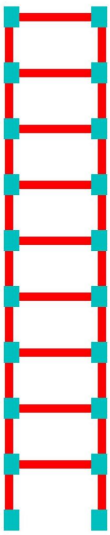


Figure 3.1: Illustration of a 9 storey planar shear frame.

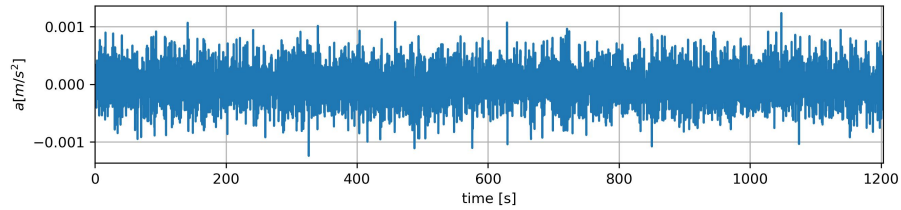


Figure 3.2: Acceleration data from one channel of a randomly picked realization.

In order to make traces from the result, a FEM model of the shear frame is used as a reference. For each realization, AOMA creates a set of modes, which may vary in size, which all are candidates to be assigned to the reference modes. In order to assign an estimated mode to to a reference mode, a simple comparison routine is carried out. The routine includes finding the estimated mode that is closest to the reference mode in the frequency domain, and the estimated mode has to satisfy a certain tolerance as well to be assigned. The tolerance is the difference in frequency Δf between the reference mode and the estimated mode, and is set to $\Delta f = 0.025$ Hz. After the comparison routine is carried out, there may still exist estimated modes that is not assigned to a reference mode because they are not similar enough and interpreted as false detections.

3.2 Results

Results for the numerical example is presented in this section. Figure 3.3 displays the trace of modes detected by AOMA. Table 3.1 compares the results from FEM to AOMA. Frequencies from the estimated modes from AOMA is extracted as the mean frequency of the trace. In total, there were 868 modes estimated by AOMA and by criteria presented in section 3.1, 862 of the estimated modes were matched to a reference mode from FEM, that is 99.3% of the estimated modes were correctly estimated by AOMA and it achieved a 96.4% overall detection rate. During the 100 analyzed time series, there was an average of 8.7 estimated modes for each analysis, with a standard deviation of 0.55.

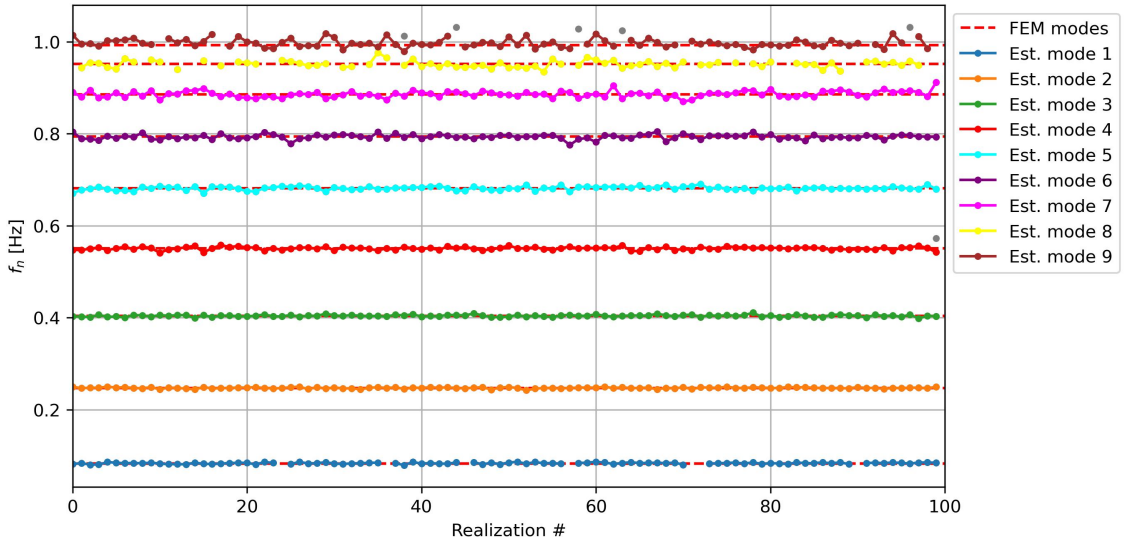


Figure 3.3: Mode trace of a numerical example. Noise modes labeled in grey.

Table 3.1: Results from the numerical example.

Mode #	f_{FEM} [Hz]	$\overline{f}_{\text{AOMA}}$ [Hz]	$\Delta f_{\text{rel}} = \frac{\lvert f_{\text{FEM}} - \overline{f}_{\text{AOMA}} \rvert}{f_{\text{FEM}}} \cdot 100[\%]$	Detection rate [%]
1	0.083	0.083	0.096	93
2	0.247	0.247	0.020	100
3	0.404	0.404	0.010	100
4	0.551	0.550	0.011	100
5	0.682	0.681	0.035	100
6	0.794	0.794	0.020	100
7	0.885	0.885	0.016	100
8	0.952	0.952	0.033	78
9	0.993	0.993	0.048	91

3.3 Discussion

On average, the algorithm for AOMA underestimates the number of modes for the analyzed time series. Nevertheless, the standard deviation is low and several of the mode traces have a 100% detection rate. Where discrepancies exists, they are not large and relative difference in frequency between estimates and references never exceeds 0.1%. The framework for AOMA gives overall good results on the numerical example, which proves a good foundation for application on experimental data from the Hålogaland bridge.

When performing AOMA, it is desirable to averagely hit the number of modes in the system with as low standard deviation as possible. Hyperparameter selection for AOMA algorithms still remains a big challenge and has to be carefully adapted to the specific application. Tuning the parameters further for the shear frame, in order to approve accuracy, would most likely not pay dividends on the application of the Hålogaland bridge. The shear frame can not mimic the dynamics of the bridge anyways. Hence, optimizing the numerical example is out of the scope of this study and the example is utilized for verification purposes, only.

There are several factors contributing to the expectation that the algorithm will not perform as good on the experimental data from the Hålogaland bridge. Firstly, the loading of the shear frame is stationary white noise loading. This will, as previously indicated, not be the case for the Hålogaland bridge. The frequency content of the loading will not be evenly distributed through the domain, hence not perfectly white noise. The loading will neither be stationary as the wind experiences moving averages, observable in figure 4.26. Furthermore, the Hålogaland bridge is a much more complex structure than the planar shear frame with a great number of modes, some easily detected, some not. This makes the task of AOMA much harder since the modes may be particular difficult to distinguish from each other. Sensor recording, drift in local timestamp and GPS systems are all sources that may introduce errors in an operational application of modal analysis.

Chapter 4

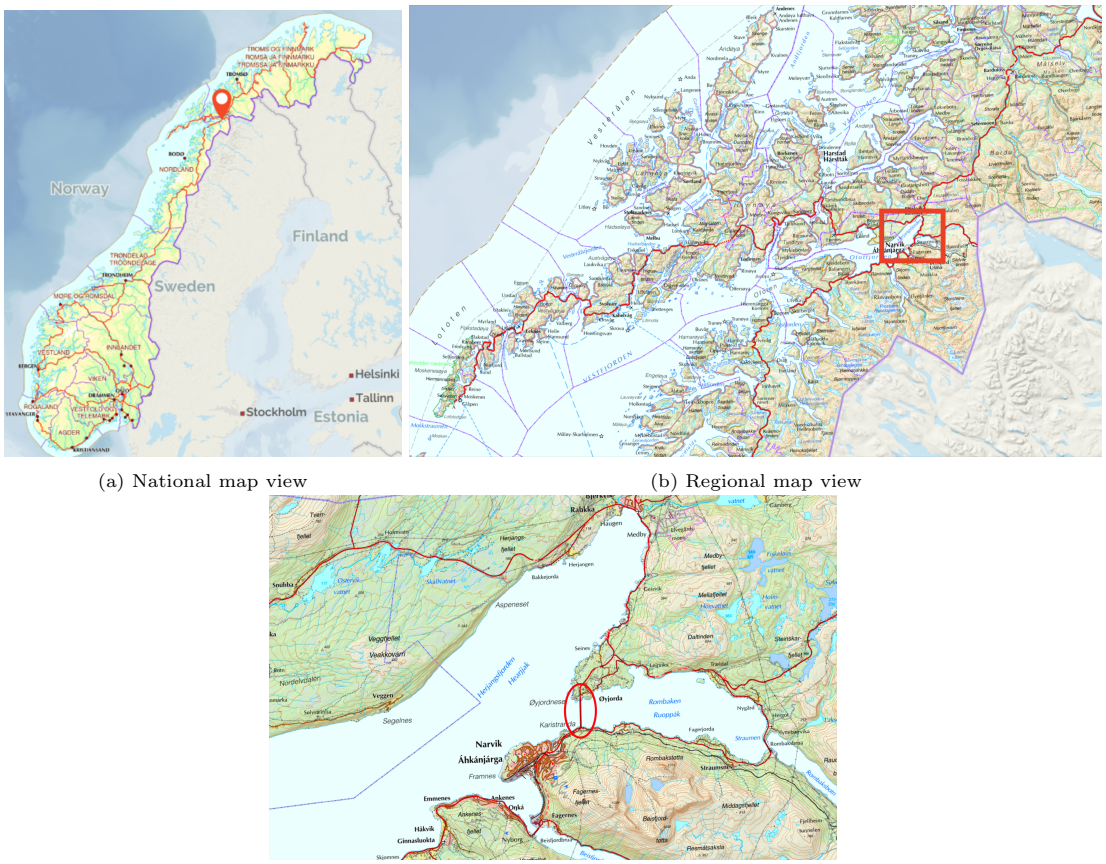
The Hålogaland bridge

4.1 Method

The Hålogaland bridge is a suspension bridge located 6 kilometers outside Narvik in norhtern Norway. The bridge is part of the E6 road and crosses the Rombak fjord, which is a part of the Ofotfjord. It goes from Karistranda in the south to Øyjord in the north. Figure 4.1 shows a picture of the bridge seen from Øyjord and figure 4.2 shows the bridge location on the map.

Figure 4.1: Picture of the Hålogaland bridge.

Source: Øyvind Wiig Petersen, ©NTNU



(a) National map view

(b) Regional map view

(c) Local map view

Figure 4.2: Location of Hålogaland bridge in national, regional and local map view respectively.

Source: ©norgeskart.no

The construction started in 2013 and was completed in December 2018. It was part of a larger infrastructure project at a cost of 3,8 billion NOK according to Norwegian Road Administration [21]. The construction of the bridge made the E6 road 18 kilometers shorter and reduced travel time by 15-20 minutes. The arrival of the bridge also made traveling safer, because it circumvents the old avalanche prone road. The Hålogaland bridge has a main span of 1145 meters, making it the second longest suspension bridge in Norway. Two reinforced concrete towers at approximately 180 meters height is supporting the two air-spun steel main cables, which again has 110 locked coil steel hangers vertically connected to them. The hangers carry the bridge deck, which is a 3 meters high and 18.6 meters wide steel box girder.

The slender design of the bridge, resulting from its long span and relatively small cross section, causes it to exhibit low natural frequencies. Additionally, the bridge is susceptible to wind loading, which primarily occurs within the same frequency range. Consequently, the bridge is particularly subjected to wind-induced response. To enhance knowledge of the impact of environmental loading on the behavior of long-span suspension bridges, a monitoring system has been implemented on the Hålogaland Bridge.

4.1.1 Monitoring system

The monitoring system of the Hålogaland bridge was in operation from May 2021. The system is comprehensive with ability to measure structural response of the deck, wind, temperature and vibrations of the hangers. Structural response measurements includes acceleration data measured by 22 tri-axial force balance accelerometers and strain measured by 36 strain gauges. Wind is measured by 10 sonic anemometers, temperature by 22 thermometers and vortex-induced vibration on hangers are measured by three piezoelectric accelerometers. The monitoring system is equipped with 11 logger boxes in total, one in top of each tower and the remaining 9 are distributed along the bridge span. All previously mentioned sensors are wired to the closest logger box. The main hardware unit of the logger is a NI CompactRIO controller with custom software responsible for sampling and filtering the measured data before it is saved locally on hard drives. Additionally each controller has a Trimble Bullet GPS antenna connected for accurate time stamping of recorded data. The controller has internet connection, so it is able to push the data to a server at regular intervals. The locations of the different sensors and components of the monitoring system relative to the geometry of the bridge is illustrated in figure 4.3.

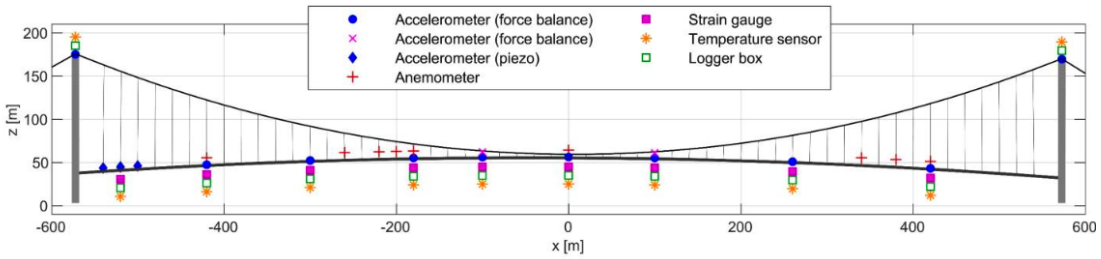


Figure 4.3: Monitoring system of the Hålogaland Bridge

Source: Petersen et al. [22]

The main objective of this study is to perform AOMA on acceleration data, hereby the data from the force-balance accelerometers are studied. There is one accelerometer in top of each tower, 16 sensors located inside the girder and 4 sensors on the hangers. The logger boxes are named with numbers in chronological order from left to right relative to figure 4.3. The sensors connected to a given logger box inherits the number of the box in addition to its own labeling. The accelerometers inside the girder are connected to logger boxes 3-10, and there are two sensors connected to each box, one at each lateral end respectively, to be able to capture torsional motion. Location of accelerometers in the girder along the bridge span is illustrated in figure 4.4 and associated coordinates of the sensors are found in table 4.1.

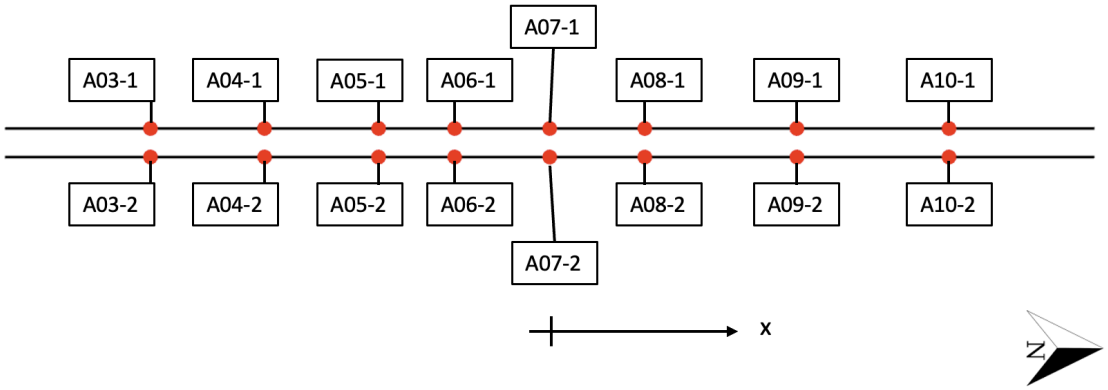


Figure 4.4: Overview of accelerometers in the bridge deck.

Table 4.1: Longitudinal coordinates of acceleration sensor location along the bridge span.

Sensor name	A03-1/ A03-2	A04-1/ A04-2	A05-1/ A05-2	A06-1/ A06-2	A07-1/ A07-2	A08-1/ A08-2	A09-1/ A09-2	A10-1/ A10-2
x-coordinate [m]	-420	-300	-180	-100	0	100	260	420

This study has chosen to focus of the motion on the bridge deck only, therefore data from the sensors in figure 4.4 and table 4.1 are the ones considered in the AOMA.

A research question of interest is how the identification of modal parameters is affected by environmental factors, namely wind and temperature [22]. For that purpose, one temperature sensor in the mid span of the bridge is selected for calculation of temperature statistics. Similarly, one anemometer in the mid span of the bridge is selected for wind statistics.

For a complete description of the monitoring system, the reader is referred to Petersen et al. [22].

4.1.2 Finite element model

A finite element (FE) model of the bridge made in Abaqus [23], and exported modal parameters, has been provided by Øyvind Wiig Petersen as a odb file and a h5 file, respectively. The two resources are used in combination to obtain the relevant modal parameters for the bridge deck, utilized for comparison with the AOMA. Only a short description of the model based on Solstad and Onstad [1] is given here.

The FE-model consists of the main parts of the bridge, namely; two towers, two main cables, hangers and bridge deck, shown in figure 4.5. The model geometry is based on as-built geometry and the main parts are modelled using beam elements. Consequently, three-dimensional parts are modelled as one-dimensional. Since the cross section of all parts are small compared to the global dimensions along the beam axes, the one-dimensional simplification is assumed to be a sufficient approximation. B31 and B32 elements are utilized, which are 2-node linear and 3-node quadratic beam elements in space. All DOFs, that is, 3 translational and 3 rotational DOFs for each node, are active for these beam elements. A lumped mass formulation is used for dynamic calculations.

Modal analysis by Lanczos eigensolver, which means that Abaqus solves the eigenvalue problem by equation (2.19), was already carried out in the provided odb file and the results was ready for inspection.

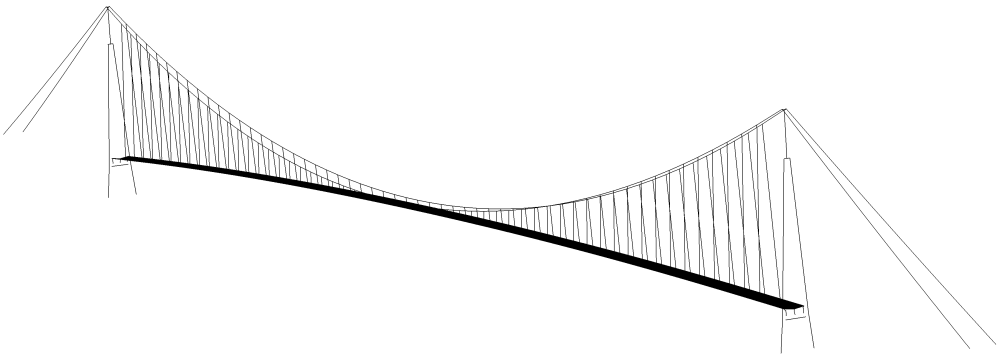
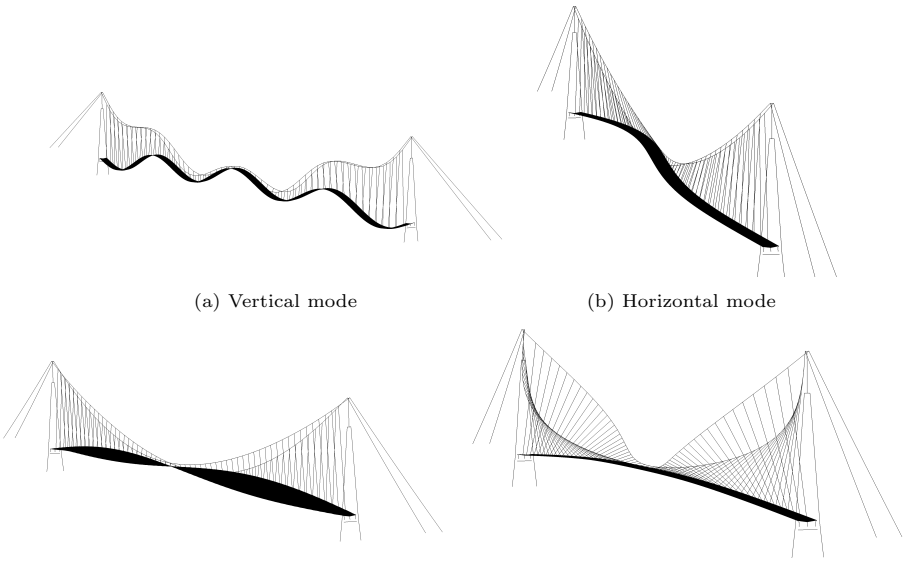


Figure 4.5: A FE-model of the Hålogaland Bridge from Abaqus.

The odb file contains all global modes for the FE-model of the Hålogaland bridge. Since this study focuses on modal analysis of the bridge deck, it is desirable to select the bridge deck modes from the FE-model for comparison with the AOMA. By visual inspection and animation in Abaqus it is possible to distinguish different mode types from each other, visualized in figure 4.6. Horizontal-, vertical- and torsional bridge deck modes are selected, while cable modes are generally discarded. When analysing the results of the AOMA, evidence of additional modes was apparent beyond those already selected from Abaqus. Therefore, cable modes was reviewed in order to find references of the additional modes and three cable modes were selected. After identifying the relevant modes, the modal parameters of these modes was accessed from the h5 file in Python. Natural frequency and mode shapes were extracted. Note that for comparison purposes, only nodes from the FE-model that corresponds to sensor location on the bridge deck was extracted in order to obtain eigenvectors of the same size, hence making the mode shapes comparable. For visualization purposes the complete eigenvectors including all bridge deck nodes in the FE-model was utilized. Results are presented in section 4.2.1.



(a) Vertical mode (b) Horizontal mode

(c) Torsional mode (c) Cable mode

Figure 4.6: Different mode types identified by visual inspection in Abaqus.

4.1.3 Data exploration and processing

The recorded data is stored in a TDMS file format, one file for each 8 hour period of data. Data in a TDMS file is slowly read to memory and significant amounts of data is required for long term analysis. In addition, time synchronization of data from different channels are also required. Time synchronized data stored in a HDF5 file format to use for this study was provided by Ole Andre Øiseth, from the Department of Structural Engineering at NTNU.

HDF5 is a hierarchical data format for storing and managing large and complex data [24]. It supports n-dimensional datasets and each element may itself be a complex object. There are no limitations on the HDF5 file size giving great flexibility for big data. It can be run on distributed systems and provides high level APIs for multiple platforms. The data is self contained, meaning metadata is kept with the data and a single HDF5 file gives context to itself. HDF5 supports high performance, parallel I/O, making it powerful and robust for big data analysis.

There are three main levels in the hierarchy of a HDF5 file, starting with the file itself. Secondly there are groups, which are folder-like containers, which may contain subgroups. Lastly there are datasets, which are array-like collections of data that is contained within the groups. One of the main advantages of the HDF5 format is that it enables a gateway where the user can request and read specific parts of the file, without requiring all the data of the file to be loaded to volatile memory, which can easily be exceeded when handling large amounts of data. The HDF5 file from NTNU contains one month of data recorded from the monitoring system and an illustration of the relevant parts of the hierarchy is shown in figure 4.7. Labeling of periods and sensors are just to exemplify.

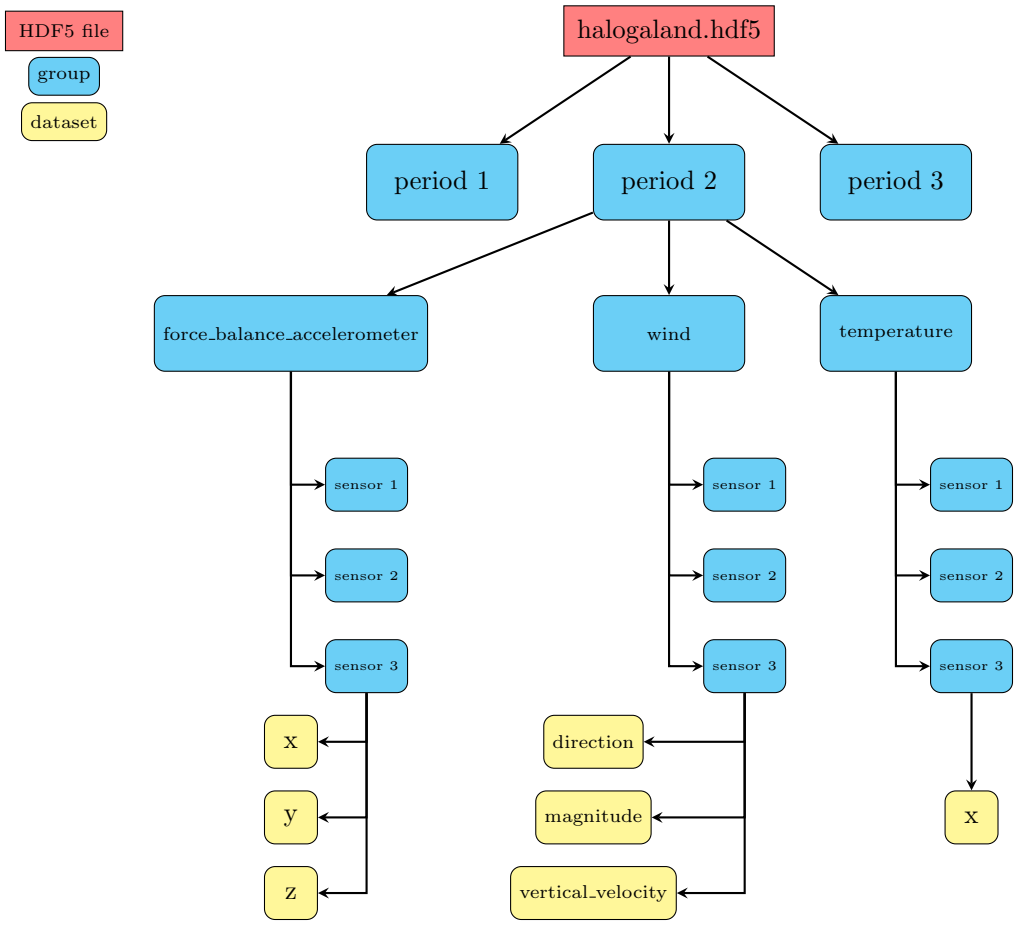


Figure 4.7: Hierarchical overview of structure of a hdf5 file.

The first group in the hierarchy is periods of 8 hours of data, each labeled `[YYYY-MM-DD-hh-mm-ss]`. The next subgroup is the sensor type, which again contains a subgroup of the sensors distributed on the bridge. Sensor labeling is according to procedure described earlier with inheriting the number of the closest logger box in addition to own number. Each sensor contains channels of recorded data, which corresponds to the dataset. For force-balance accelerometers, all channels from the sensors shown in figure 4.4 are utilized. Magnitude and direction is picked for a wind sensor in the mid span of the bridge, the same applies for the temperature which only has one channel.

Further discussion on data exploration and processing applies to the acceleration data only, which is the data used in the AOMA procedure. The HDF5 file is accessed and read to Python through the h5py [25] library.

The acceleration data from the HDF5 file is provided at a 16 Hz sampling rate. A typical 30 minute time series from the x channel of the A03-1 sensor is shown in figure 4.8. The first step in the data exploration is to investigate the frequency content of the signal, which will affect parameter selection at later stages. For this purpose, the PSD of the signal for each channel respectively is obtained by Welch's method. An arbitrary 8 hour period is chosen, then the PSD for each sensor in x, y and z direction respectively is computed. The average PSD in each direction is shown in figure 4.9 as solid lines, min and max respectively is shown as transparent lines.

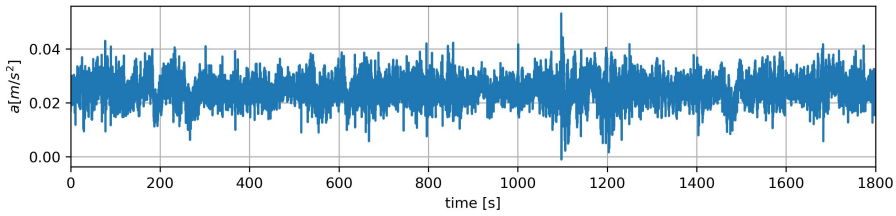


Figure 4.8: A typical 30 minute time series from the x channel of the A03-1 sensor, starting mid night of February the 4^{th}, 2022.

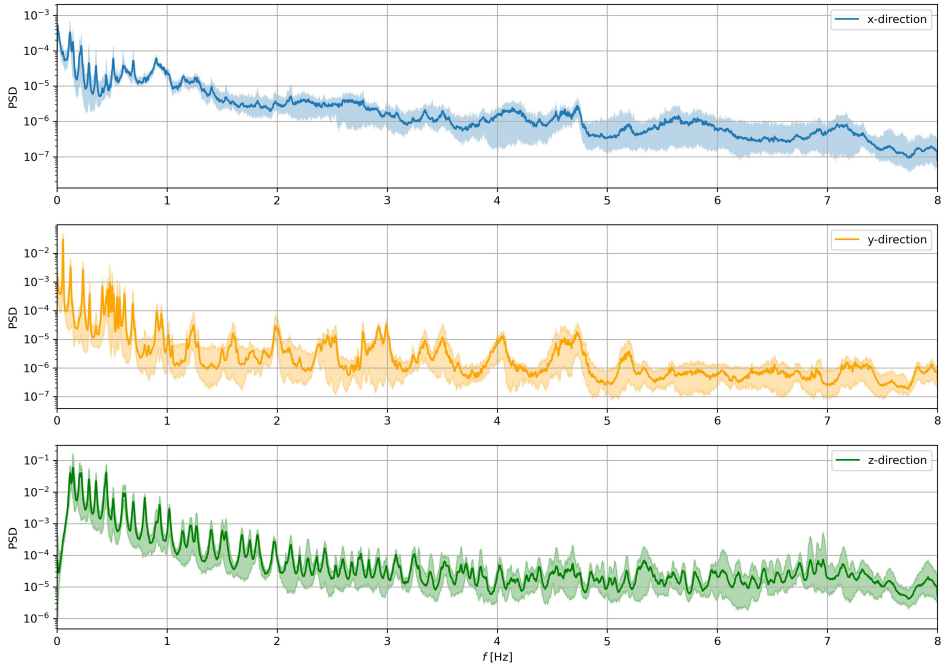


Figure 4.9: PSD by Welch's method of 16 Hz acceleration data in x, y and z direction respectively. Filled line represents mean PSD off all channels in given direction. Transparent area represents max and min PSD, respectively.

From figure 4.9 one can see that the majority of the energy in the system is situated in the range from 0 to 1 Hz. Hence, this becomes the main frequency range of interest. This finding offers the opportunity to down-sample the signal in order to reduce the amount of recordings and hereby reduce computational cost. The Nyquist rule described by equation (2.16) requires the new signal to have a sampling rate of 2 Hz. Before downsampling is performed, a low pass filter of the type Butterworth [7] by equation 2.15 is applied with filter order $n = 10$ and cutoff frequency $f_c = 1$ Hz. Section 2.3.3 stated that a fundamental assumption for the discrete-time stochastic state space model is that the process noise $\mathbf{w}_k$ and measurement noise $\mathbf{v}_k$ are zero mean Gaussian processes, which implies that the output of the system $\mathbf{y}_k$ also becomes a zero mean Gaussian process. From figure 4.8 it is clearly vissible that the output response from the Hålogaland bridge does not have a zero mean. In order to obtain a zero mean process the, the mean of each 8 hour period is subtracted from the signal. All these procedures were implemented in the loading process of the data from the HDF5 file in a custom dataloader, for details see Appendix.

4.1.4 Application of AOMA

After the acceleration data has been loaded and pre-processed, it is structured as a two dimensional matrix with size `[number_of_channels x number_of_records]`. The 8 hour period in the HDF5 file is dived into time series of 30 minutes each, yielding the number of records to be 30min $\times$ 60sek $\times$ 2Hz = 3600. There are 16 sensors located in the bridge deck, each with channels in x, y and z direction respectively, leading to 48 channels in total. Reference channels are picked among the sensors on the bridge deck that are assumed to capture global mode shapes of the bridge. The reference channels were chosen to be y- and z-channels of sensors at one side of the bridge, specifically the west side of the bridge, giving a total of 16 reference channels. The stages of the AOMA outlined in section 2.6 is then carried out. The process is repeated for each time series within each 8 hour period and results are written to file for each time series under investigation. There are a total of 73 periods of 8 hours within February 2022 on which analysis has been conducted, yielding a total of 1168 analyzed time series. Criteria for a given period to undergo analysis is that all relevant channels are present in the logged data.

The number of DOFs for the Hålogaland bridge is unknown in a system identification context. Previously studies [18] [19] [1] [26] with application of both OMA and AOMA utilizes a maximum modal order between 200 and 250. Based on this experience, the number of orders to perform system identification has for this study been chosen to 200, with a step of two. That is, system identification is run for every other order from 2 to 200 inclusively. This yields the potential of twice the amount of poles as the numerical example in section 3.1. This is the reason why "minimum cluster size" and "minimum samples" is twice as big for the Hålogaland bridge as the numerical example in table 2.2.

The results from the AOMA gives a large set of estimated modes over the frequency range of interest, but the number of estimated modes for each time series may vary. It is desirable to investigate how the modal parameters of a given mode develops over time, to potentially see the effects of environmental factors. Modal tracking algorithms can be used for such a purpose, but is not within the objective of this study. This study uses the modes obtained from the FE-model as reference modes. In order to obtain a trace of each mode, a comparison and sorting routine has been carried out that is further described here. For each time series analyzed, AOMA has created a set of estimated modes, which is referred to as candidates. For each reference mode, find the candidate that is most similar to the reference mode and in addition satisfy a tolerance in difference from the reference mode. There will be at most one candidate matched to each reference mode per time series. All the modes estimated by AOMA that is matched to the same reference mode over the total analysed period, constitutes the mode trace. For similarity and difference measurement, relative difference in frequency δf_i and difference in mode shape $d_{\text{ref}i,m} = 1 - \text{MAC}(\phi_{\text{ref}}, \phi_i)$ is used. For the latter, ϕ_{ref} is the mode shape of the reference mode and ϕ_i is the mode shape of the candidate mode. Relative difference in frequency δf_i is defined as:

$$\delta f_i = \frac{|f_{\text{ref}} - f_i|}{\max(f_{\text{ref}}, f_i)} \tag{4.1}$$

where f_{ref} is the natural frequency of the reference mode and f_i is the natural frequency of the candidate mode. Tolerances were set to $\delta f_i = 0.1$ and $d_{\text{ref}i,m} = 0.2$, relatively large tolerances with the objective to not discard too many candidates. Naturally, some traces have a good detection rate, while other have worse. Results are presented in chapter 4.2.

In order to obtain the mode shapes of the bridge girder in the horizontal, vertical, and torsional directions, the following matrix is utilized to convert the horizontal and vertical mode shape values on either side of the bridge into corresponding horizontal, vertical, and torsional values at the midspan. The transformation matrix is based on Solstad and Onstad [1] and is defined as:

$$\begin{bmatrix} y \\ z \\ \theta \end{bmatrix} = \begin{bmatrix} \frac{1}{2} & \frac{1}{2} & 0 & 0 \\ 0 & 0 & \frac{1}{2} & \frac{1}{2} \\ 0 & 0 & \frac{-1}{B} & \frac{-1}{B} \end{bmatrix} \begin{bmatrix} y_1 \\ y_2 \\ z_1 \\ z_2 \end{bmatrix} \tag{4.2}$$

where y, z and θ are the new mode shape values at the midspan of the bridge, B is the width of the bridge and $y_{1,2}$ and $z_{1,2}$ are the initial mode shape values of each side of the girder.

Because of damping in the structure, the mode shape components are represented as complex numbers, introducing a phase shift between them. Consequently, the maximum amplitudes of these components do not align simultaneously [18]. To address this, a function from the KOMA package is employed to obtain the modes shapes, where the mode shape vectors are rotated in the complex plane such that the absolute value of the real part is maximized. Finally, the mode shapes are plotted as the real parts of the components, one for each mode in the trace. Results are presented in section 4.2.2.

4.2 Results

4.2.1 Results from FEM

The horizontal, torsional and vertical bridge deck modes found from the FE-model in Abaqus, with natural frequency lower than 1 Hz, are presented in figure 4.10, 4.11 and 4.12, respectively. Figure 4.13 shows three selected cable modes assumed to appear in the modal analysis. There are a total of 24 relevant modes selected as references from Abaqus.

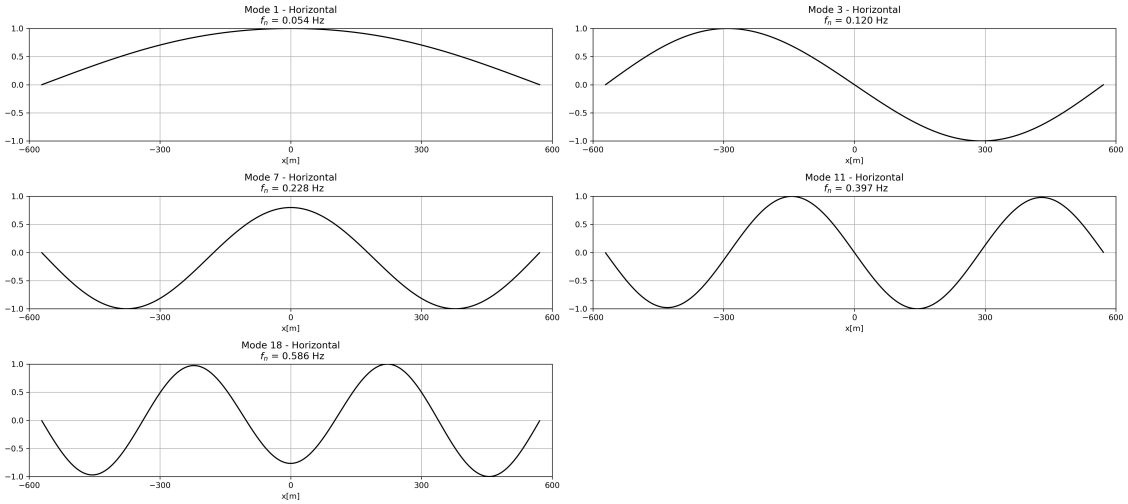


Figure 4.10: Mode shapes of horizontal modes from the FE-model, with corresponding natural frequencies.

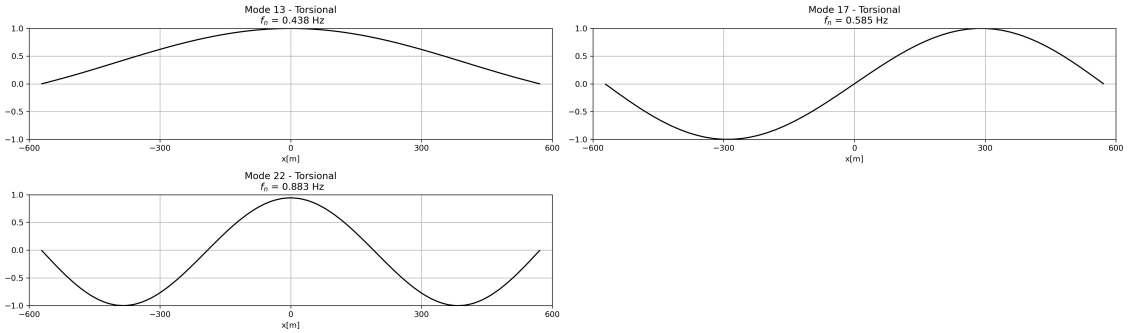


Figure 4.11: Mode shapes of torsional modes from the FE-model, with corresponding natural frequencies.

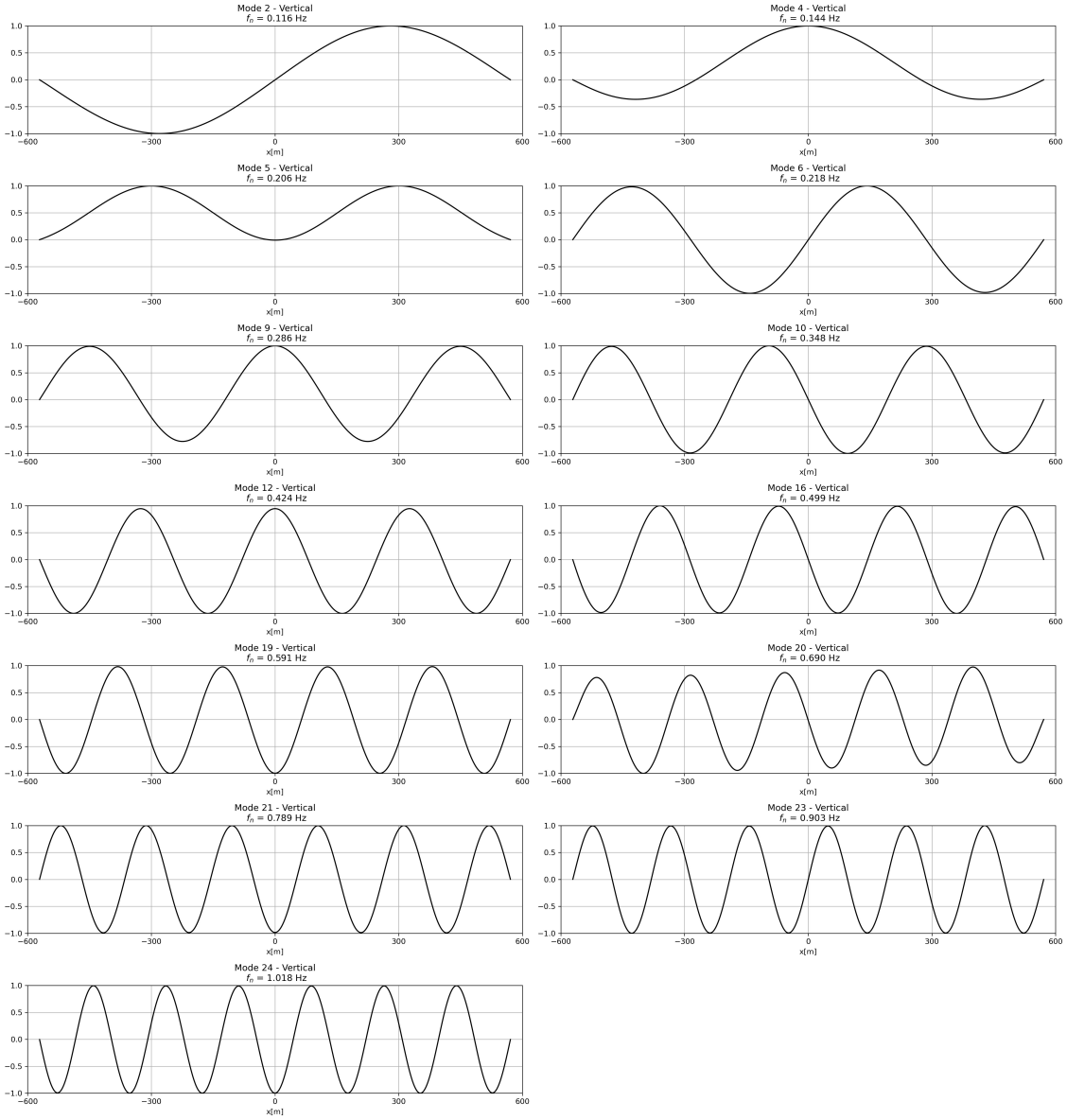


Figure 4.12: Mode shapes of vertical modes from the FE-model, with corresponding natural frequencies.

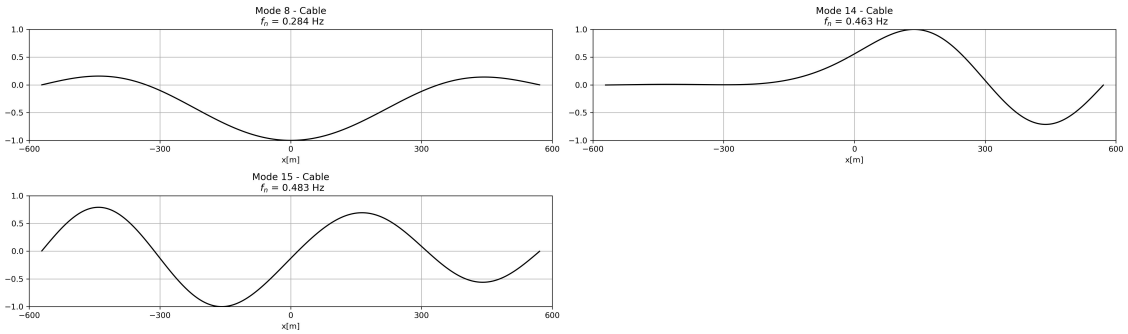


Figure 4.13: Mode shapes of three cable modes from the FE-model, with corresponding natural frequencies. The figure shows the horizontal component of the bridge deck, excited by the cable modes.

4.2.2 Results for AOMA

The results for the AOMA is presented in this section, along with mode traces found by reference to FEM results. Figure 4.14 displays all modes estimated by AOMA as blue dots, plotted by frequency and time series number. Frequencies from Abaqus is plotted as red, horizontal dashed lines. During the 1168 analyzed time series, there was an average of 20.7 estimated modes for each analysis, with a standard deviation of 2.04. The smallest number of estimated modes from an analysis was 14 and the largest number was 33. The distribution of the number of estimated modes for each time series is visualized in figure 4.15 as a histogram. The average execution time for analyzing a time series was 4.9 seconds.

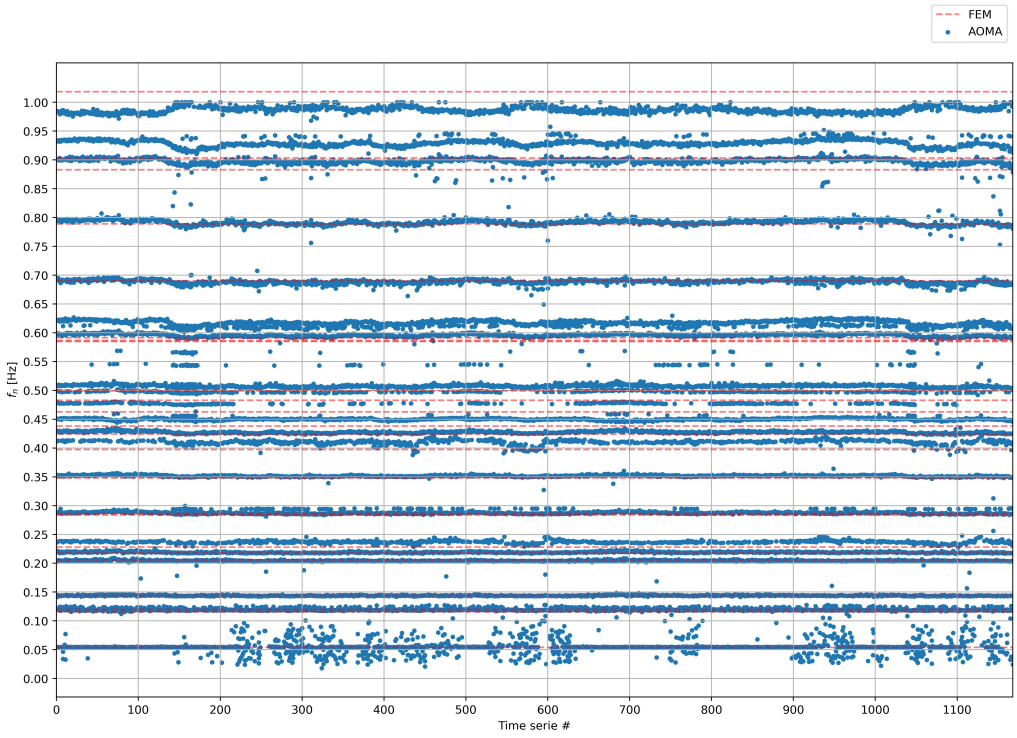


Figure 4.14: All estimated modes from AOMA, plotted by its frequency vs. time. Frequencies from Abaqus model is plotted as horizontal dashed lines, labeled FEM.

The total number of estimated modes over the whole period, is 24 126. Figure 4.16 displays modes estimated by AOMA after mode traces are found by reference to FEM results. Modes matched to references are labeled blue, while un-matched modes are labeled orange. The number of matched modes is 22 478, while the number of un-matched modes is 1 648, that is 93.2% of the estimated modes were matched to a reference mode from FEM, achieving a 80.2% overall detection rate. All estimated modes matched to the same reference, constitutes the trace of the mode. Each mode trace with its own color labeling is displayed in figure 4.17.

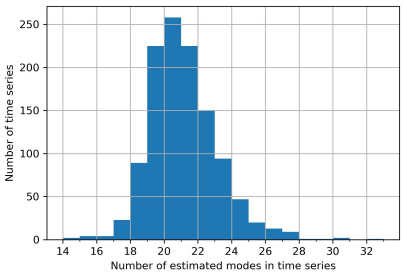


Figure 4.15: Distribution of number of estimated modes for each time series.

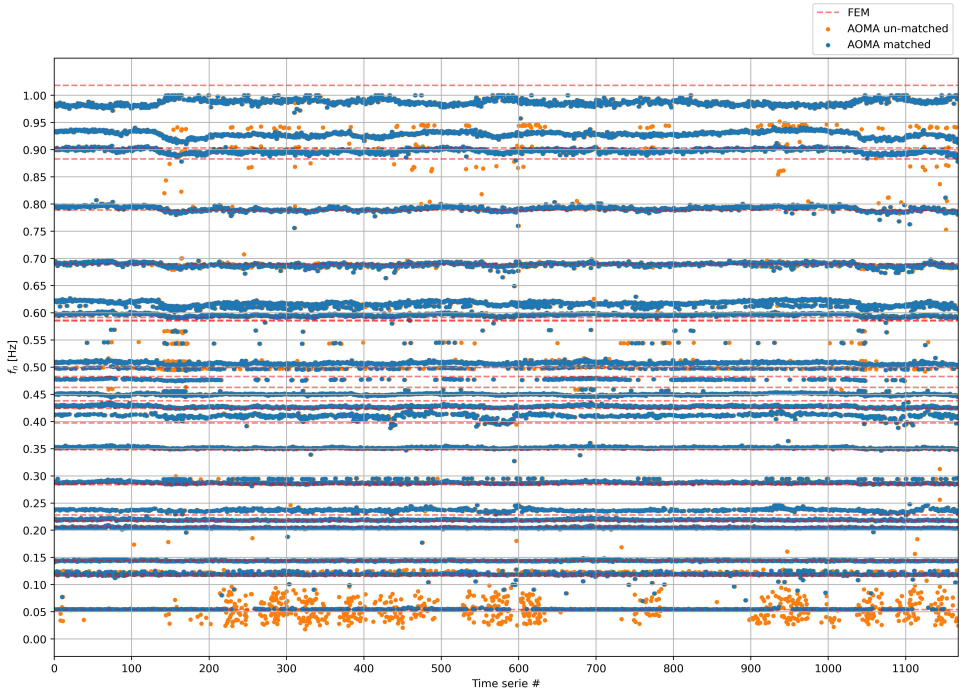


Figure 4.16: All estimated modes from AOMA, plotted by its frequency vs time. Modes matched to a reference mode from Abaqus is labeled blue, while un-matched modes are orange. Frequencies from Abaqus model is plotted as horizontal dashed lines, labeled FEM.

Observe in figure 4.16 the occasional appearance of orange scatter for the lowest order mode trace at approximately 0.05 Hz. Because of this finding specifically, a thorough investigation of environmental factors influence on identification of mode 1 will be carried out in section 4.2.4.

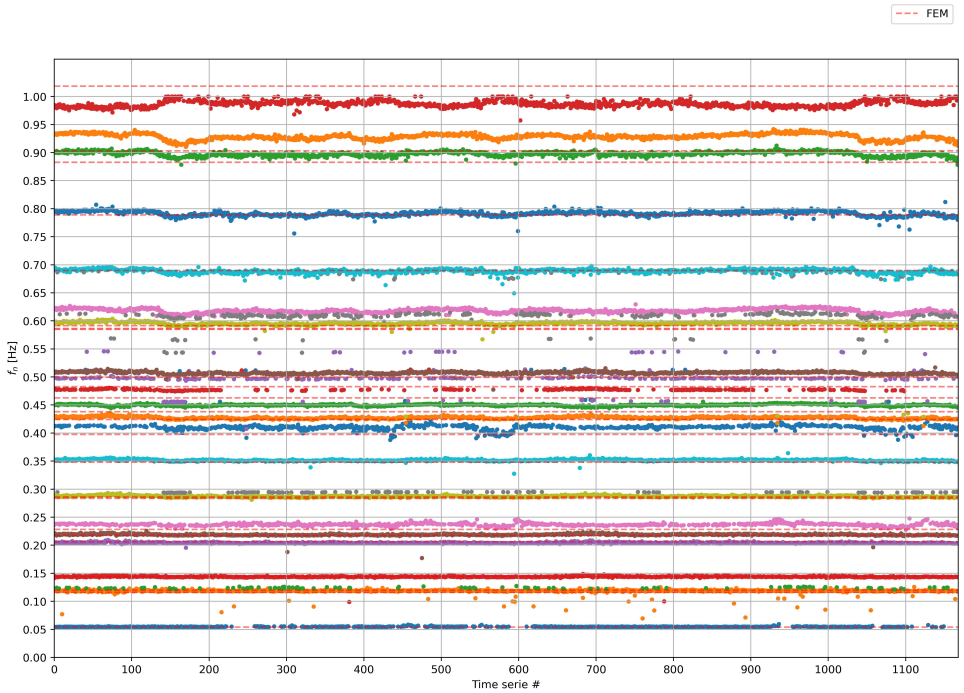


Figure 4.17: Modes assigned to a reference mode from Abaqus, plotted by its frequency vs. time. All modes within each color belongs to the same trace. Frequencies from Abaqus model is plotted as horizontal dashed lines, labeled FEM.

Figure 4.18 displays the frequency distributions within each mode trace. Here ”n” is the number of detections within each trace and ”r” is the detection rate. Both varies greatly over the different mode traces, from several traces accomplishing a detection rate of 100%, to mode 3, achieving a detection rate of only 9.7%. Comparing the average detection rate across the different mode types reveals a clear pattern. The vertical modes are the most easily detectable with a average detection rate of 99.7% and the torsional comes second with an average 98.6%. Then there is a huge gap down to the horizontal modes with an average of 50.8%. Not surprisingly, the cable modes are the most difficult to detect with an average detection rate at 26.2%. ”Head and shoulder” pattern in the histogram, clearly visible for mode 15 and 18, may indicate that some modes has been wrongly matched to the reference mode and should belong somewhere else (either another reference mode or a mode not considered in the analysis). Further findings from figure 4.18 will be discussed section 4.3.

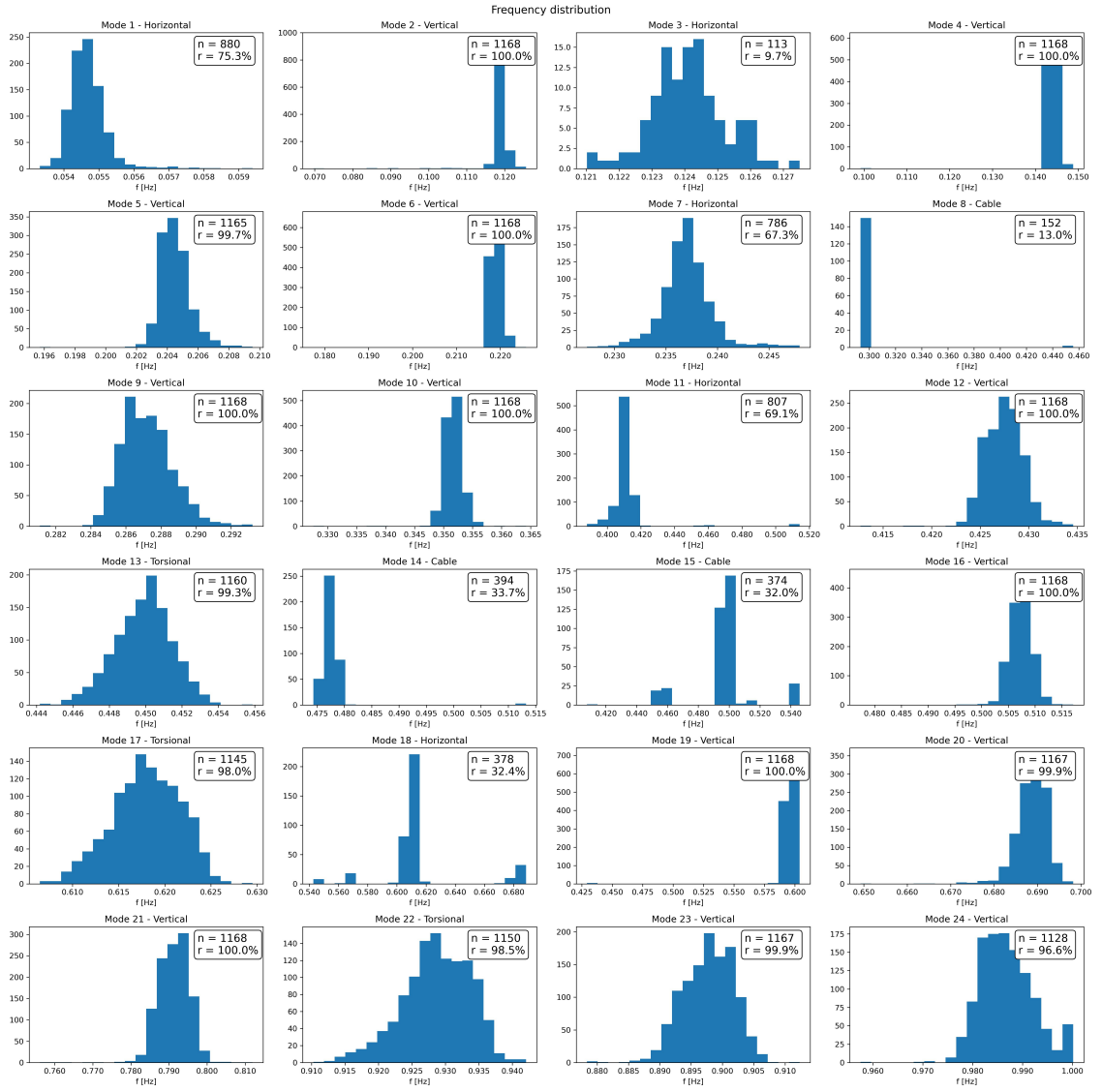


Figure 4.18: Frequency distributions among assigned modes. Here ”n” is the number of detections and ”r” is the detection rate.

Mode shapes with corresponding mean frequency and mean damping for all horizontal, torsional, vertical and cable modes found by AOMA are visualized in figure 4.19, 4.20, 4.21 and 4.22, respectively. Mode shapes of reference modes from the FE-model is drawn as a black line in the background and sensor locations at the bridge are marked grey, dashed vertical lines. As for the mode shapes found by AOMA, one line is drawn for each detection assigned to the trace. The stronger the opacity, the more mode shapes lie on top of each other. Discrepancies by thinner lines is clearly visible in several plots.

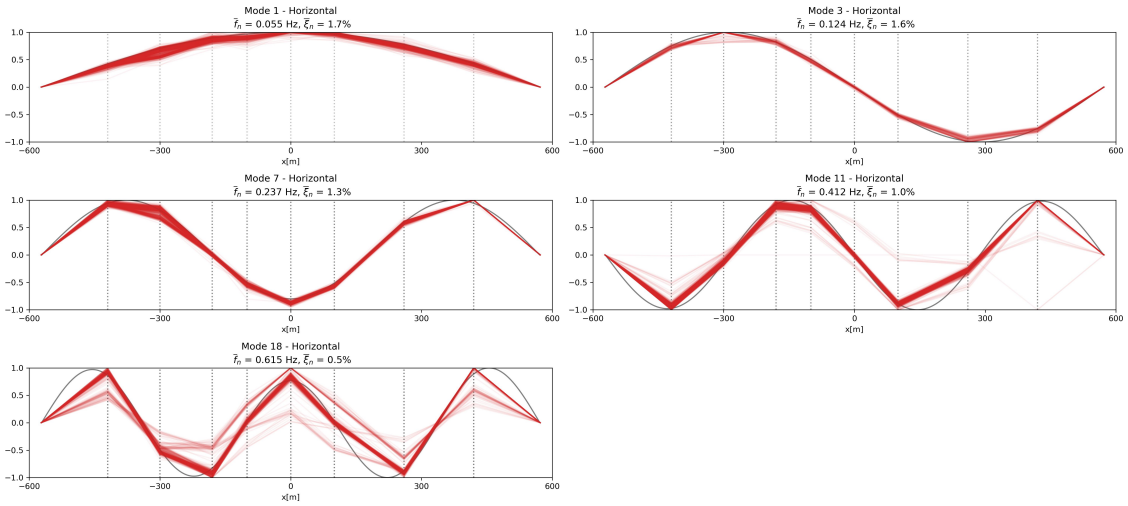


Figure 4.19: Mode shapes of horizontal modes from AOMA, with corresponding mean natural frequencies and mean damping ratios.

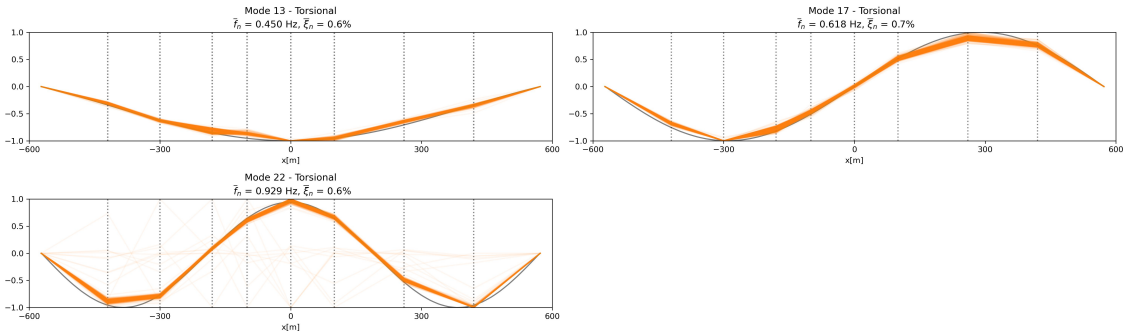


Figure 4.20: Mode shapes of torsional modes from AOMA, with corresponding mean natural frequencies and mean damping ratios.

As for the vertical mode shapes shown in figure 4.21, for mode 12 and remaining higher order modes, the amount of sensors on the bridge deck limits the ability of AOMA to reproduce the mode shapes correctly. For these higher order modes, AOMA will interpolate the mode shape in the areas between the sensor locations. There are still valuable information in the plots as it is of interest to observe the consistency of the mode shapes. The points of intersection between the mode shapes from Abaqus and the ones estimated by AOMA may also say something about the accuracy of the estimations by AOMA.

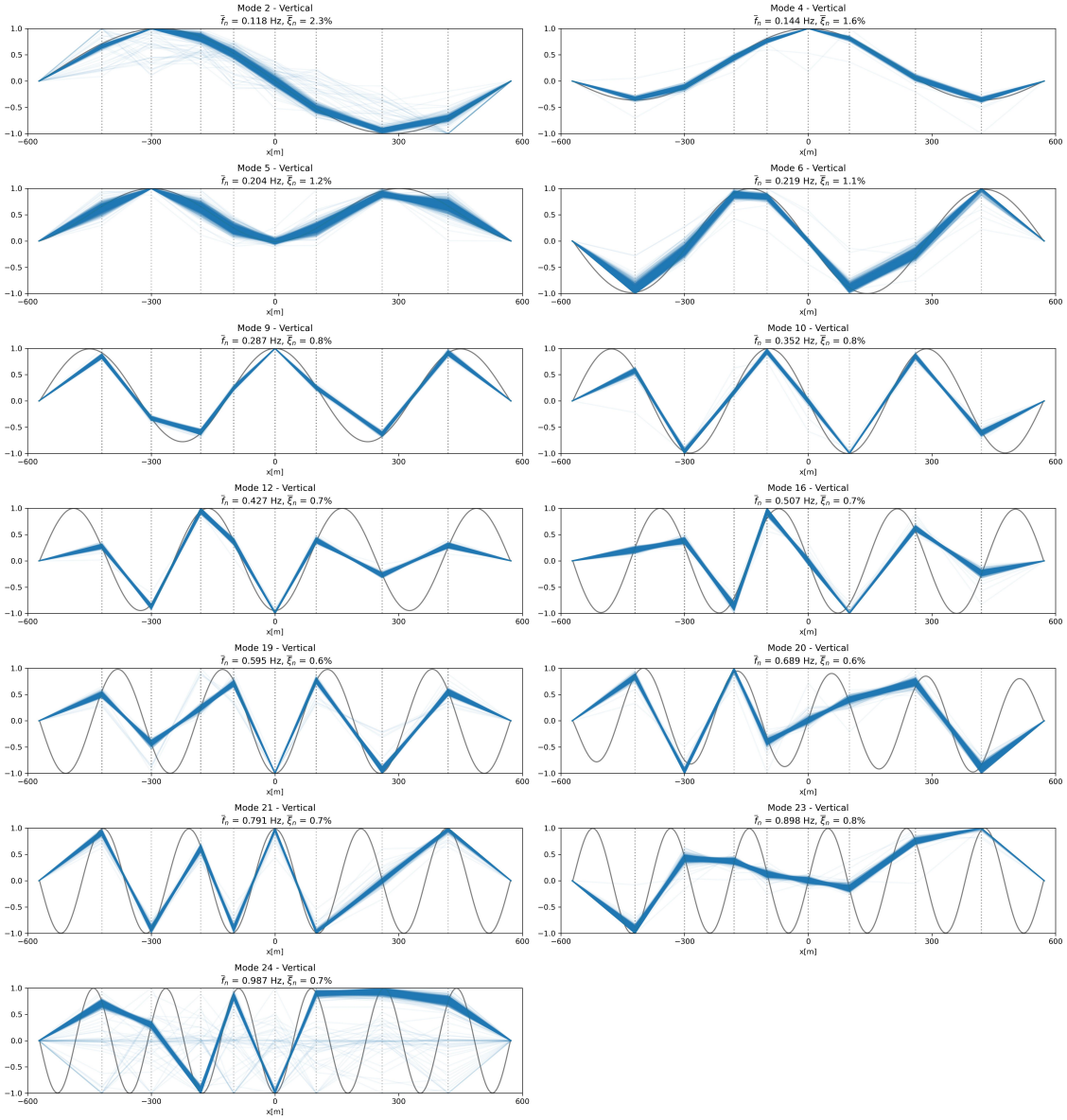


Figure 4.21: Mode shapes of vertical modes from AOMA, with corresponding mean natural frequencies and mean damping ratios.

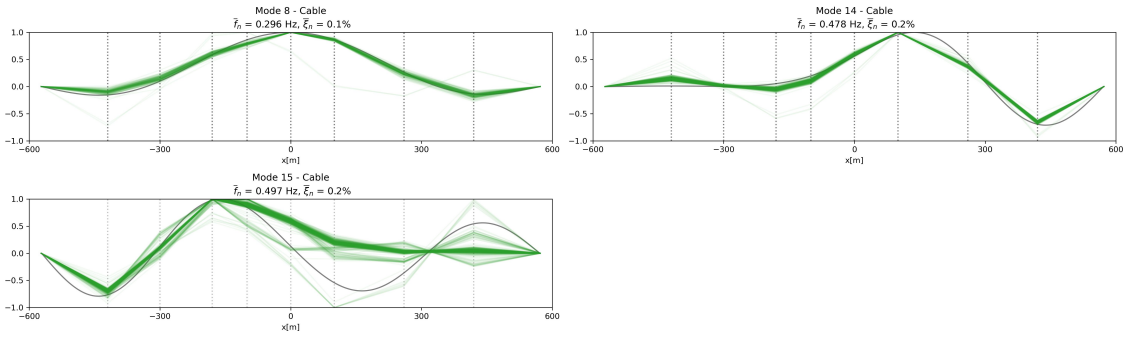
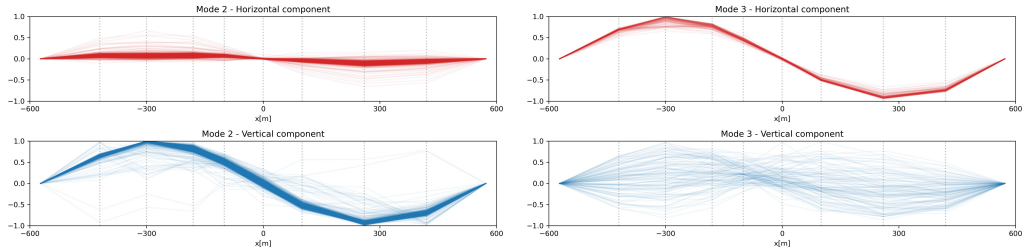


Figure 4.22: Mode shapes of cable modes from AOMA, with corresponding mean natural frequencies and mean damping ratios. The figure shows the horizontal component of the bridge deck, excited by the cable modes.

Mode 2 and 3, which is vertical and horizontal modes respectively, is closely spaced in the frequency domain with similar mode shapes in each respective component. Because of the bad detection rate of mode 3, suspicion about dynamic coupling in between the modes was raised. To further investigate this suspicion, the different components of the two mode shapes is visualized in figure 4.23.



(a) Mode 2 (b) Mode 3

Figure 4.23: Mode shape plot with horizontal, vertical and torsional components of mode 2 and 3.

4.2.3 Comparison of AOMA and FEM

A comparison of the results from AOMA and FEM is presented in this section. Comparison of mode shapes is already presented in section 4.2.2. Table 4.2 presents natural frequency and damping ratios of all modes that is member of a trace. Both natural frequency and damping ratios are extracted as the mean over all members within each trace. The table also shows the natural frequency of modes from FE-model and their relative difference to the modes from AOMA. Detection rate is also presented and type H, V, T and C denotes horizontal, vertical, torsional and cable modes, respectively.

Table 4.2: Comparative results between FEM and AOMA.

Mode #	Type	f_{FEM} [Hz]	$\overline{f}_{\text{AOMA}}$ [Hz]	$\Delta f_{\text{rel}} = \frac{\lvert f_{\text{FEM}} - \overline{f}_{\text{AOMA}} \rvert}{f_{\text{FEM}}} \cdot 100$ [%]	$\overline{\xi}_{\text{AOMA}}$ [%]	Detection rate [%]
1	H	0.054	0.055	1.52	1.69	75.3
2	V	0.116	0.118	2.16	2.30	100.0
3	H	0.120	0.124	3.57	1.60	9.7
4	V	0.144	0.144	0.12	1.61	100.0
5	V	0.206	0.204	0.66	1.16	99.7
6	V	0.218	0.219	0.63	1.06	100.0
7	H	0.228	0.237	3.89	1.30	67.3
8	C	0.284	0.296	4.52	0.11	13.0
9	V	0.286	0.287	0.38	0.84	100.0
10	V	0.348	0.352	1.05	0.77	100.0
11	H	0.397	0.412	3.53	0.97	69.1
12	V	0.424	0.427	0.73	0.68	100.0
13	T	0.438	0.450	2.71	0.56	99.3
14	C	0.463	0.478	3.23	0.24	33.7
15	C	0.483	0.497	2.88	0.20	32.0
16	V	0.499	0.507	1.63	0.71	100.0
17	T	0.585	0.618	5.63	0.71	98.0
18	H	0.586	0.615	4.82	0.52	32.4
19	V	0.591	0.595	0.61	0.64	100.0
20	V	0.690	0.689	0.16	0.64	99.9
21	V	0.789	0.791	0.30	0.69	100.0
22	T	0.883	0.929	5.19	0.64	98.5
23	V	0.903	0.898	0.61	0.75	99.9
24	V	1.018	0.987	3.10	0.66	96.6

Natural frequencies from AOMA and FEM is plotted against each other in figure 4.24, one subplot for horizontal, vertical, torsional and cable modes respectively.

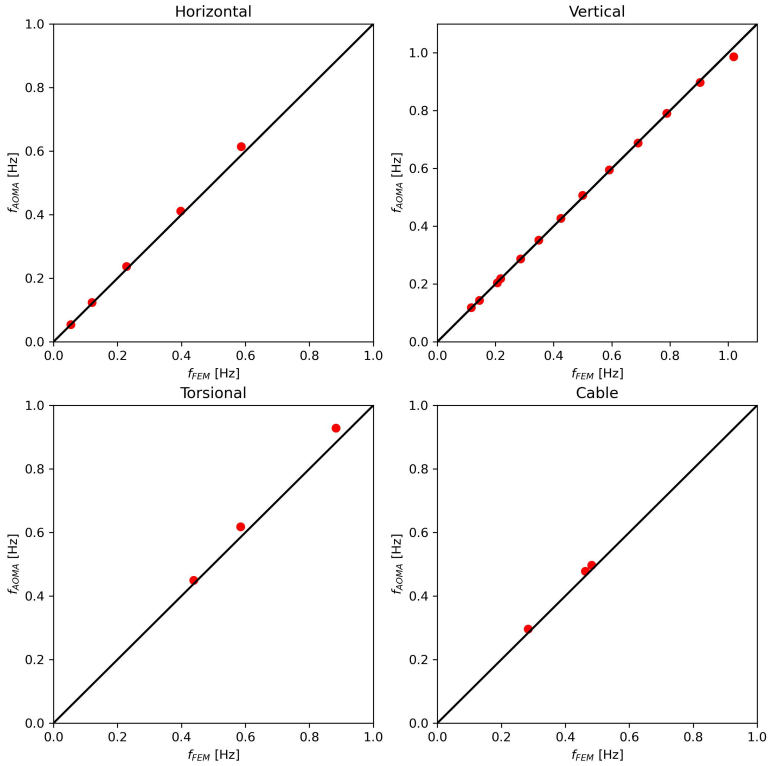


Figure 4.24: Natural frequencies from FEM plotted against natural frequencies from AOMA for horizontal, vertical, torsional and cable modes, respectively.

4.2.4 Environmental factors influence on modal parameters

This section presents result of the modal analysis in context with weather data recorded by the monitoring system, described in section 4.1.1. The weather data is extracted as the average over the corresponding period passed into the AOMA. Figure 4.25 shows a wind-rose over the wind direction and magnitude at the mid-span of the Hålogaland bridge for the recorded period. The wind magnitude is given in meters per second (m/s) and is divided into five buckets with interval of 5 m/s each. Observe that the predominating wind direction is from the east, hence hitting perpendicular on the longitudinal axis of the Hålogaland bridge. Specifically, mean wind direction is 121.5 degrees, mean wind speed is 8.5 m/s and max wind speed is 30 m/s.

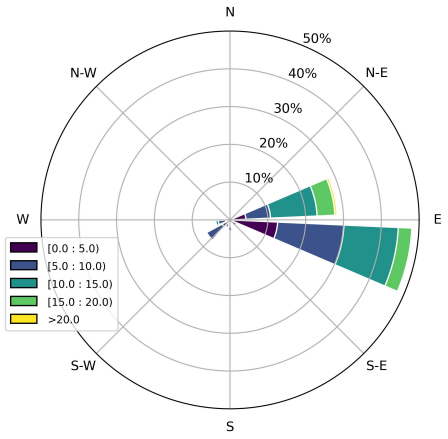


Figure 4.25: Wind-rose of wind at the Hålogaland bridge for the analyzed period.

Figure 4.26 displays the modes matched to references in Abaqus, in addition, temperature, mean wind speed and mean wind direction is included as subplots. Temperatures alternates from $-9.1°C$ to $3.0°C$, with a mean temperature of $-4.1°C$. Small changes in the mode traces, especially for higher order modes, is observable and it is desirable to investigate potential correlation between weather patterns and change in mode trace development.

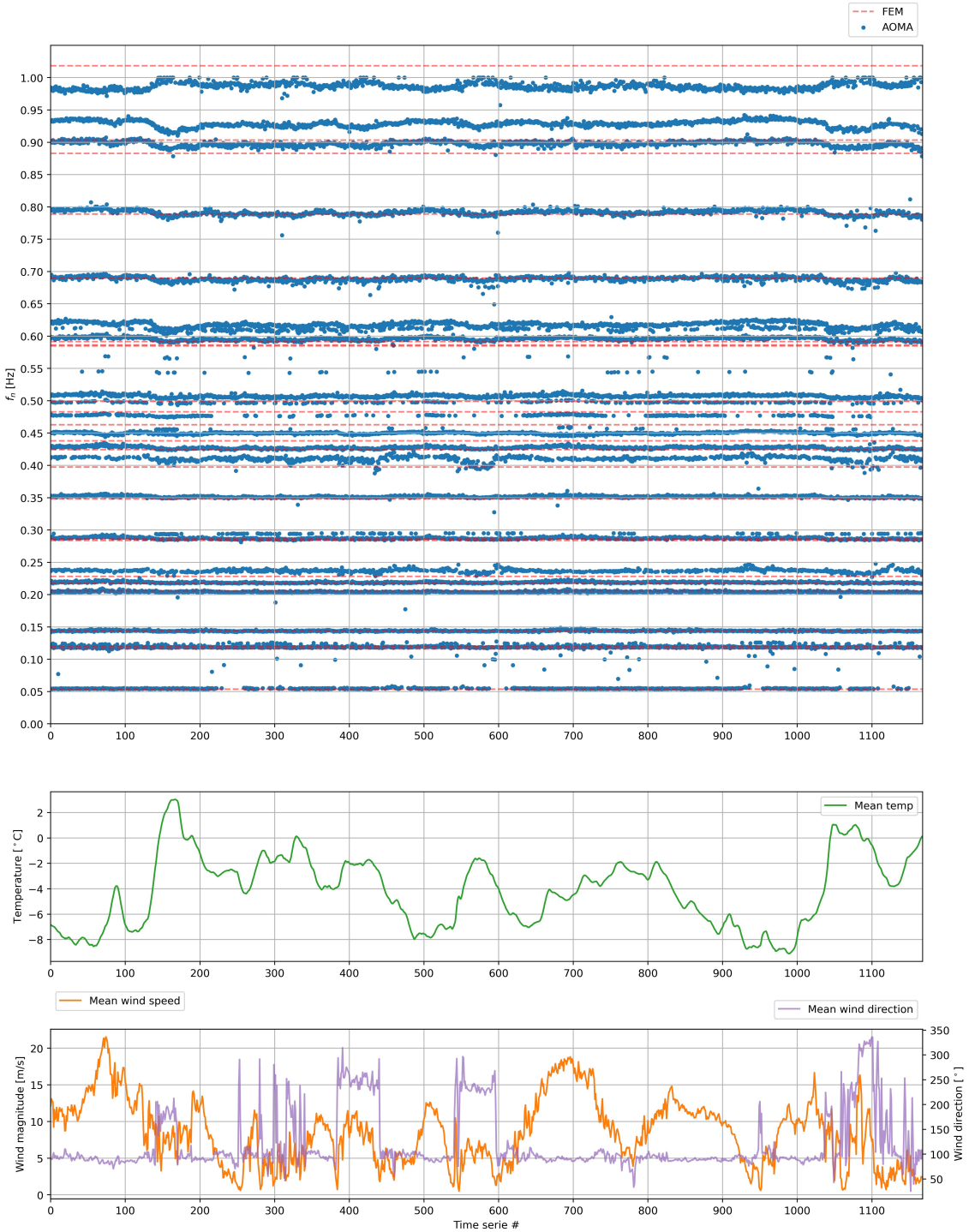


Figure 4.26: Modes detected by AOMA matched to reference mode in Abaqus, plotted by frequency vs. time. Temperature, mean wind speed, and mean wind direction vs. time is added as subplots.

Figure 4.27 shows frequency vs. temperature correlation plot for each mode trace. Note that for mode 8, the pronounced incline on the red regression line is caused by outliers not visible in the plot, disturbing the regression. Multiple vertical lines are visible for mode 15 and 18, indicating that some modes has been wrongly matched to the reference, which agrees with figure 4.18.

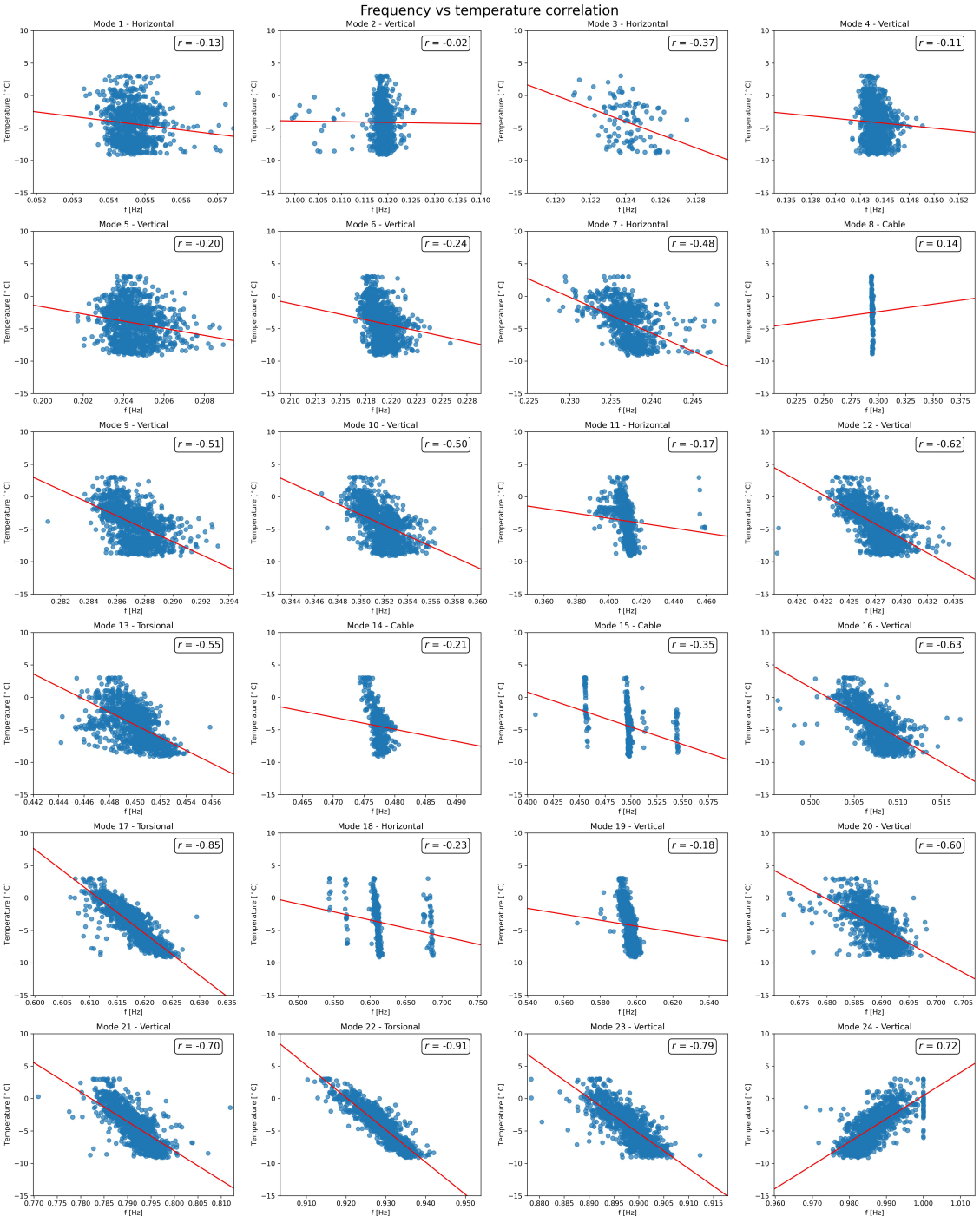


Figure 4.27: Frequency vs. temperature correlation plot. Pearson correlation coefficient r between the two variables is displayed in the top right corner of each subplot and plotted as a red solid line.

Figure 4.28 shows damping vs. time of all mode traces superimposed on each other, in addition to mean wind speed as subplot below.

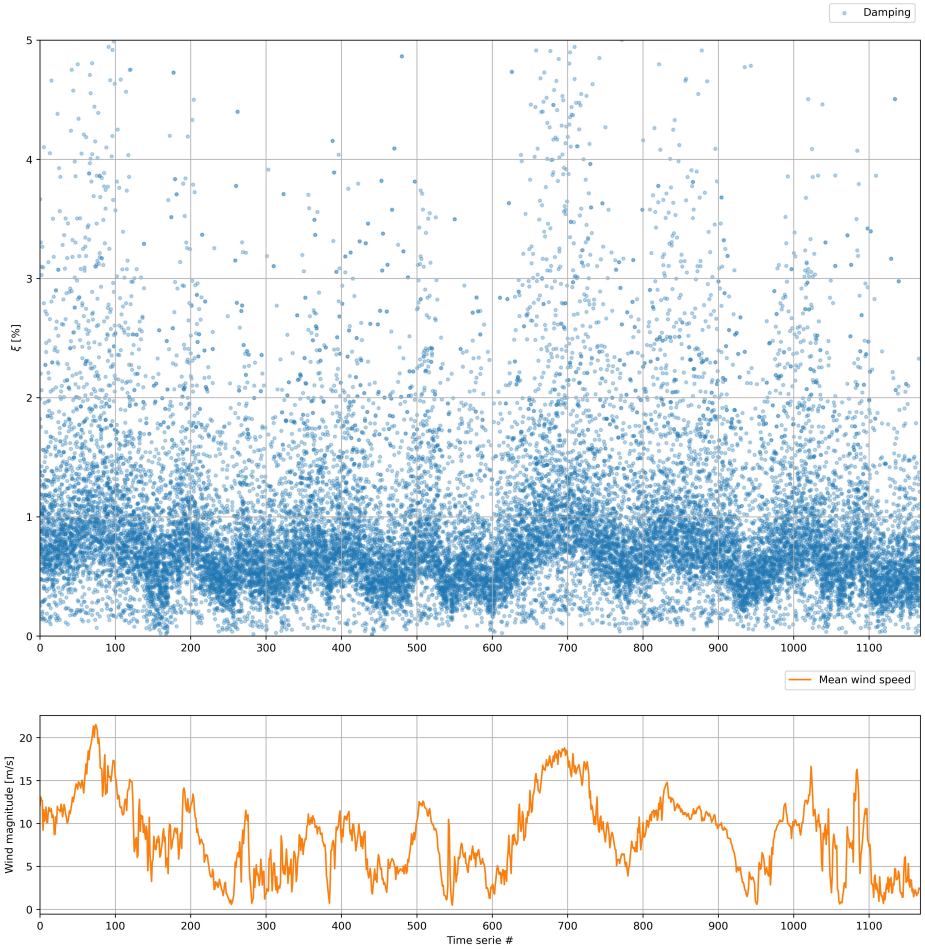


Figure 4.28: Damping vs. time plot of all mode traces and mean wind speed is added as subplot.

Figure 4.29 shows damping vs. wind speed correlation plot for each mode trace.

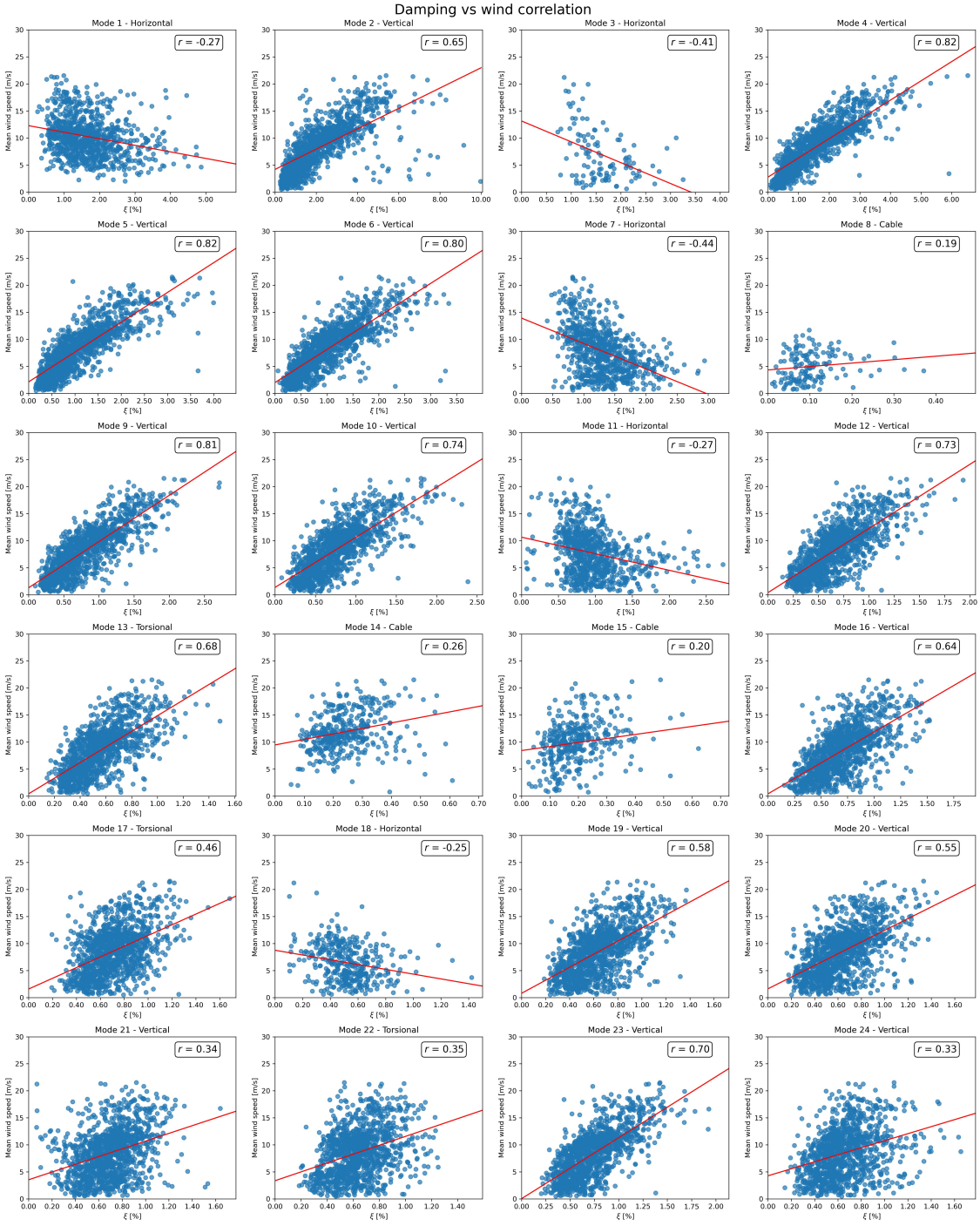


Figure 4.29: Damping vs. wind correlation plot. Pearson correlation coefficient r between the two variables is displayed in the top right corner of each subplot and plotted as a red solid line.

As mentioned in section 4.2.2, it was desirable to make a more thorough investigation in the detection pattern of mode 1 due to discrepancies compared to the other mode traces in figure 4.16. Figure 4.30 shows the trace of mode 1 with a plot of the mean wind speed, where the mean wind is labeled red at time instances where no detection of mode 1 exists, orange where detection exists. The figure shows that missed detections mostly occurs at lower wind speeds. The mean wind speed of the periods where mode 1 is detected is 10.2 m/s and the mean wind speed of periods where the mode is not detected is 3.2 m/s.

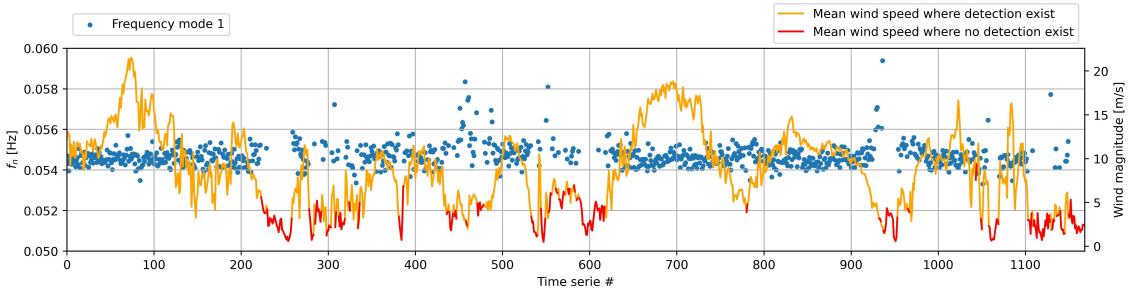


Figure 4.30: Frequency trace of mode 1 with mean wind speed.

4.3 Discussion

4.3.1 Automatic Operational Modal Analysis

Results from AOMA on the Hålogaland bridge is presented in section 4.2.2. As previously mentioned, all modes up to 1 Hz are considered from the modal analysis. On average, the algorithm underestimates the number of modes for the analyzed time series. Comparing the standard deviation to the numerical example, taking the number of reference modes in account, one achieves $2.04/24 = 0.09$ for AOMA against $0.55/9 = 0.06$ for the numerical example.

It is pointed out that there has been an iterative process to arrive at the results presented in section 4.2. Important decisions has been made and it is desirable to explain the background for some of them in the following sections.

Cov-SSI

Cov-SSI as outlined in section 2.4.1 was first carried out, that is covariances between all input channels were computed as in equation 2.44. The result was an expensive calculation cost with long run times at first. This made iterative work impractical as different configurations had to be tested on the data set. Connecting to a data cluster possessed by the university would be a potential solution, but it would not reduce the computational cost and later applications would also have a demand for powerful computers. Therefore, reference-based Cov-SSI, as outlined in section 2.4.2, was considered an alternative to reduce computational cost, enabling execution on local computers and making it convenient for everyone to carry out. Here, covariances between all channels and reference channels only, as in equation 2.75, had to be carried out. Theoretically, this should reduce computational cost by $1 - \frac{48 \cdot 16}{48^2} = 67\%$. The implementation of reference-based Cov-SSI reduced the experimental computational cost by roughly 50%. It is important to emphasize

that other adjustments also made important contributions to arrive at the final run time. Still, Cov-SSI was deemed, by far, the most expensive sub routine in the framework, so the introduction of reference-based Cov-SSI was of great importance.

Comparison of accuracy in between reference-based Cov-SSI and traditional Cov-SSI was not carried out in his study, nor was it an objective. Peeters and De Roeck [11] performed experimental testing on a steel mast and found that reference-based SSI was considerably faster than SSI. Concluding remarks was that the natural frequencies and damping ratios was determined with low uncertainties for both methods, the same applies for mode shapes. The study found that reference-based SSI prediction errors are higher for channels that do not belong to the reference channels. The practical significance of this finding for the Hålogaland bridge is that the torsional modes may suffer a higher prediction error, since all y and z channels on the sensors on the west side of the bridge was chosen as reference channels.

Selection of hyperparameters

Hyperparameter selection for AOMA algorithms remains one of the most difficult topics when applied on operating structures. The variety in structure complexity and load characteristics has prevented the algorithms to be off the shelf items where one size fits all. Once again, it is emphasized that this study's main objective is data analysis of the recordings from the Hålogaland bridge, not algorithm optimization or hyperparameter search. Still, some experiences has been made that can contribute against generalizing the choices for later usage.

Selection of the number of block-rows i according to equation 2.84 seems to be sufficient for practical applications. Yang et al. [17] states that i can be chosen as large as possible within practical limitations, at the cost of computational expenses. As Cov-SSI is found to be the most expensive routine of AOMA, it is suggested to take every possible action to reduce computational time. Keeping i close to equation 2.84 by adding only 25%, is found sufficient.

The length of the time series was chosen to 30 minutes after recommendations from supervisor without further exploration. This leaves a room for more investigations and potential improvements of the study. The essence of choosing the length, is as previously mentioned, the compromise of obtaining a good statistical representation of the signal without experiencing moving averages. In order to improve the latter, it is possible to subtract the mean of the signal for every time series passed into Cov-SSI, i.e. every 30 minutes, instead of every 8 hour as performed in this study.

The model orders to perform system identification for, is as common, heavily overestimated at the cost of increased computational time. Still, it is considered useful for reasons explained later as it should be seen in context with HDBSCAN.

Stabilization analysis is a relatively simple assessment routine utilized in previously automated [19] and non-automated [1] [18] applications of OMA. The intention is to discard mathematical poles from a stabilization diagram. It is important to note that in an AOMA context, such a routine should be less rigorous than in traditional OMA, just as in the previously cited studies. The reason is that it is important to preserve all potential physical poles and some mathematical poles representing noise for the later applied HDBSCAN, which is designed to be applied on data with noise. Hence, preservation of mathematical poles is important for the algorithm to perform well. Manual inspection of several stabilization diagrams equivalent to figure 2.9 was done iteratively

in order to obtain parameters for the stabilization analysis, such that as much as possible of the consistent clear vertical lines was preserved for later analysis. Resulting outcome was that parameters for stabilization analysis were set relatively generous.

HDBSCAN is the most powerful tool for data pattern recognition and outlier detection amongst the routines in the AOMA framework presented in this study. The parameters "minimum cluster size" and "minimum samples" are, as previously indicated, closely linked to the number of orders system identification is performed for. The more order the system identification is performed for, the more mathematical poles will appear in the stabilization diagram, but the vertical lines of assumed physical poles will also grow taller. The mathematical poles will, even though the order is increased, have a scattered and messy pattern, while the physical poles will continue to stack vertically. While increasing the noise in the data set, the size of the consistent clusters will also increase and a better statistical foundation for HDBSCAN is created. When increasing the cluster size and the amount of noise, HDBSCAN will acquire a better understanding of what is dense areas and what is scattered areas among the data. Setting the number of orders sufficiently high and setting less rigorous stabilization criterion in the stabilization analysis, will enable good working conditions for HDBSCAN. It is then possible to increase the "minimum cluster size" so that HDBSCAN looks for larger consistent clusters and at the same time increase "minimum samples" such that more points are declared noise and clustering is confined to denser areas. Manual inspection of several stabilization diagrams equivalent to figure 2.10 was done iteratively in order to obtain values for "minimum cluster size" and "minimum samples". It should be noted that this was a trial and error process, as tuning two dependant parameters simultaneously is a complex exercise that would have deserved more attention. The optimal outcome of figure 2.10 is to obtain the same number of clusters each time with as low standard deviation as possible. In practice this is a hard problem and wrongly detected clusters are inevitable as excitation of the bridge varies greatly according to wind magnitude.

Traces and references

Figure 4.14 is the pure output of the AOMA and reveals traces of how the modes develops over time. A challenging question concerning mode traces is how to decide which trace a mode belongs to. For that purpose, reference modes from Abaqus was utilized to match the estimated modes from AOMA to, where measures of mode shape similarity and difference in frequency described in section 4.1.4 was used. The task of picking the right reference modes from Abaqus was not straight forward. There are about 50 modes of vibration from the Abaqus model in the frequency range between 0 and 1 Hz. Several of those modes are cable modes or tower modes that is not expected to be well captured by the monitoring system with limited amounts of sensors located on those components of the bridge. Nevertheless, some of the cable modes of the main cables, seems to excite the bridge deck, so the question arose if they would appear in the mode trace plot. To begin with, all horizontal, vertical and torsional modes were chosen as reference modes. Two horizontal modes from Abaqus, specifically number 46 and 49 amongst all modes, at natural frequency 0.86 Hz and 0.89 Hz respectively, did not show any evidence to appear in the mode trace plot of figure 4.14 by manual inspection and was for that reason removed for further analysis. Also, the second horizontal mode, mode 3, showed little sign of appearance, but because of its significance it was still kept. However, there appeared to be evidence of a couple of traces on other frequencies, firstly labeled orange in figure 4.16 that were assumed to be horizontal bridge deck motions excited by

cable modes. Assessment of cable modes in the relevant frequency range was therefore executed in order to find references for the appearing excessive traces. References was successfully identified, even though the detection rate of the cable modes by AOMA was not particularly impressive as shown in figure 4.18. It is still an interesting finding that AOMA, under the right circumstances, is able to detect cable modes of the bridge.

A consideration was made on whether to include the channels of the cable sensors in the modal analysis, to better capture the motions of the cable modes. An attempt was made that appeared to provide a better detection rate of the cable modes, but unfortunately led to worse detection rate for some other horizontal modes. As modal analysis of the bridge deck has been the main focus of this study, the cable sensors was discarded and the original set up was kept.

The mode shapes in figure 4.19, 4.20, 4.21 and 4.22 seems to coincide well with the reference mode shapes from Abaqus, except from mode 12 and remaining higher order vertical modes. This is expected an mentioned earlier. Evidence of good consistency in the plots appears for almost all modes, with exception for mode 15, 18 and 24 that has some discrepancies. Since three different types of modes, that is horizontal, vertical and cable, are experiencing discrepancies, no further conclusion is to be drawn on the ability to describe mode shapes of a certain type over others.

It still remains unknown why mode 3 had such a bad detection rate of only 9.7% (see figure 4.18). It is a low order mode with a well characterized mode shape, expected to give significant contribution to the global displacement of the bridge deck. Observed in figure 4.17 as green dots between 0.10 and 0.15 Hz, it is situated right next to mode 2 in the frequency domain. Even though mode 2 is a vertical mode, it should be able to distinguish between them by assessment of mode shape. Nevertheless, it may still happen that the described framework for AOMA struggles to distinguish between modes that is close by frequency. After all, the difference in frequency between the two modes is only 0.004 Hz. The most plausible explanation though, may be found in 3D flutter analysis [27] which states that dynamic coupling between natural modes closely spaced in the frequency domain, takes place because of self-exciting aerodynamic forces. Hence, mode 2 and 3 may become coupled during adequate wind loading and therefore only be detected as one mode by AOMA. This assumption is confirmed by figure 4.23 where the mode shape of mode 3 is recognized in the horizontal component of mode 2 and the mode shape of mode 2 is recognized in the vertical component of mode 3.

Lastly, it is important to emphasize that orange dots in figure 4.16 labeled un-matched, may not be actual noise or clusters represented by spurious poles. They might as well be physical modes representing other vibration modes of the bridge than the ones undertaken in this study.

4.3.2 Comparison of AOMA and FEM

Comparison of modal parameters from the Abaqus model and AOMA is presented in section 4.2.3. Modal parameters estimated by AOMA seems to confirm the modal parameters obtained from the Abaqus model. Relative difference in between the two methods are presented in table 4.2 and the largest relative difference is 5.63%. Vertical mode shapes seems to be the mode type lying closest to the Abaqus model, averaging at a relative difference at 0.93%, with only mode 24 contributing with a significant large deviation of 3.10%. Torsional modes is the type that deviates the most from the Abaqus model with an average relative difference of 4.51%.

4.3.3 Environmental effects

Results from the modal analysis in context of environmental effects, specifically wind and temperature, is presented in section 4.2.4. Observe from figure 4.26 that several traces undergoes slightly changes over time. These changes in modal features are expected as circumstances, namely environmental factors, are also changing. An interesting observation is that some traces seems to respond more to certain factors than other. It was desirable to carry out a more thorough investigation of how environmental factors influence specific modes. Two relationships in particular were interesting to explore; relationship between temperature and frequency, and the relationship between wind and damping. For this purpose, correlation plots in figure 4.27 and 4.29 were made for frequency vs. temperature and damping vs. wind, respectively.

Correlation between frequency and temperature is weak for lower order modes and there is an increasing trend for correlation for higher order modes. Seven of the nine highest order modes has a correlation coefficient, either positive of negative, of above 0.6. The overall trend is that there is a negative correlation for higher order modes between frequency and temperature, indicating that the structure becomes stiffer at lower temperatures and vibrates at higher frequencies. The highest negative correlation coefficient is obtained from mode 22, a torsional mode, at -0.91, which is a strong negative correlation. There is a clear exception in mode 24 that experiences a positive correlation of 0.72

Aerodynamic damping is a well known phenomenon for wind exposed structures [27]. Figure 4.28 indicates that damping increases when wind magnitude increases, and this is further explored in figure 4.29. Consider well detected modes to obtain a good statistical foundation for correlation calculations and let a well detected mode have a detection rate of 80% or higher. Thus 16 modes becomes well detected and 75% of the well detected modes have a Pearson correlation coefficient of higher than 0.5. The correlation is strongest for the lower order vertical modes with a correlation coefficient of 0.82 for mode 4 and 5. The results provides good indications that aerodynamic damping, or wind induced damping, is present in the Hålogaland bridge.

4.3.4 Comparison with OMA

There is a previous study [1] that applied OMA on the vibration data from the bridge deck of the Hålogaland bridge. Modes in the frequency range from 0 - 1 Hz was considered and manually picked from a stabilization diagram produced by Cov-SSI. The study found 4 horizontal modes, 13 vertical modes and 3 torsional modes, all present in the FE-model of the bridge. In comparison, the application of AOMA in this thesis found 5 horizontal modes, 13 vertical modes, 3 torsional modes and 3 cable modes exciting the bridge deck in horizontal direction, all modes present in the FE-model of the bridge. In comparison with the FE-model, OMA had an average difference in natural frequency of 2.3% while AOMA in this study has an average difference in natural frequency of 2.2%. That is, AOMA finds more modes than OMA with about the same accuracy. It is important to mention that OMA was only applied on one single 30 minute period of vibration data from the bridge deck, assumed to have sufficient excitation of the bridge deck. The generalizability of a single experiment will be somewhat limited, but it would be extremely time consuming to perform manual interpretation of 1168 stabilization diagrams. These facts supports the use of AOMA over the use of traditional OMA.

Chapter 5

Conclusion and Further work

This chapter presents the most important findings from the results and proposal for further work. The main objectives of this study was to implement a framework for AOMA, verification of the framework on a numerical example, application of AOMA on long term vibration data from the Hålogaland bridge, comparison of modal parameters from AOMA with results from a FE-model and investigate the influence on modal parameters by environmental factors.

5.1 Conclusion

AOMA on the numerical example provided an overall detection rate of 96.4% and an average difference in natural frequency of 0.03% compared to results from a FE-model of the shear frame. This was promising results for application on the vibration data on the Hålogaland bridge.

AOMA on the vibration data from the bridge deck of the Hålogaland bridge provided an overall detection rate of 80.2% of the reference modes from the FE-model with a 2.2% average difference in natural frequency. The average computational time of each 30 minute time series under investigation was 4.9 seconds. In the frequency range of 0-1 Hz, all vertical modes of the bridge deck was estimated with a detection rate of 99.7% and an average difference in natural frequency of 0.93% compared to the FE-model. 5 horizontal modes of the bridge deck was estimated with a detection rate of 50.8% and an average difference in natural frequency of 3.6%. All torsional modes of the bridge deck was estimated with a detection rate 98.6% and an average difference in natural frequency of 4.5%. 3 cable modes exciting the bridge deck was estimated with a detection rate of 26.2% and an average difference in natural frequency of 3.5%. Mode shapes of all the detected modes coincided well with mode shapes obtain from the FE-model. A possible reason for reduced consistency in the detection of mode 3 (the second horizontal mode) specifically, may be dynamic coupling to a vertical mode closely spaced in the frequency domain, because of self-exciting aerodynamic forces. A general source of uncertainty in this study is to what extent the vibration data is stationary.

Environmental data analysis, specifically wind and temperature, showed that the dominating wind direction in the analysis period was from the east. Investigation of the modal analysis in context with environmental data revealed some interesting correlations. Negative correlation between temperature and frequency turns out to be present for higher order modes, and the correlation is

stronger the higher the order. This finding indicates that the structure becomes stiffer at lower temperatures and higher order modes vibrates at higher frequencies. Wind speeds lower than 4-5 m/s reduces the excitation of the bridge, causing AOMA to generate more false estimations in the low frequency range, which makes detection of the first natural mode of vibration, specifically, more difficult. The results also shows positive correlation between wind speed and damping in the structure, in agreement with buffeting theory.

Thus, a final conclusion is that AOMA is able to provide repeatedly good accuracy estimations of modal parameters from the bridge deck of the Hålogaland bridge. Dynamic coupling between closely spaced modes in the frequency domain may cause difficulties to distinguish between still-air-modes in an AOMA context. AOMA is able to identify more modes than traditional OMA on the same structure, supporting the use of AOMA over OMA. The results proves good for future evolution of structural health monitoring, damage detection and digital twins of critical infrastructure.

5.2 Further work

The following topics are recommended for further work:

- A thorough investigation of the bad detection rate of mode 3 should be carried out. The research should consider multimode buffeting theory and other relevant theory in order to determine if the potential dynamic coupling and response is expected behaviour.
- The potential false detections in the vicinity of mode 1, visualized as orange scattered dots in figure 4.16 should be reviewed more closely. It should be investigated if the findings can confirm lower credibility of AOMA at lower wind speeds.
- Furthermore, to carry out an even more comprehensive modal analysis of the Hålogaland bridge, sensors from cable hangers and towers should be included in addition to the corresponding nodes from the FE-model to obtain references.
- The effect of stationarity as source of uncertainty in the vibration data should be investigated thoroughly.
- Data from even longer periods, in terms of a year, should be analysed to provide better knowledge of how environmental factors influence the dynamics of the structure.
- The framework for AOMA should be tested on different structures.

References

[1] A. A. Solstad and E. L. Onstad, *Comparison of Measured and Predicted Buffeting Response of the Hålogaland Bridge Using a Probabilistic Description of the Wind Field*, 2022. [Online]. Available: https://hdl.handle.net/11250/3020477.

[2] G. T. Frøseth and O. Guddal, 'Gunnstein/strid: strid - v0.4.3', May 2022. DOI: 10.5281/ZENODO.6540518. [Online]. Available: https://zenodo.org/record/6540518.

[3] K. A. Kvåle, 'knutankv/koma: Minor fixes', Jan. 2022. DOI: 10.5281/ZENODO.5881841. [Online]. Available: https://zenodo.org/record/5881841.

[4] *Python 3.10.8 Documentation*. [Online]. Available: https://docs.python.org/release/3.10.8/.

[5] D. E. Newland, *Random Vibrations, Spectral & Wavelet Analysis*, Third Edition. 2005.

[6] C. Rainieri and G. Fabbrocino, *Operational Modal Analysis of Civil Engineering Structures*. New York, NY: Springer New York, 2014, ISBN: 978-1-4939-0766-3. DOI: 10.1007/978-1-4939-0767-0.

[7] C. C. Tseng and S. L. Lee, 'Closed-form designs of digital fractional order Butterworth filters using discrete transforms', *Signal Processing*, vol. 137, pp. 80–97, Aug. 2017, ISSN: 0165-1684. DOI: 10.1016/J.SIGPRO.2017.01.015.

[8] A. K. Chopra, *Dynamics of Structures*, Fourth Edition. 2012.

[9] L. Hermans and H. d. Van Auweraer, 'MODAL TESTING AND ANALYSIS OF STRUCTURES UNDER OPERATIONAL CONDITIONS: INDUSTRIAL APPLICATIONS', *Mechanical Systems and Signal Processing*, vol. 13, no. 2, pp. 193–216, Mar. 1999, ISSN: 0888-3270. DOI: 10.1006/MSSP.1998.1211.

[10] R. Brincker and P. Andersen, 'Understanding Stochastic Subspace Identification', vol. 18, 2006, p. 2023.

[11] B. Peeters and G. De Roeck, 'REFERENCE-BASED STOCHASTIC SUBSPACE IDENTIFICATION FOR OUTPUT-ONLY MODAL ANALYSIS', *Mechanical Systems and Signal Processing*, vol. 13, no. 6, pp. 855–878, Nov. 1999, ISSN: 0888-3270. DOI: 10.1006/MSSP.1999.1249.

[12] P.-N. Tan, M. Steinbach, A. Karpatne and V. Kumar, *Introduction to Data Mining*, Second Edition. 2020.

[13] T. H. Cormen, C. E. Leiserson, R. L. Rivest and C. Stein, *Introduction to Algorithms*. MIT press, 2022.

[14] R. J. Campello, D. Moulavi, A. Zimek and J. Sander, 'Hierarchical density estimates for data clustering, visualization, and outlier detection', *ACM Transactions on Knowledge Discovery from Data*, vol. 10, no. 1, Jul. 2015, ISSN: 1556472X. DOI: 10.1145/2733381.

[15] *How HDBSCAN Works*. [Online]. Available: https://hdbscan.readthedocs.io/en/latest/how_hdbscan_works.html.

[16] P. Van Overschee and B. De Moor, *Subspace Identification for Linear Systems*. Boston, MA: Springer US, 1996, ISBN: 978-1-4613-8061-0. DOI: 10.1007/978-1-4613-0465-4.

[17] X.-M. Yang, T.-H. Yi, C.-X. Qu, H.-N. Li and H. Liu, 'Automated Eigensystem Realization Algorithm for Operational Modal Identification of Bridge Structures', *Journal of Aerospace Engineering*, vol. 32, no. 2, Mar. 2019, ISSN: 0893-1321. DOI: 10.1061/(ASCE)AS.1943-5525.0000984.

[18] K. A. Kvåle, O. Øiseth and A. Rønnquist, 'Operational modal analysis of an end-supported pontoon bridge', *Engineering Structures*, vol. 148, pp. 410–423, Oct. 2017, ISSN: 18737323. DOI: 10.1016/j.engstruct.2017.06.069.

[19] K. A. Kvåle and O. Øiseth, 'Automated operational modal analysis of an end-supported pontoon bridge using covariance-driven stochastic subspace identification and a density-based hierarchical clustering algorithm', 2020.

[20] *The hdbscan Clustering Library — hdbscan 0.8.1 documentation*. [Online]. Available: https://hdbscan.readthedocs.io/en/latest/index.html#.

[21] *E6 Hålogalandsbrua — Statens vegvesen*. [Online]. Available: https://www.vegvesen.no/vegprosjekter/europaveg/e6halogalandsbrua/.

[22] Ø. W. Petersen, G. T. Frøseth and O. Øiseth, 'Design and deployment of a monitoring system on a long-span suspension bridge', 2021.

[23] *Abaqus - SIMULIA User Assistance 2022*. [Online]. Available: https://help.3ds.com/2022/English/DSSIMULIA_Established/SIMULIA_Established_FrontmatterMap/sim-r-DSDocAbaqus.htm?contextscope=all&id=ec01bc8c83d743a6a30123c5a034edca&fbclid=IwAR3b_sxY8FiSW-2E_oGMyZCPnPYOxtVBjnOv-DEgKqjci3kQQCdM39O4_M4.

[24] *The HDF5® Library & File Format - The HDF Group*. [Online]. Available: https://www.hdfgroup.org/solutions/hdf5/.

[25] *HDF5 for Python — h5py 3.8.0 documentation*. [Online]. Available: https://docs.h5py.org/en/stable/index.html.

[26] O. Guddal, *Comparison of methods for automatic modal analysis*, 2022. [Online]. Available: https://hdl.handle.net/11250/3026223.

[27] Y. Tamura and A. Kareem, Eds., *Advanced Structural Wind Engineering*. Tokyo: Springer Japan, 2013, ISBN: 978-4-431-54336-7. DOI: 10.1007/978-4-431-54337-4.

[28] *GitHub - emilnebb/AOMA_Halogaland*. [Online]. Available: https://github.com/emilnebb/AOMA_Halogaland.

All web pages accessed at 7^{th} of June, 2023.

Appendix

The most important python code is included in this appendix. For complete source code, see the GitHub repository of the author[28].

generate_vibration_data.py

```python
import numpy as np
import strid


def generate_data(num_stories: int, path: str):
    """
    Generate simulated data for a shear frame system and save it to a file.

    Args:
        num_stories (int): The number of stories in the shear frame system.
        path (str): The path where the generated data will be saved.

    Returns:
        None
    """

    # Create a shear frame
    sf = strid.utils.ShearFrame(num_stories, 1e3, 1e4)
    sf.set_rayleigh_damping_matrix([sf.get_natural_frequency(1), sf.get_natural_frequency(sf.n)], [.05] * 2)

    # Determine the time discretization and period
    Tmax = 1. / strid.w2f(sf.get_natural_frequency(1))
    fmax = strid.w2f(sf.get_natural_frequency(sf.n))
    T = 100 * Tmax
    fs = 5 * fmax
    t = np.arange(0., T, 1 / fs)

    # Define loads on system
    ## Unmeasureable: Stochastic loads on all floors (Process noise)
    w = np.random.normal(size=(sf.n, t.size)) * 1e-1

    ## Load matrix, f
    F = w.copy()

    # Simulate response, accelerations at each floor measured
    y0, _, _ = sf.simulate(t, F)

    noise_std = y0.std()

    # Add measurement noise
    v = np.random.normal(size=y0.shape) * noise_std
    y = y0 + v


    true_frequencies = np.array([sf.get_natural_frequency(i) / (2 * np.pi) for i in range(1, sf.n + 1)])
    true_damping = np.array([sf.get_rayleigh_damping_ratio(i) for i in range(1, sf.n + 1)])
```

```
true_modeshapes = np.array([sf.get_mode_shape(i) for i in range(1, sf.n + 1)])

# Saving the data
np.savez(path,
         y=y, fs=fs,
         true_frequencies=true_frequencies,
         true_damping=true_damping,
         true_modeshapes=true_modeshapes
         )
```

numerical_example.py

```
import koma.oma
import numpy as np
import matplotlib.pyplot as plt
import koma.clustering
import geneate_vibration_data
import strid
import copy

# Main script for numerical example

path = "/../../data/vibration_data/data_stochastic_3_floor_"

number_of_realizations = 100

# Data generation

for i in range(number_of_realizations):
    geneate_vibration_data.generate_data(9, path + str(i) + ".npz")

# Define SSI parameters
i = 50
s = 1

orders = np.arange(1, 50, 1)
stabcrit = {'freq': 0.2, 'damping': 0.2, 'mac': 0.5 }

# Cov-SSI call and pole clustering
freq_modes = []

for j in range(number_of_realizations):

    data = np.load(path + str(j) + ".npz")
    y = data["y"]
    fs = data["fs"]
    true_f = data["true_frequencies"].transpose()
    true_xi = data["true_damping"].transpose()
    true_modeshapes = data["true_modeshapes"].transpose()

    # Cov-SSI
    ssid = strid.CovarianceDrivenStochasticSID(y, fs) #, ix_references)
    modes = {}
    for order in orders:
        A, C, G, R0 = ssid.perform(order, i)
        modes[order] = strid.Mode.find_modes_from_ss(A, C, ssid.fs)

    # Sorting routine
    lambdas = []
    phis = []

    for order in modes.keys():
        modes_in_order = modes[order]
        lambdas_in_order = []
        phis_in_order = []
        for mode in modes_in_order:
            lambdas_in_order.append(mode.eigenvalue)
            phis_in_order.append(mode.eigenvector)
        lambdas.append(np.array(lambdas_in_order))
        phis.append(np.array(phis_in_order).transpose())

    lambd_stab, phi_stab, orders_stab, ix_stab = koma.oma.find_stable_poles(lambdas, phis, orders, s,
                            stabcrit=stabcrit, valid_range={'freq': [0, np.inf], 'damping':[0, 0.2]},
                            indicator='freq', return_both_conjugates=False)

    #Pole clustering
    pole_clusterer = koma.clustering.PoleClusterer(lambd_stab, phi_stab, orders_stab, min_cluster_size=25,
                                    min_samples=10, scaling={'mac':1.0, 'lambda_real':1.0, 'lambda_imag': 1.0})
    prob_threshold = 0.99   #probability of pole to belong to cluster,
```

```
                        # based on estimated "probability" density function
    args = pole_clusterer.postprocess(prob_threshold=prob_threshold, normalize_and_maxreal=True)

    xi_auto, omega_n_auto, phi_auto, order_auto, probs_auto, ixs_auto = koma.clustering.group_clusters(*args)

    xi_mean = np.array([np.mean(xi_i) for xi_i in xi_auto])
    fn_mean = np.array([np.mean(om_i) for om_i in omega_n_auto])/2/np.pi

    xi_std = np.array([np.std(xi_i) for xi_i in xi_auto])
    fn_std = np.array([np.std(om_i) for om_i in omega_n_auto])/2/np.pi

    freq_modes.append([freq for freq in fn_mean])


new_freqs = np.empty(shape=[number_of_realizations, len(true_f)], dtype=object)


def remove_and_return_min_distance(lst, reference):
    """
    Removes and returns the element in the list `lst` that has the minimum absolute difference with `reference`.

    Args:
        lst (list): A list of elements.
        reference (float): The reference value to compare against.

    Returns:
        The element with the minimum absolute difference with `reference`, if it exists and the absolute difference
↪   is less than 0.03.
        None, if no such element exists or the absolute difference is greater than or equal to 0.03.
    """
    min_element = min(lst, key=lambda x: abs(x - reference), default="EMPTY")
    if min_element in lst and abs(min_element-reference)<0.03:
        lst.remove(min_element)
        return min_element
    else:
        return None

new_freqs = np.array(np.empty(shape=[number_of_realizations, len(true_f)], dtype=object), dtype=np.float)
candidates = copy.deepcopy(freq_modes)

for i in range(number_of_realizations):
    for j in range(len(true_f)):
        candidate = remove_and_return_min_distance(candidates[i], true_f[j])
        new_freqs[i, j] = candidate

freqs = []
num = []

for i in range(number_of_realizations):
    freqs.extend(freq_modes[i])
    num.extend(np.ones_like(freq_modes[i])*i)

#Plot

colors =['tab:blue', 'tab:orange', 'tab:green', 'red', 'cyan', 'purple', 'magenta', 'yellow', 'brown']

plt.figure(figsize=(9, 5), dpi=300)
plt.axhline(y = true_f[0], color = 'r', linestyle = '--', label="FEM modes")
plt.scatter(np.array(num), np.array(freqs), marker='.', color='grey')

for i in range(len(true_f)):
    if (i<len(true_f)):
        plt.axhline(y = true_f[i], color = 'r', linestyle = '--')
    plt.plot(np.arange(0, number_of_realizations), new_freqs[:,i], color=colors[i], linestyle="-", marker=".",
             label="Est. mode " + str(i+1))

plt.grid()
plt.legend(bbox_to_anchor = (1,1))
plt.xlabel("Realization #")
```

```
plt.ylabel("$f_n$ [Hz]")
#plt.ylim([0,1.05])
plt.xlim([0,100])
plt.savefig("num_example.jpg", bbox_inches='tight')
plt.show()

print(np.count_nonzero(~np.isnan(new_freqs), axis=0))

print(np.count_nonzero(~np.isnan(new_freqs)))
print(len(freqs))

f_mean = np.nanmean(new_freqs, axis=0)
print(f_mean)

print((true_f))

print(100*np.abs(true_f-f_mean)/true_f)
```

processer.py

```python
import numpy as np
from scipy import signal


def low_pass(old_signal: np.ndarray, sampling_frequency, cutoff_frequency, filter_order) -> np.ndarray:
    """
    Apply a low-pass Butterworth filter to the input signal.

    Args:
        old_signal (np.ndarray): The input signal to be filtered.
        sampling_frequency (float): The sampling frequency of the input signal.
        cutoff_frequency (float): The cutoff frequency of the low-pass filter.
        filter_order (int): The order of the Butterworth filter.

    Returns:
        np.ndarray: The filtered signal after applying the low-pass filter.
    """

    sos = signal.butter(filter_order, cutoff_frequency, btype='lowpass',
                        fs=sampling_frequency, output='sos')

    filtered_signal = signal.sosfilt(sos, old_signal)

    return filtered_signal


def downsample(sampling_frequency_old, old_signal: np.ndarray, sampling_frequency_new) -> np.ndarray:
    """
    Downsample the input signal by a given factor.

    Args:
        sampling_frequency_old (float): The original sampling frequency of the input signal.
        old_signal (np.ndarray): The input signal to be downsampled.
        sampling_frequency_new (float): The desired sampling frequency of the downsampled signal.

    Returns:
        np.ndarray: The downsampled signal.
    """

    factor = int(sampling_frequency_old / sampling_frequency_new)

    down_sampled_signal = old_signal[::factor]

    return down_sampled_signal
```

dataloader.py

```
import matplotlib.pyplot as plt
import numpy as np
import h5py
from src.AOMA.processer import low_pass, downsample


class HDF5_dataloader:
    """
    A dataloader specified for the data logged at Hålogaland bridge, loaded from HDF5 file format.
    """

    def __init__(self, path: str, bridgedeck_only: bool):
        """
        Initializes an instance of the class.

        Args:
            path (str): The path to the HDF5 file.
            bridgedeck_only (bool): Specifies whether to consider only bridge deck sensors.

        Attributes:
            path (str): The path to the HDF5 file.
            data_types (list): The list of data types available in the HDF5 file.
            hdf5_file (h5py.File): The HDF5 file object.
            periods (list): The list of periods in the HDF5 file.
            acceleration_sensors (list): The list of acceleration sensors.
            wind_sensors (list): The list of wind sensors.
            temp_sensors (list): The list of temperature sensors.
        """

        self.path = path
        self.data_types = None
        self.hdf5_file = None
        self.hdf5_file = h5py.File(self.path, 'r')
        self.periods = list(self.hdf5_file.keys())
        self.data_types = list(self.hdf5_file[self.periods[0]].keys())

        if bridgedeck_only:
            self.acceleration_sensors = ['A03-1', 'A03-2', 'A04-1', 'A04-2', 'A05-1', 'A05-2', 'A06-1', 'A06-2',
                                         'A07-1', 'A07-2', 'A08-1', 'A08-2', 'A09-1',
                                         'A09-2', 'A10-1', 'A10-2']  # bridge deck only
        else:
            self.acceleration_sensors = ['A03-1', 'A03-2', 'A04-1', 'A04-2', 'A05-1', 'A05-2', 'A06-1',
                                         'A06-2', 'A07-1', 'A07-2', 'A08-1', 'A08-2', 'A09-1', 'A09-2',
                                         'A10-1', 'A10-2', 'A06-3', 'A06-4', 'A08-3', 'A08-4']  # hangers added

        self.wind_sensors = ['W03-7-1', 'W04-15-1', 'W05-17-1', 'W05-18-1', 'W05-19-1', 'W05-19-2', 'W07-28-1',
                             'W10-45-1', 'W10-47-1', 'W10-49-1']

        self.temp_sensors = ['T01-1', 'T01-2', 'T02-1', 'T02-2', 'T03-1', 'T03-2', 'T04-1', 'T04-2', 'T05-1',
                             'T05-2', 'T06-1', 'T06-2', 'T07-1', 'T07-2', 'T08-1', 'T08-2', 'T09-1', 'T09-2',
                             'T10-1', 'T10-2', 'T11-1', 'T11-2']

    def load_acceleration(self, period: str, sensor: str, axis: str, preprosess=False, cutoff_frequency=None,
                          filter_order=None):
        """
        Loads acceleration data from the HDF5 file.

        Args:
            period (str): The period of data to load.
            sensor (str): The sensor from which to load the data.
            axis (str): The axis of acceleration data to load.
            preprosess (bool, optional): Flag to enable data preprocessing. Defaults to False.
            cutoff_frequency (float, optional): The cutoff frequency for low-pass filtering. Defaults to None.
            filter_order (int, optional): The order of the filter for low-pass filtering. Defaults to None.

        Returns:
            np.ndarray: The loaded acceleration data.
```

```
        """

        acc_data = self.hdf5_file[period][self.data_types[0]][sensor][axis]

        if preprosess:
            sampling_rate = self.hdf5_file[period][self.data_types[0]][sensor].attrs['samplerate']
            filtered_acc = low_pass(acc_data - np.mean(acc_data), sampling_rate, cutoff_frequency, filter_order)
            acc_data = downsample(sampling_rate, filtered_acc, cutoff_frequency * 2)

        return acc_data

    def load_all_acceleration_data(self, period: str, preprosess=False, cutoff_frequency=None, filter_order=None):
        """
        Loads all acceleration data for a given period.

        Args:
            period (str): The period of data to load.
            preprosess (bool, optional): Flag to enable data preprocessing. Defaults to False.
            cutoff_frequency (float, optional): The cutoff frequency for low-pass filtering. Defaults to None.
            filter_order (int, optional): The order of the filter for low-pass filtering. Defaults to None.

        Returns:
            Union[np.ndarray, bool]: The loaded acceleration data as a matrix,
            or False if all channels are not included.
        """

        # Check if all channels are included
        if not set(self.acceleration_sensors).issubset(list(self.hdf5_file[period][self.data_types[0]].keys())):
            return False

        acc_example = self.load_acceleration(self.periods[12], self.acceleration_sensors[0], 'x', preprosess,
                                             cutoff_frequency, filter_order)

        acc_x = np.zeros((len(acc_example), len(self.acceleration_sensors)))
        acc_y = np.zeros((len(acc_example), len(self.acceleration_sensors)))
        acc_z = np.zeros((len(acc_example), len(self.acceleration_sensors)))

        counter = 0
        for sensor in self.acceleration_sensors:
            acc_x[:, counter] = self.load_acceleration(period, sensor, 'x', preprosess, cutoff_frequency,
            ↪  filter_order)
            acc_y[:, counter] = self.load_acceleration(period, sensor, 'y', preprosess, cutoff_frequency,
            ↪  filter_order)
            acc_z[:, counter] = self.load_acceleration(period, sensor, 'z', preprosess, cutoff_frequency,
            ↪  filter_order)
            counter += 1

        acc_matrix = np.concatenate((acc_x, acc_y, acc_z), axis=1)

        return acc_matrix

    def load_wind(self, period: str, sensor: str):
        """
        Loads wind measurements for a given period and sensor.

        Args:
            period (str): The period of data to load.
            sensor (str): The sensor from which to load the data.

        Returns:
            Tuple[np.ndarray, np.ndarray]: Tuple containing wind magnitude and wind direction arrays.
        """

        # Wind measurements has a 32 Hz sampling rate

        wind_magnitude = np.array(self.hdf5_file[period]['wind'][sensor]['magnitude'])
        wind_direction = np.array(self.hdf5_file[period]['wind'][sensor]['direction'])

        return wind_magnitude, wind_direction
```

```
def load_wind_stat_data(self, period: str, timeseries_length: int, timeseries_num: int):
    """
    Loads wind statistical data for a given period and time series.

    Args:
        period (str): The period of data to load.
        timeseries_length (int): The length of each time series in minutes.
        timeseries_num (int): The index of the time series.

    Returns:
        Union[Tuple[float, float, float], bool]: Tuple containing mean wind speed, max wind speed,
        and mean wind direction, or False if all channels are not included.
    """

    # Check if all channels are included
    if not set(self.wind_sensors).issubset(list(self.hdf5_file[period]['wind'].keys())):
        return False

    wind_magnitude, wind_direction = self.load_wind(period, 'W07-28-1')

    fs = self.hdf5_file[period]['wind']['W07-28-1'].attrs['samplerate']

    time_series_wind_magnitude = wind_magnitude[timeseries_num * timeseries_length * fs *
                                                60:(timeseries_num + 1) * timeseries_length * fs * 60]
    time_series_wind_direction = wind_direction[timeseries_num * timeseries_length * fs *
                                                60:(timeseries_num + 1) * timeseries_length * fs * 60]
    mean_wind_speed = np.mean(time_series_wind_magnitude)
    max_wind_speed = np.max(time_series_wind_magnitude)
    mean_wind_direction = np.mean(time_series_wind_direction)

    return mean_wind_speed, max_wind_speed, mean_wind_direction

def load_temp(self, period: str, sensor: str):
    """
    Loads temperature measurements for a given period and sensor.

    Args:
        period (str): The period of data to load.
        sensor (str): The sensor from which to load the data.

    Returns:
        np.ndarray: Array containing temperature data.
    """

    # Temperature measurements has a 0.25 Hz sampling rate

    temp_data = np.array(self.hdf5_file[period]['temperature'][sensor]['x'])

    return temp_data

def load_temp_stat_data(self, period: str, timeseries_length: int, timeseries_num: int):
    """
    Loads temperature statistical data for a given period and time series.

    Args:
        period (str): The period of data to load.
        timeseries_length (int): The length of each time series in minutes.
        timeseries_num (int): The index of the time series.

    Returns:
        Union[float, bool]: The mean temperature for the specified time series,
        or False if all channels are not included.
    """

    # Check if all channels are included
    if not set(self.temp_sensors).issubset(list(self.hdf5_file[period]['temperature'].keys())):
        return False
```

```
        # Pick one temp sensor to collect data from
        all_temp_data = self.load_temp(period, 'T07-1')

        fs = self.hdf5_file[period]['temperature']['T07-1'].attrs['samplerate']

        time_series_temp_data = all_temp_data[timeseries_num * timeseries_length *
                                int(fs * 60):(timeseries_num + 1) * timeseries_length * int(fs * 60)]
        mean_temp = np.mean(time_series_temp_data)

        return mean_temp


class Mode:
    """
    Mode class
    """

    def __init__(self, frequency, mode_shape, damping=None, mode_type=None):
        """
        Initialize a class instance representing a mode.

        Args:
            frequency (float): The frequency of the mode.
            mode_shape (ndarray): The mode shape associated with the mode.
            damping (float, optional): The damping ratio of the mode. Defaults to None.
            mode_type (str, optional): The type of mode. Defaults to None.

        Attributes:
            frequency (float): The frequency of the mode.
            damping (float or None): The damping ratio of the mode.
            mode_shape (ndarray): The mode shape associated with the mode.
            mode_type (str or None): The type of mode.
            delta_f (None): Placeholder for a calculated value, set to None by default.
            mac_1 (None): Placeholder for a calculated value, set to None by default.
        """
        self.frequency = frequency
        self.damping = damping
        self.mode_shape = mode_shape
        self.mode_type = mode_type
        self.delta_f = None
        self.mac_1 = None


class HDF5_result_loader:
    """
    A dataloader specified to load logs from AOMA analysis stored in a h5 format.
    """

    def __init__(self, path: str):
        """
        Initializes an instance of the class.

        Args:
            path (str): The path to the HDF5 file.

        Attributes:
            path (str): The path to the HDF5 file.
            hdf5_file (h5py.File): The HDF5 file object.
            periods (list): The list of periods in the HDF5 file.
            features (list): The list of features available in the HDF5 file.
        """
        self.path = path
        self.hdf5_file = h5py.File(self.path, 'r')
        self.periods = list(self.hdf5_file.keys())
        self.features = ['Damping', 'Frequencies', 'Modeshape']

    def get_modes_in_period(self, period):
        """
        Retrieves the modes present in a specific period.
```

```
        Args:
            period (str): The period from which to retrieve the modes.

        Returns:
            List[Mode]: A list of Mode objects representing the modes in the specified period.
        """

        freqs = np.array(self.hdf5_file[period]['Frequencies'])
        dampings = np.array(self.hdf5_file[period]['Damping'])
        mode_shapes = np.array(self.hdf5_file[period]['Modeshape'])

        modes_in_period = []

        for i in range(len(freqs)):
            mode = Mode(freqs[i], mode_shapes[i], dampings[i])
            modes_in_period.append(mode)

        return modes_in_period

    def get_modes_all_periods(self):
        """
        Retrieves the modes for all periods.

        Returns:
            List[List[Mode]]: A nested list of Mode objects representing the modes in each period.
        """

        all_modes = []

        for period in self.periods:
            all_modes.append(self.get_modes_in_period(period))

        return all_modes

    def get_statistics(self):
        """
        Retrieves statistics for each period.

        Returns:
            Tuple[np.ndarray, np.ndarray, np.ndarray, np.ndarray, np.ndarray]:
            A tuple containing arrays of temperatures, mean wind speeds, max wind speeds,
            mean wind directions, and execution times for each period.
        """

        temps = []
        mean_wind_speed = []
        max_wind_speed = []
        mean_wind_direction = []
        execution_time = []

        for period in self.periods:
            temps.append(self.hdf5_file[period].attrs['Mean temp'])
            mean_wind_speed.append(self.hdf5_file[period].attrs['Mean wind speed'])
            max_wind_speed.append(self.hdf5_file[period].attrs['Max wind speed'])
            mean_wind_direction.append(self.hdf5_file[period].attrs['Mean wind direction'])
            execution_time.append(self.hdf5_file[period].attrs['Execution time'])

        temps = np.array(temps)
        mean_wind_speed = np.array(mean_wind_speed)
        max_wind_speed = np.array(max_wind_speed)
        mean_wind_direction = np.array(mean_wind_direction)
        execution_time = np.array(execution_time)

        return temps, mean_wind_speed, max_wind_speed, mean_wind_direction, execution_time

    def get_detection_statistics(self):
        """
        Retrieves detection statistics for the estimated modes in each period.
```

```
        Returns:
            Tuple[Dict[str, float], matplotlib.figure.Figure]: A tuple containing a dictionary with average,
            standard deviation, maximum, and minimum values of the number of estimated modes,
            and the matplotlib Figure object of the histogram.
        """

        modes_in_period = []

        for period in self.periods:
            modes_in_period.append(len(self.get_modes_in_period(period)))

        modes_in_period = np.array(modes_in_period)
        avg = np.mean(modes_in_period)
        max = np.max(modes_in_period)
        min = np.min(modes_in_period)
        std = np.std(modes_in_period)

        fig, ax = plt.subplots(figsize=(6, 4), dpi=300)
        ax.hist(modes_in_period, max - min)
        ax.set_xticks(np.arange(min, max + 1, step=2))
        ax.set_xlabel('Number of estimated modes in time series')
        ax.set_ylabel('Number of time series')
        plt.grid()

        ax.xaxis.set_major_locator(plt.MultipleLocator(2))
        ax.xaxis.set_minor_locator(plt.MultipleLocator(1))

        return {'avg': avg, 'std': std, 'max': max, 'min': min}, fig


class FEM_result_loader:
    """
    A dataloader specified to load modal parameters obtained from a FE-model created in Abaqus and
    exported to a h5 file format.
    """

    def __init__(self, path: str):
        """
          Initializes an instance of the class.

          Args:
              path (str): The path to the HDF5 file.

          Attributes:
              path (str): The path to the HDF5 file.
              hf (h5py.File): The HDF5 file object.
              deck_modes_idx (np.array): The indices of the manually picked bridge deck modes from the Abaqus
↪   model.
              mode_type (list): The type of each mode.
              sensor_labels (list): The labels of the sensors.
              phi_label_temp (np.array): Temporary array of phi labels.
              phi_label (list): The phi labels.
              sensor_indexes (list): Indexes of the sensor labels in phi_label.
              f (np.array): Frequencies of the deck modes.
              phi (np.array): Mode shapes of the deck modes.
              nodecoord (np.array): Node coordinates.
              node_deck (np.array): Array of node indexes in the bridge deck.
              index_node_deck (list): List index of nodes in the bridge deck.
              nodecoord_deck (np.array): Node coordinates of the bridge deck.
              index_y (list): List of indexes of y-DOFs in the bridge deck.
              index_z (list): List of indexes of z-DOFs in the bridge deck.
              index_t (list): List of indexes of t-DOFs in the bridge deck.
              phi_y (np.array): Mode shapes corresponding to y-DOFs.
              phi_z (np.array): Mode shapes corresponding to z-DOFs.
              phi_t (np.array): Mode shapes corresponding to t-DOFs.
              x_plot (np.array): x-coordinates of deck nodes.
          """
        self.path = path
```

```
        self.hf = h5py.File(self.path, 'r')

        # manually picked bridge deck modes from Abaqus model
        self.deck_modes_idx = np.array([1, 2, 3, 4, 5, 6, 7, 11, 12, 13, 14, 15, 16, 19, 23, 25, 32, 33,
                                        34, 40, 45, 48, 50, 58]) - 1

        self.mode_type = ['Horizontal', 'Vertical', 'Horizontal', 'Vertical', 'Vertical',
                          'Vertical', 'Horizontal', 'Cable', 'Vertical', 'Vertical', 'Horizontal',
                          'Vertical', 'Torsional', 'Cable', 'Cable', 'Vertical', 'Torsional', 'Horizontal',
                          'Vertical', 'Vertical', 'Vertical', 'Torsional', 'Vertical', 'Vertical']

        self.sensor_labels = ['3080_U1', '2080_U1', '3140_U1', '2140_U1', '3200_U1',
                              '2200_U1', '3240_U1', '2240_U1', '3290_U1', '2290_U1',
                              '3340_U1', '2340_U1', '3420_U1', '2420_U1', '3500_U1',
                              '2500_U1',
                              '3080_U2', '2080_U2', '3140_U2', '2140_U2', '3200_U2',
                              '2200_U2', '3240_U2', '2240_U2', '3290_U2', '2290_U2',
                              '3340_U2', '2340_U2', '3420_U2', '2420_U2', '3500_U2',
                              '2500_U2',
                              '3080_U3', '2080_U3', '3140_U3', '2140_U3', '3200_U3',
                              '2200_U3', '3240_U3', '2240_U3', '3290_U3', '2290_U3',
                              '3340_U3', '2340_U3', '3420_U3', '2420_U3', '3500_U3',
                              '2500_U3']

        phi_label_temp = np.array(self.hf.get('phi_label'))
        phi_label = phi_label_temp[:].astype('U10').ravel().tolist()

        sensor_indexes = []
        for label in self.sensor_labels:
            if label in phi_label:
                sensor_indexes.append(phi_label.index(label))

        self.f = np.array(self.hf.get('f'))[self.deck_modes_idx]
        phi = np.array(self.hf.get('phi'))[:, self.deck_modes_idx]
        self.phi = phi[sensor_indexes, :]

        nodecoord = np.array(self.hf.get('nodecoord'))
        node_deck = np.arange(1004, 1576 + 1, 1)
        # Create list index of nodes in bridge deck
        index_node_deck = []

        for k in np.arange(len(node_deck)):
            index_node_deck.append(np.argwhere(node_deck[k] == nodecoord[:, 0])[0, 0])

        nodecoord_deck = nodecoord[index_node_deck, :]

        # Create list of index of y-DOFs,z-DOFs, and t-DOFs in bridge deck
        index_y = []
        index_z = []
        index_t = []

        for k in np.arange(len(node_deck)):
            str_y = str(node_deck[k]) + '_U2'
            index_y.append(phi_label.index(str_y))

            str_z = str(node_deck[k]) + '_U3'
            index_z.append(phi_label.index(str_z))

            str_t = str(node_deck[k]) + '_UR1'
            index_t.append(phi_label.index(str_t))

        self.phi_y = phi[index_y, :]
        self.phi_z = phi[index_z, :]
        self.phi_t = phi[index_t, :]
        self.x_plot = nodecoord_deck[:, 1]  # x-coordinate of deck nodes

    def get_all_modes(self):
        """
        Retrieves all modes from the HDF5 file.
```

```
    Returns:
        list: A list of Mode objects representing the modes.
    """

    modes = []

    for i in range(len(self.f)):
        mode = Mode(self.f[i], self.phi[:, i], mode_type=self.mode_type[i])
        modes.append(mode)

    return modes
```

AOMA.py

```
import numpy as np
import h5py
import src.AOMA.dataloader as dl
from src.AOMA.plot import stabilization_diagram
import os
import koma.oma
import koma.clustering
import strid
from time import time
from datetime import datetime, timedelta
import warnings

# This is the main script for running AOMA on vibration data from the Hålogaland bridge

np.warnings.filterwarnings('ignore', category=np.VisibleDeprecationWarning)
warnings.filterwarnings('ignore', category=RuntimeWarning)

analysis_length = 30  # minutes
cutoff_frequency = 1  # Hz
bridgedeck_only = True

loader = dl.HDF5_dataloader(os.getcwd()+'/../../../../../../../Volumes/LaCie/Halogaland_sixth_try.hdf5',
                            bridgedeck_only=bridgedeck_only)

output_path = os.getcwd() + '/../../output/logs/output_AOMA_normal.h5'

# Hyperparameters
i = 50  # number of block rows
s = 1
fs = 2
orders = np.arange(2, 200+2, 2)  # orders to perform system
stabcrit = {'freq': 0.2, 'damping': 0.2, 'mac': 0.5}  # Default
prob_threshold = 0.99
min_cluster_size = 50
min_samples = 20
scaling={'mac':1.0, 'lambda_real':1.0, 'lambda_imag': 1.0}

# Write hyperparameters as attributes to output file
with h5py.File(output_path, 'a') as hdf:
    hdf.attrs['i'] = i
    hdf.attrs['s'] = s
    hdf.attrs['order'] = np.max(orders)
    hdf.attrs['stabcrit_freq'] = stabcrit['freq']
    hdf.attrs['stabcrit_damping'] = stabcrit['damping']
    hdf.attrs['stabcrit_mac'] = stabcrit['mac']
    hdf.attrs['prob_threshold'] = prob_threshold
    hdf.attrs['min_cluster_size'] = min_cluster_size
    hdf.attrs['min_samples'] = min_samples
    hdf.attrs['scaling_mac'] = scaling['mac']
    hdf.attrs['scaling_lambda_real'] = scaling['lambda_real']
    hdf.attrs['scaling_lambda_imag'] = scaling['lambda_imag']
    hdf.attrs['analysis_length [min]'] = analysis_length
    hdf.attrs['cutoff_frequency'] = cutoff_frequency
    hdf.attrs['bridgedeck_only'] = bridgedeck_only

if bridgedeck_only:
    ix_references_y = (np.array([0, 2, 4, 6, 8, 10, 12, 14]) + 16)
    ix_references_z = (np.array([0, 2, 4, 6, 8, 10, 12, 14]) + 32)
    ix_references = np.concatenate((ix_references_y, ix_references_z)).tolist()
else:  # with hangers in addition
    ix_references_y = (np.array([0, 2, 4, 6, 8, 10, 12, 14]) + 20)
    ix_references_z = (np.array([0, 2, 4, 6, 8, 10, 12, 14]) + 40)
    ix_references = np.concatenate((ix_references_y, ix_references_z)).tolist()


number_of_periods = len(loader.periods)
print("Number of periods to run " + str(number_of_periods))
```

```
number_in_sample = fs*60*analysis_length

skipped = 0
for period in range(number_of_periods-44):
    period = period + 44
    acc = loader.load_all_acceleration_data(loader.periods[period], preprosess=True,
                                            cutoff_frequency=cutoff_frequency, filter_order=10)

    # If all channels are present, proceed with split up in intervals and perform Cov-SSI and clustering,
    # if not, move to the next period
    if isinstance(acc, np.ndarray):
        acc = np.array_split(acc, acc.shape[0]/number_in_sample)

        for j in range(len(acc)):

            t0 = time()  # Start timer of computation process

            # Cov-SSI
            ssid = strid.CovarianceDrivenStochasticSID(acc[j].transpose(), fs) #, ix_references)
            modes = {}
            for order in orders:
                A, C, G, R0 = ssid.perform(order, i)
                modes[order] = strid.Mode.find_modes_from_ss(A, C, ssid.fs)

            # Sorting routine
            lambdas = []
            phis = []

            for order in modes.keys():
                modes_in_order = modes[order]
                lambdas_in_order = []
                phis_in_order = []
                for mode in modes_in_order:
                    lambdas_in_order.append(mode.eigenvalue)
                    phis_in_order.append(mode.eigenvector)
                lambdas.append(np.array(lambdas_in_order))
                phis.append(np.array(phis_in_order).transpose())

            # Stabilization analysis
            lambd_stab, phi_stab, orders_stab, idx_stab = koma.oma.find_stable_poles(lambdas, phis, orders, s,
                                    stabcrit=stabcrit, valid_range={'freq': [0.05, np.inf], 'damping':[0, 0.2]},
                                    indicator='freq', return_both_conjugates=False)

            # HDBSCAN
            pole_clusterer = koma.clustering.PoleClusterer(lambd_stab, phi_stab, orders_stab,
                            min_cluster_size=min_cluster_size,
                            min_samples=min_samples, scaling=scaling)

            args = pole_clusterer.postprocess(prob_threshold=prob_threshold, normalize_and_maxreal=True)

            xi_auto, omega_n_auto, phi_auto, order_auto, probs_auto, ixs_auto = \
                koma.clustering.group_clusters(*args)

            xi_mean = np.array([np.median(xi_i) for xi_i in xi_auto])
            fn_mean = np.array([np.median(om_i) for om_i in omega_n_auto])/2/np.pi

            xi_std = np.array([np.std(xi_i) for xi_i in xi_auto])
            fn_std = np.array([np.std(om_i) for om_i in omega_n_auto])/2/np.pi

            # Sort and arrange modeshapes
            lambd_used, phi_used, order_stab_used, group_ixs, all_single_ix, probs = \
                pole_clusterer.postprocess(prob_threshold=prob_threshold)

            grouped_phis = koma.clustering.group_array(phi_used, group_ixs, axis=1)

            phi_extracted = np.zeros((len(grouped_phis), len(loader.acceleration_sensors)*3))

            for a in range(len(grouped_phis)):
                for b in range(np.shape(grouped_phis[a])[0]):
```

```
                phi_extracted[a, b] = (np.real(np.median(grouped_phis[a][b])))


        # Load environmental statistical data for analyzed time series
        mean_wind_speed, max_wind_speed, mean_wind_direction = \
            loader.load_wind_stat_data(loader.periods[period], analysis_length, j)
        mean_temp = loader.load_temp_stat_data(loader.periods[period], analysis_length, j)

        t1 = time()  # end timer of computation process
        print("Time serie " + str(j+1) + " of " + str(len(acc)) + " done in " + str(t1-t0) + " sec. Period " +
              str(period+1) + " of " + str(number_of_periods) +
              " done. Number of skipped periods: " + str(skipped)+".")

        # Prepare timestamp of the time series in process
        timestamp = (datetime.strptime(loader.periods[period], "%Y-%m-%d-%H-%M-%S") +
                     timedelta(minutes=j*analysis_length)).strftime("%Y-%m-%d-%H-%M-%S")

        # Save stabilization plot
        stab_diag = stabilization_diagram(acc[j], fs, 2, (np.array(omega_n_auto) / 2 / np.pi),
                                          np.array(order_auto),
                                          all_freqs=np.abs(lambd_stab) / 2 / np.pi, all_orders=orders_stab)

        # Write logs to h5 file
        with h5py.File(output_path, 'a') as hdf:
            G1 = hdf.create_group(timestamp)

            # Write logs
            G1.create_dataset('Frequencies', data=fn_mean)
            G1.create_dataset('Damping', data=xi_mean)
            G1.create_dataset('Modeshape', data=phi_extracted)

            # Write attributes
            G1.attrs['Mean wind speed'] = mean_wind_speed
            G1.attrs['Max wind speed'] = max_wind_speed
            G1.attrs['Mean wind direction'] = mean_wind_direction
            G1.attrs['Mean temp'] = mean_temp
            G1.attrs['Execution time'] = (t1-t0)
            G1.attrs['Std of acceleration data'] = np.mean(np.std(acc[j], axis=0))

    else:
        skipped += 1
        print("One or more channels are missing, period skipped.")
```

trace.py

```
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.ticker as ticker
import dataloader as dl
from strid.utils import modal_assurance_criterion
import scipy as sp
import koma.modal as modal




class ModeTrace:
    """
    Mode trace object
    """
    def __init__(self, reference_modes: list[dl.Mode], numb_analysis,
                 simcrit = {'freq': 0.4, 'mac': 0.5}):
        """
        Initializes an instance of the ModeComparison class.

        Args:
            reference_modes (list[dl.Mode]): A list of reference modes to compare against.
            numb_analysis (int): The number of analysis modes to compare.
            simcrit (dict, optional): A dictionary specifying the similarity criteria for comparison.
                Defaults to {'freq': 0.4, 'mac': 0.5}.
        """

        self.reference_modes = {}
        for i, mode in enumerate(reference_modes):
            new_mode = {i:mode}
            self.reference_modes.update(new_mode)

        self.numb_analysis = numb_analysis
        self.mode_trace = np.empty(shape=[len(self.reference_modes), self.numb_analysis], dtype=object)
        self.simcrit = simcrit
        self.mode_type = ['Horizontal', 'Vertical', 'Horizontal', 'Vertical', 'Vertical',
                          'Vertical', 'Horizontal', 'Cable', 'Vertical', 'Vertical', 'Horizontal',
                          'Vertical', 'Torsional', 'Cable', 'Cable', 'Vertical', 'Torsional', 'Horizontal',
                          'Vertical', 'Vertical', 'Vertical', 'Torsional', 'Vertical', 'Vertical']

    def add_modes_from_period(self, candidate_modes: list[dl.Mode], period: int):
        """
        Adds candidate modes to the mode trace for a specific period, based on similarity criteria.

        Args:
            candidate_modes (list[dl.Mode]): A list of candidate modes to be added.
            period (int): The period associated with the candidate modes.

        Returns:
            None
        """
        candidate_modes = candidate_modes
        f_tol = self.simcrit['freq']
        mac_tol = self.simcrit['mac']
        if len(candidate_modes[0].mode_shape) > 48:
            indexes = [0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,20,21,
                       22,23,24,25,26,27,28,29,30,31,32,33,34,35,40,
                       41,42,43,44,45,46,47,48,49,50,51,52,53,54,55]
        else:
            indexes = np.arange(0, 48, 1)

        for key, ref_mode in self.reference_modes.items():
            for candidate in candidate_modes:
                delta_f = np.abs(ref_mode.frequency - candidate.frequency)/\
                                np.max([ref_mode.frequency, candidate.frequency])
                mac_1 = 1 - modal_assurance_criterion(ref_mode.mode_shape, candidate.mode_shape[indexes])
                if delta_f < f_tol and mac_1 < mac_tol:
```

```
                if not isinstance(self.mode_trace[key, period], dl.Mode):
                    self.mode_trace[key, period] = candidate
                    candidate_modes.remove(candidate)
                else:
                    competitor = self.mode_trace[key, period]
                    comp_delta_f = np.abs(ref_mode.frequency - competitor.frequency)/\
                                np.max([ref_mode.frequency, competitor.frequency])
                    comp_mac_1 = 1 - modal_assurance_criterion(ref_mode.mode_shape,
                    ↪  competitor.mode_shape[indexes])
                    if (delta_f < comp_delta_f and
                            mac_1 < comp_mac_1):
                        self.mode_trace[key, period] = candidate
                        candidate_modes.remove(candidate)
                        candidate_modes.append(competitor)

    def add_all_modes(self, all_modes):
        """
        Adds all modes from a list of modes to the mode trace, corresponding to their respective periods.

        Args:
            all_modes (list): A list containing modes for each period.

        Returns:
            None
        """
        for i in range(len(all_modes)):
            self.add_modes_from_period(all_modes[i], period=i)

    def get_frequencies_from_trace(self, trace):
        """
        Retrieves the frequencies of modes from a specific trace in the mode trace.

        Args:
            trace (int): The index of the trace to retrieve frequencies from.

        Returns:
            list: A list of tuples containing the index and frequency of each mode in the trace.
        """
        traces = self.mode_trace[trace,:]

        freqs = []

        for i in range(len(traces)):
            if isinstance(traces[i], dl.Mode):
                freqs.append((i, traces[i].frequency))

        return freqs

    def get_damping_from_trace(self, trace):
        """
        Retrieves the damping values of modes from a specific trace in the mode trace.

        Args:
            trace (int): The index of the trace to retrieve damping values from.

        Returns:
            list: A list of tuples containing the index and damping value of each mode in the trace.
        """
        traces = self.mode_trace[trace,:]

        damps = []

        for i in range(len(traces)):
            if isinstance(traces[i], dl.Mode):
                damps.append((i, traces[i].damping))

        return damps

    def plot_frequency_distribution(self):
```

```
        """
        Plots the frequency distribution of the modes.

        Returns:
            matplotlib.figure.Figure: The generated figure object.
        """
        fig, axs = plt.subplots(6, 4, figsize=(20, 20), dpi=300)
        axs = axs.ravel()
        remove = 0
        for i, ax in enumerate(axs):
            if i < len(self.reference_modes):
                freqs = self.get_frequencies_from_trace(i)
                ax.set_title('Mode ' + str(i + 1) + ' - ' + str(self.mode_type[i]))
                ax.set_xlabel('f [Hz]')
                if len(freqs) == 0:
                    continue
                freqs_mode = np.array(list(zip(*freqs))[1])
                ax.hist(freqs_mode, 20, label='AOMA')
                ax.xaxis.set_major_formatter(ticker.FormatStrFormatter('%0.3f'))
                text = 'n = ' + str(len(freqs_mode)) + '\nr = ' + \
                       f"{(100*len(freqs_mode)/self.numb_analysis):.1f}" + '%'
                ax.text(0.72, 0.95, text, transform=ax.transAxes, fontsize=14, verticalalignment='top',
                        bbox= dict(boxstyle='round', facecolor='white'))
            else:
                remove += 1
                continue
        fig.suptitle('Frequency distribution', fontsize=14, y=0.99)
        fig.tight_layout()
        while remove > 0:
            fig.delaxes(axs[-remove])
            remove -= 1

        return fig

    def plot_damping_distribution(self):
        """
        Plots the damping distribution of the modes.

        Returns:
            matplotlib.figure.Figure: The generated figure object.
        """
        fig, axs = plt.subplots(6, 4, figsize=(20, 25), dpi=300)
        axs = axs.ravel()
        for i, ax in enumerate(axs):
            damp = self.get_damping_from_trace(i)
            ax.set_title('Mode ' + str(i + 1)+ ' - ' + str(self.mode_type[i]))
            ax.set_xlabel('$\\xi$ [%]')
            if len(damp) == 0:
                continue
            damp_mode = np.array(list(zip(*damp))[1])*100
            ax.hist(damp_mode, 20)
            ax.xaxis.set_major_formatter(ticker.FormatStrFormatter('%0.3f'))
        fig.suptitle('Damping distribution', fontsize=20, y=0.99)
        fig.tight_layout()

        return fig

    def plot_freq_vs_temp_corr(self, temps):
        """
        Plots the correlation between mode frequencies and temperatures.

        Args:
            temps (numpy.ndarray): Array of temperatures.

        Returns:
            matplotlib.figure.Figure: The generated figure object.
        """
        fig, axs = plt.subplots(6, 4, figsize=(20, 25), dpi=300)
        axs = axs.ravel()
```

```
        remove = 0
        for i, ax in enumerate(axs):
            if i < len(self.reference_modes):
                freqs = self.get_frequencies_from_trace(i)
                ax.set_title('Mode ' + str(i + 1)+ ' - ' + str(self.mode_type[i]))
                ax.set_xlabel('f [Hz]')
                ax.set_ylabel('Temperature [$^\circ$C]')
                if len(freqs) == 0:
                    continue
                indexes = np.array(list(zip(*freqs))[0])
                temps_for_mode = temps[indexes]
                freqs_mode = np.array(list(zip(*freqs))[1])

                # Find regression line
                b, a = np.polyfit(np.array(freqs_mode), np.array(temps_for_mode), deg=1)
                xseq = np.linspace(0, 10, num=len(temps_for_mode))

                # Pearson correlation coefficient
                corr = sp.stats.linregress(np.array(freqs_mode), np.array(temps_for_mode))

                ax.scatter(np.array(freqs_mode), np.array(temps_for_mode), alpha=0.7)
                ax.plot(xseq, a + b * xseq, color='red')
                ax.set_xlim([np.mean(freqs_mode) - np.std(freqs_mode) * 5,
                             np.mean(freqs_mode) + np.std(freqs_mode) * 5])
                ax.xaxis.set_major_formatter(ticker.FormatStrFormatter('%0.3f'))
                ax.set_ylim([-15, 10])
                ax.set_xlim([np.min(freqs_mode), np.max(freqs_mode)])
                text = '$r$ = ' + f"{corr.rvalue:.2f}"
                ax.text(0.72, 0.95, text, transform=ax.transAxes, fontsize=14, verticalalignment='top',
                        bbox=dict(boxstyle='round', facecolor='white'))

        fig.suptitle('Frequency vs temperature correlation', fontsize=20, y=0.99)
        fig.tight_layout()
        while remove > 0:
            fig.delaxes(axs[-remove])
            remove -= 1

        return fig

    def plot_damp_vs_wind_corr(self, wind):
        """
        Plots the correlation between damping and mean wind speed.

        Args:
            wind (numpy.ndarray): Array of mean wind speeds.

        Returns:
            matplotlib.figure.Figure: The generated figure object.
        """
        fig, axs = plt.subplots(6, 4, figsize=(20, 25), dpi=300)
        axs = axs.ravel()
        remove = 0
        for i, ax in enumerate(axs):
            if i < len(self.reference_modes):
                damp = self.get_damping_from_trace(i)
                ax.set_title('Mode ' + str(i + 1)+ ' - ' + str(self.mode_type[i]))
                ax.set_xlabel('$\\xi$ [%]')
                ax.set_ylabel('Mean wind speed [m/s]')
                if len(damp) < 1:
                    continue
                indexes = np.array(list(zip(*damp))[0])
                wind_for_mode = wind[indexes]
                damp_mode = np.array(list(zip(*damp))[1])*100

                # Find regression line
                b, a = np.polyfit(np.array(damp_mode), np.array(wind_for_mode), deg=1)
                xseq = np.linspace(0, 10, num=len(wind_for_mode))

                # Pearson correlation coefficient
```

```
            corr = sp.stats.linregress(np.array(damp_mode), np.array(wind_for_mode))

            ax.scatter(np.array(damp_mode), np.array(wind_for_mode), alpha=0.7)
            ax.plot(xseq, a + b * xseq, color='red')
            ax.set_xlim([0, np.mean(damp_mode) + np.std(damp_mode) * 5])
            ax.xaxis.set_major_formatter(ticker.FormatStrFormatter('%0.2f'))
            ax.set_ylim([0, 30])
            text = '$r$ = ' + f"{corr.rvalue:.2f}"
            ax.text(0.72, 0.95, text, transform=ax.transAxes, fontsize=14, verticalalignment='top',
                    bbox=dict(boxstyle='round', facecolor='white'))

    fig.suptitle('Damping vs wind correlation', fontsize=20, y=0.99)
    fig.tight_layout()
    while remove > 0:
        fig.delaxes(axs[-remove])
        remove -= 1

    return fig

def plot_AOMA_vs_FEM(self):
    """
    Plots the AOMA frequencies versus FEM frequencies for each mode type.

    Returns:
        matplotlib.figure.Figure: The generated figure object.
    """

    fig, axs = plt.subplots(2, 2, figsize=(10, 10), dpi=300)

    for i in range(len(self.reference_modes)):

        if self.mode_type[i] == 'Horizontal':
            axs[0, 0].plot(self.reference_modes[i].frequency,
                           np.mean(np.array(self.get_frequencies_from_trace(i)[:])[:, 1]), marker='o', color='r')
            axs[0, 0].plot([0, 1], [0, 1], transform=axs[0, 0].transAxes, color='black')
            axs[0, 0].set_xlabel('$f_{FEM}$ [Hz]')
            axs[0, 0].set_ylabel('$f_{AOMA}$ [Hz]')
            axs[0, 0].set_title(self.mode_type[i])
            axs[0, 0].set_xticks(np.arange(0, 1.2, step=0.2))
            axs[0, 0].set_yticks(np.arange(0, 1.2, step=0.2))
            axs[0, 0].set_xlim([0, 1])
            axs[0, 0].set_ylim([0, 1])

        elif self.mode_type[i] == 'Vertical':
            axs[0, 1].plot(self.reference_modes[i].frequency,
                           np.mean(np.array(self.get_frequencies_from_trace(i)[:])[:, 1]), marker='o', color='r')
            axs[0, 1].plot([0, 1], [0, 1], transform=axs[0, 1].transAxes, color='black')
            axs[0, 1].set_xlabel('$f_{FEM}$ [Hz]')
            axs[0, 1].set_ylabel('$f_{AOMA}$ [Hz]')
            axs[0, 1].set_title(self.mode_type[i])
            axs[0, 1].set_xticks(np.arange(0, 1.2, step=0.2))
            axs[0, 1].set_yticks(np.arange(0, 1.2, step=0.2))
            axs[0, 1].set_xlim([0, 1.1])
            axs[0, 1].set_ylim([0, 1.1])

        elif self.mode_type[i] == 'Torsional':
            axs[1, 0].plot(self.reference_modes[i].frequency,
                           np.mean(np.array(self.get_frequencies_from_trace(i)[:])[:, 1]), marker='o', color='r')
            axs[1, 0].plot([0, 1], [0, 1], transform=axs[1, 0].transAxes, color='black')
            axs[1, 0].set_xlabel('$f_{FEM}$ [Hz]')
            axs[1, 0].set_ylabel('$f_{AOMA}$ [Hz]')
            axs[1, 0].set_title(self.mode_type[i])
            axs[1, 0].set_xticks(np.arange(0, 1.2, step=0.2))
            axs[1, 0].set_yticks(np.arange(0, 1.2, step=0.2))
            axs[1, 0].set_xlim([0, 1])
            axs[1, 0].set_ylim([0, 1])

        elif self.mode_type[i] == 'Cable':
            axs[1, 1].plot(self.reference_modes[i].frequency,
```

```
                        np.mean(np.array(self.get_frequencies_from_trace(i)[:])[:, 1]), marker='o', color='r')
            axs[1, 1].plot([0, 1], [0, 1], transform=axs[1, 1].transAxes, color='black')
            axs[1, 1].set_xlabel('$f_{FEM}$ [Hz]')
            axs[1, 1].set_ylabel('$f_{AOMA}$ [Hz]')
            axs[1, 1].set_title(self.mode_type[i])
            axs[1, 1].set_xticks(np.arange(0, 1.2, step=0.2))
            axs[1, 1].set_yticks(np.arange(0, 1.2, step=0.2))
            axs[1, 1].set_xlim([0, 1])
            axs[1, 1].set_ylim([0, 1])

        return fig


def plotModeShapeAOMA(tracer: ModeTrace, FEM_loader: dl.FEM_result_loader, type ='Vertical'):
    """
    Plots the mode shapes for the given ModeTrace and FEM_result_loader objects.

    Args:
        tracer (ModeTrace): ModeTrace object containing mode information.
        FEM_loader (dl.FEM_result_loader): FEM_result_loader object containing FEM results.
        type (str, optional): Type of mode shape to plot. Defaults to 'Vertical'.

    Returns:
        fig: The generated matplotlib Figure object.
    """

    all_modeshapes = np.array(np.empty([tracer.mode_trace.shape[0], tracer.mode_trace.shape[1],
                                    len(tracer.mode_trace[0,0].mode_shape)]), dtype=np.float)

    for i in range(tracer.mode_trace.shape[0]):
        for j in range(tracer.mode_trace.shape[1]):
            if isinstance(tracer.mode_trace[i,j], dl.Mode):
                all_modeshapes[i, j, :] = tracer.mode_trace[i, j].mode_shape

    f_mean = []
    xi_mean = []
    ref_phi = np.zeros([48, len(tracer.reference_modes)])
    for i in range(len(tracer.reference_modes)):
        ref_phi[:, i] = tracer.reference_modes[i].mode_shape
        f_mean.append(np.mean(np.array(tracer.get_frequencies_from_trace(i)[:])[:, 1]))
        xi_mean.append(100*np.mean(np.array(tracer.get_damping_from_trace(i)[:])[:, 1]))

    num = tracer.mode_type.count(type)

    # Plot
    fig, axs = plt.subplots(int(np.ceil(num/2)), 2, figsize=(20, int(np.ceil(num/2))*3), dpi=300)
    x = np.array([-572.5, -420, -300, -180, -100, 0, 100, 260, 420,
                          572.5])  # Sensor x-coordinates - [TOWER, A03, A04, A05, A06, A07, A08, A09, A10, TOWER]
    B = 18.6  # Width of bridge girder

    phi_y_ref = ref_phi[16:32, :]
    phi_z_ref_temp = ref_phi[32:48, :]

    phi_y_ref = (modal.maxreal((phi_y_ref[::2, :] + phi_y_ref[1::2, :]) / 2))
    phi_z_ref = (modal.maxreal((phi_z_ref_temp[::2, :] + phi_z_ref_temp[1::2, :]) / 2))
    phi_t_ref = (modal.maxreal((- phi_z_ref_temp[::2, :] + phi_z_ref_temp[1::2, :]) / B))

    # Add plot of reference modes here
    f = FEM_loader.f
    phi_y_FEM = FEM_loader.phi_y
    phi_y_FEM[:, [6, 7, 14, 17]] = phi_y_FEM[:, [6, 7, 14, 17]]*(-1)
    phi_z_FEM = FEM_loader.phi_z
    phi_z_FEM[:, [1, 5, 11, 19, 20]] = phi_z_FEM[:, [1, 5, 11, 19, 20]]*(-1)
    phi_t_FEM = FEM_loader.phi_t*(-1)
    phi_t_FEM[:, [16, 21]] = phi_t_FEM[:, [16, 21]]*(-1)
    x_FEM = FEM_loader.x_plot

    j = 0
    for i in range(len(f)):
```

```
        axs[int(np.floor(j / 2)), j % 2].set_xlabel('x[m]')
        axs[int(np.floor(j / 2)), j % 2].set_ylim([-1, 1])
        axs[int(np.floor(j / 2)), j % 2].set_xlim([-600, 600])
        axs[int(np.floor(j / 2)), j % 2].set_xticks([-600, -300, 0, 300, 600])
        axs[int(np.floor(j / 2)), j % 2].set_yticks([-1, -0.5, 0, 0.5, 1])
        axs[int(np.floor(j / 2)), j % 2].vlines([-420, -300, -180, -100, 0, 100, 260, 420], -1, 1, color='grey',
                                               linestyles=':', alpha=0.5)

        if FEM_loader.mode_type[i] == 'Horizontal' and type == 'Horizontal':
            factor = 1 / np.max(np.abs(phi_y_FEM[:, i]))
            axs[int(np.floor(j / 2)), j % 2].plot(x_FEM, phi_y_FEM[:, i] * factor, color='black', alpha=0.5)
            j += 1
        elif FEM_loader.mode_type[i] == 'Vertical' and type == 'Vertical':
            factor = 1 / np.max(np.abs(phi_z_FEM[:, i]))
            axs[int(np.floor(j / 2)), j % 2].plot(x_FEM, phi_z_FEM[:, i] * factor, color='black', alpha=0.5)
            j += 1
        elif FEM_loader.mode_type[i] == 'Torsional' and type == 'Torsional':
            factor = 1 / np.max(np.abs(phi_t_FEM[:, i]))
            axs[int(np.floor(j / 2)), j % 2].plot(x_FEM, phi_t_FEM[:, i] * factor, color='black', alpha=0.5)
            j += 1
        elif FEM_loader.mode_type[i] == 'Cable' and type == 'Cable':
            factor = 1 / np.max(np.abs(phi_y_FEM[:, i]))
            axs[int(np.floor(j / 2)), j % 2].plot(x_FEM, phi_y_FEM[:, i] * factor, color='black', alpha=0.5)
            j += 1

    for a in range(all_modeshapes.shape[1]):

        phi = all_modeshapes[:, a, :].transpose()

        phi_x, phi_y, phi_z_temp = np.split(phi, 3, axis=0)
        phi_y = phi_y[:16, :]
        phi_z_temp = phi_z_temp[:16, :]

        phi_y = (modal.maxreal((phi_y[::2, :] + phi_y[1::2, :]) / 2))
        phi_z = (modal.maxreal((phi_z_temp[::2, :] + phi_z_temp[1::2, :]) / 2))
        phi_t = (modal.maxreal((-phi_z_temp[::2, :] + phi_z_temp[1::2, :]) / B))

        j = 0
        for i in range(len(tracer.reference_modes)):
            axs[int(np.floor(j / 2)), j % 2].set_xlabel('x[m]')
            axs[int(np.floor(j / 2)), j % 2].set_ylim([-1, 1])
            axs[int(np.floor(j / 2)), j % 2].set_xlim([-600, 600])
            axs[int(np.floor(j / 2)), j % 2].set_xticks([-600, -300, 0, 300, 600])
            axs[int(np.floor(j / 2)), j % 2].set_yticks([-1, -0.5, 0, 0.5, 1])


            if tracer.mode_type[i] == 'Horizontal' and type == 'Horizontal':
                factor = 1 / np.max(np.abs(phi_y[:, i]))
                factor_ref = 1 / np.max(np.abs(phi_y_ref[:, i]))

                if np.sum(np.abs(phi_y[:, i] - phi_y_ref[:, i]*factor_ref)) > 5.0:
                    phi_y[:, i] = phi_y[:, i]*(-1)

                axs[int(np.floor(j/2)), j % 2].plot(x, np.concatenate((np.array([0]), phi_y[:, i], np.array([0]))),
                                                    color='tab:red', alpha = 0.05)
                axs[int(np.floor(j / 2)), j % 2].set_title(
                    'Mode ' + str(i + 1) + ' - ' + type + '\n $\overline{f}_n$ = ' + f"{f_mean[i]:.3f}" + ' Hz, '
                                                          '$\overline{\\xi}_n$ = ' + f"{xi_mean[i]:.1f}"
                                                          ↪  +'%')
                axs[int(np.floor(j / 2)), j % 2].grid()
                j += 1
            elif tracer.mode_type[i] == 'Vertical' and type == 'Vertical':
                factor = 1 / np.max(np.abs(phi_z[:, i]))
                factor_ref = 1 / np.max(np.abs(phi_z_ref[:, i]))

                if np.sum(np.abs(phi_z[:, i] - phi_z_ref[:, i]*factor_ref)) > 5.0:
                    phi_z[:, i] = phi_z[:, i]*(-1)

                axs[int(np.floor(j/2)), j % 2].plot(x, np.concatenate((np.array([0]), phi_z[:, i], np.array([0]))),
```

```
                                                , color='tab:blue', alpha = 0.05)
                axs[int(np.floor(j / 2)), j % 2].set_title(
                    'Mode ' + str(i + 1) + ' - ' + type + '\n $\overline{f}_n$ = ' + f"{f_mean[i]:.3f}" + ' Hz,'
                                                          ' $\overline{\\xi}_n$ = ' + f"{xi_mean[i]:.1f}" +
                                                          ↪  '%')
                axs[int(np.floor(j / 2)), j % 2].grid()
                j += 1
            elif tracer.mode_type[i] == 'Torsional' and type == 'Torsional':
                factor = 1 / np.max(np.abs(phi_t[:, i]))
                factor_ref = 1 / np.max(np.abs(phi_t_ref[:, i]))

                if np.sum(np.abs(phi_t[:, i] - phi_t_ref[:, i]*factor_ref)) > 4.0:
                    phi_t[:, i] = phi_t[:, i]*(-1)

                axs[int(np.floor(j/2)), j % 2].plot(x, np.concatenate((np.array([0]), phi_t[:, i], np.array([0]))
                                                , color='tab:orange', alpha = 0.05)
                axs[int(np.floor(j / 2)), j % 2].set_title(
                    'Mode ' + str(i + 1) + ' - ' + type + '\n $\overline{f}_n$ = ' + f"{f_mean[i]:.3f}" + ' Hz,'
                                                          ' $\overline{\\xi}_n$ = ' + f"{xi_mean[i]:.1f}" +
                                                          ↪  '%')
                axs[int(np.floor(j / 2)), j % 2].grid()
                j += 1
            elif tracer.mode_type[i] == 'Cable' and type == 'Cable':
                factor = 1 / np.max(np.abs(phi_y[:, i]))
                factor_ref = 1 / np.max(np.abs(phi_y_ref[:, i]))

                if np.sum(np.abs(phi_y[:, i] - phi_y_ref[:, i]*factor_ref)) > 5.0:
                    phi_y[:, i] = phi_y[:, i]*(-1)

                axs[int(np.floor(j/2)), j % 2].plot(x, np.concatenate((np.array([0]), phi_y[:, i], np.array([0]))
                                                , color='tab:green', alpha = 0.05)
                axs[int(np.floor(j / 2)), j % 2].set_title(
                    'Mode ' + str(i + 1) + ' - ' + type + '\n $\overline{f}_n$ = ' + f"{f_mean[i]:.3f}" + ' Hz, '
                                                          '$\overline{\\xi}_n$ = ' + f"{xi_mean[i]:.1f}"
                                                          ↪  +'%')
                axs[int(np.floor(j / 2)), j % 2].grid()
                j += 1

    if num % 2:
        fig.delaxes(axs[int(np.ceil(num/2))-1, 1])

    fig.tight_layout()

    return fig


def plotSingleModeAllComponents(tracer: ModeTrace, FEM_loader: dl.FEM_result_loader, mode_i):
    """
    Plots the mode shapes of a single mode along with the reference modes.

    Args:
        tracer (ModeTrace): Object containing the mode trace information.
        FEM_loader (dl.FEM_result_loader): Object containing FEM results.
        mode_i (int): Index of the mode to be plotted.

    Returns:
        fig: Figure object containing the plot.
    """

    all_modeshapes = np.array(np.empty([tracer.mode_trace.shape[0], tracer.mode_trace.shape[1],
                                    len(tracer.mode_trace[0,0].mode_shape)]), dtype=np.float)

    for i in range(tracer.mode_trace.shape[0]):
        for j in range(tracer.mode_trace.shape[1]):
            if isinstance(tracer.mode_trace[i,j], dl.Mode):
                all_modeshapes[i, j, :] = tracer.mode_trace[i, j].mode_shape

    f_mean = []
    xi_mean = []
```

```
ref_phi = np.zeros([48, len(tracer.reference_modes)])
for i in range(len(tracer.reference_modes)):
    ref_phi[:, i] = tracer.reference_modes[i].mode_shape
    f_mean.append(np.mean(np.array(tracer.get_frequencies_from_trace(i)[:])[:, 1]))
    xi_mean.append(100*np.mean(np.array(tracer.get_damping_from_trace(i)[:])[:, 1]))

num = tracer.mode_type.count(type)

# Plot
fig, axs = plt.subplots(2, 1, figsize=(10,5), dpi=300)
x = np.array([-572.5, -420, -300, -180, -100, 0, 100, 260, 420,
                      572.5])  # Sensor x-coordinates - [TOWER, A03, A04, A05, A06, A07, A08, A09, A10, TOWER]
B = 18.6  # Width of bridge girder

phi_y_ref = ref_phi[16:32, :]
phi_z_ref_temp = ref_phi[32:48, :]

phi_y_ref = (modal.maxreal((phi_y_ref[::2, :] + phi_y_ref[1::2, :]) / 2))
phi_z_ref = (modal.maxreal((phi_z_ref_temp[::2, :] + phi_z_ref_temp[1::2, :]) / 2))
phi_t_ref = (modal.maxreal((- phi_z_ref_temp[::2, :] + phi_z_ref_temp[1::2, :]) / B))

# Add plot of reference modes here
f = FEM_loader.f
phi_y_FEM = FEM_loader.phi_y
phi_y_FEM[:, [6, 7, 14, 17]] = phi_y_FEM[:, [6, 7, 14, 17]]*(-1)
phi_z_FEM = FEM_loader.phi_z
phi_z_FEM[:, [1, 5, 11, 19, 20]] = phi_z_FEM[:, [1, 5, 11, 19, 20]]*(-1)
phi_t_FEM = FEM_loader.phi_t
phi_t_FEM[:, [16, 21]] = phi_t_FEM[:, [16, 21]]
x_FEM = FEM_loader.x_plot

for ax in axs:
    ax.set_xlabel('x[m]')
    ax.set_ylim([-1, 1])
    ax.set_xlim([-600, 600])
    ax.set_xticks([-600, -300, 0, 300, 600])
    ax.set_yticks([-1, -0.5, 0, 0.5, 1])
    ax.vlines([-420, -300, -180, -100, 0, 100, 260, 420], -1, 1, color='grey',
                                          linestyles=':', alpha=0.5)


for a in range(all_modeshapes.shape[1]):

    phi = all_modeshapes[:, a, :].transpose()

    phi_x, phi_y, phi_z_temp = np.split(phi, 3, axis=0)
    phi_y = phi_y[:16, :]
    phi_z_temp = phi_z_temp[:16, :]

    phi_y = (modal.maxreal((phi_y[::2, :] + phi_y[1::2, :]) / 2))
    phi_z = (modal.maxreal((phi_z_temp[::2, :] + phi_z_temp[1::2, :]) / 2))
    phi_t = (modal.maxreal((-phi_z_temp[::2, :] + phi_z_temp[1::2, :]) / B))

    # Horizontal component
    factor = 1 / np.max(np.abs(phi_y[:, mode_i]))
    factor_ref = 1 / np.max(np.abs(phi_y_ref[:, mode_i]))

    #if np.sum(np.abs(phi_y[:, mode_i] - phi_y_ref[:, mode_i]*factor_ref)) > 8.0:
    #    phi_y[:, mode_i] = phi_y[:, mode_i]*(-1)

    axs[0].plot(x, np.concatenate((np.array([0]), phi_y[:, mode_i], np.array([0]))
                                       , color='tab:red', alpha = 0.05)
    axs[0].set_title(
        'Mode ' + str(mode_i + 1) + ' - ' + 'Horizontal component')

    # Vertical component
    factor = 1 / np.max(np.abs(phi_z[:, mode_i]))
    factor_ref = 1 / np.max(np.abs(phi_z_ref[:, mode_i]))
```

```
        axs[1].plot(x, np.concatenate((np.array([0]), phi_z[:, mode_i], np.array([0])))
                                        , color='tab:blue', alpha = 0.05)
        axs[1].set_title(
            'Mode ' + str(mode_i + 1) + ' - ' + 'Vertical component')


    fig.tight_layout()

    return fig
```

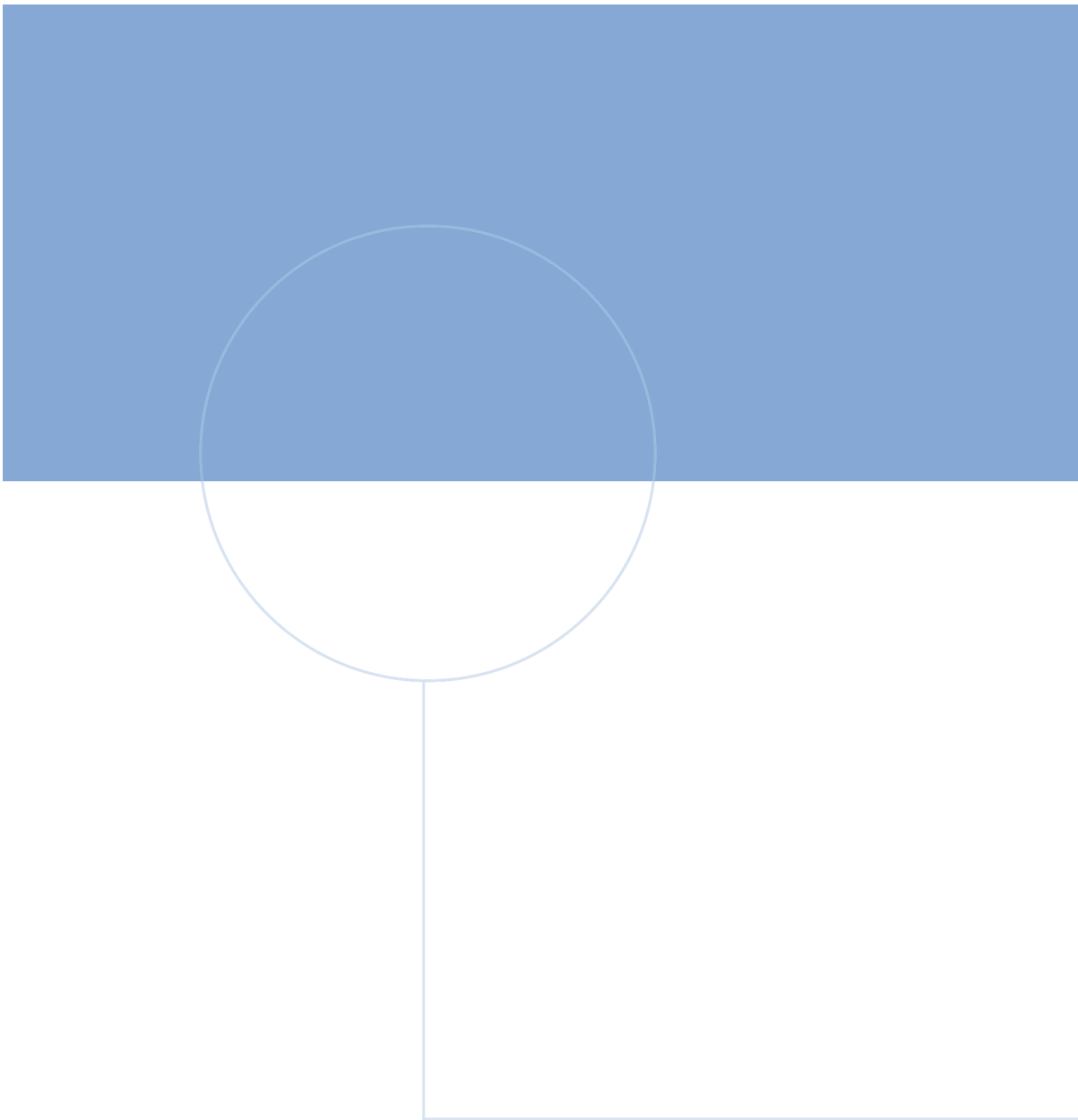
plot.py

```
import numpy as np
from scipy import signal
import matplotlib.pyplot as plt
import koma.modal as modal
from dataloader import FEM_result_loader
import colordict

color_values = list(colordict.ColorDict(norm=1).values())


def welch_plot(acc, sampling_frequency, Ndivisions):
    """
    Compute and plot the Welch spectrum of the acceleration data.

    Args:
        acc (array-like): The acceleration data.
        sampling_frequency (float): The sampling frequency of the acceleration data.
        Ndivisions (int): The number of divisions/segments for computing the Welch spectrum.

    Returns:
        None
    """

    Nwindow = np.ceil(len(acc) / Ndivisions)  # Length of window/segment

    Nfft_pow2 = 2 ** (np.ceil(np.log2(Nwindow)))  # Next power of 2 for zero padding
    dt = 1 / sampling_frequency  # Time step

    # Call welch from scipy signal processing
    f, Sx_welch = signal.welch(acc, fs=1 / dt, window='hann', nperseg=Nwindow, noverlap=None, nfft=Nfft_pow2,
                               detrend='constant', return_onesided=True, scaling='density', axis=- 1,
                               ↪  average='mean')

    plt.figure(figsize=(14, 7), dpi=250)
    plt.plot(f, Sx_welch, label='Welch spectrum of acceleration data')
    plt.xlabel('$f$ [Hz]')
    plt.ylabel('$S(f)$')
    plt.grid()
    plt.legend()
    plt.show()


def stabilization_diagram(acceleration, sampling_frequency, Ndivisions, frequencies, orders,
                          all_freqs=None, all_orders=None):
    """
    Plot the stabilization diagram showing model orders and frequencies, along with power spectral density (PSD)
    of the acceleration data.

    Args:
        acceleration (array-like): The acceleration data.
        sampling_frequency (float): The sampling frequency of the acceleration data.
        Ndivisions (int): The number of divisions/segments for computing the PSD.
        frequencies (array-like): An array of frequencies for each cluster.
        orders (array-like): An array of model orders for each cluster.
        all_freqs (array-like, optional): An array of frequencies for discarded modes (default: None).
        all_orders (array-like, optional): An array of model orders for discarded modes (default: None).

    Returns:
        matplotlib.figure.Figure: The generated figure.
    """

    fig, ax = plt.subplots(figsize=(14, 5), dpi=300)

    if len(all_freqs) > 0 and len(all_orders) > 0:
        ax.scatter(all_freqs, all_orders, marker='o', color='grey', label='Discarded modes')

    for cluster in range(frequencies.shape[0]):
```

```
        ax.plot(frequencies[cluster], orders[cluster], marker='o', color=color_values[cluster + cluster * 3])

    ax.set_ylabel("Model order")
    ax.set_xlabel('$f$ [Hz]')
    ax.set_xticks(np.arange(0, 1.1, step=0.1))
    ax.set_xlim([0, 1])

    # Creating power spectral density in each direction of motion of the bridge
    Nwindow = np.ceil(len(acceleration) / Ndivisions)  # Length of window/segment

    Nfft_pow2 = 2 ** (np.ceil(np.log2(Nwindow)))  # Next power of 2 for zero padding
    dt = 1 / sampling_frequency  # Time step

    length = acceleration.shape[1]
    length = int(length / 3)
    n_channels = int(length / 2)

    B = 18.6
    # Transforming to find y, z, theta component for the sensor pair at the midspan
    y = (acceleration[:, length + int(np.floor(length / 4))] +
         acceleration[:, length + int(np.floor(length / 4)) + n_channels]) / 2
    z = (acceleration[:, length + int(np.floor(length / 4)) + n_channels * 2] +
         acceleration[:, length + int(np.floor(length / 4)) + n_channels * 3]) / 2
    theta = (-acceleration[:, length + int(np.floor(length / 4)) + n_channels * 2] +
             acceleration[:, length + int(np.floor(length / 4)) + n_channels * 3]) / B

    # Call welch from scipy signal processing
    f, Sy_welch = signal.welch(y, fs=1 / dt, window='hann', nperseg=Nwindow, noverlap=None, nfft=Nfft_pow2,
                               detrend='constant', return_onesided=True, scaling='density', axis=- 1,
                               ↪  average='mean')
    f, Sz_welch = signal.welch(z, fs=1 / dt, window='hann', nperseg=Nwindow, noverlap=None, nfft=Nfft_pow2,
                               detrend='constant', return_onesided=True, scaling='density', axis=- 1,
                               ↪  average='mean')
    f, Stheta_welch = signal.welch(theta, fs=1 / dt, window='hann', nperseg=Nwindow, noverlap=None,
    ↪  nfft=Nfft_pow2,
                                   detrend='constant', return_onesided=True, scaling='density', axis=- 1,
                                   average='mean')

    ax2 = ax.twinx()
    ax2.plot(f, Sy_welch, color='black', label='$PSD\ y-direction$', lw=0.5)
    ax2.plot(f, Sz_welch, color='blue', label='$PSD\ z-direction$', lw=0.5)
    ax2.plot(f, Stheta_welch * 10, color='red', label=r'$PSD\ \theta-direction\ e1$', lw=0.5)
    ax2.set_ylabel("PSD")
    plt.legend()
    plt.grid()

    return fig


def stabilization_diagram_cov_ssi(acceleration, sampling_frequency, Ndivisions,
                                  all_freqs, all_orders, freq_stab=None, orders_stab=None):
    """
    Plot the stabilization diagram for covariance-driven subspace identification (SSI) method,
    showing model orders and frequencies, along with power spectral density (PSD)
    of the acceleration data.

    Args:
        acceleration (array-like): The acceleration data.
        sampling_frequency (float): The sampling frequency of the acceleration data.
        Ndivisions (int): The number of divisions/segments for computing the PSD.
        all_freqs (array-like): An array of frequencies for all the identified modes.
        all_orders (array-like): An array of model orders for all the identified modes.
        freq_stab (array-like, optional): An array of frequencies for stable modes (default: None).
        orders_stab (array-like, optional): An array of model orders for stable modes (default: None).

    Returns:
        matplotlib.figure.Figure: The generated figure.
    """
```

```
    fig, ax = plt.subplots(figsize=(14, 5), dpi=300)
    ax.scatter(all_freqs, all_orders, marker='o', color='grey')

    if not (freq_stab is None) and not (orders_stab is None):
        ax.scatter(freq_stab, orders_stab, marker='o', color='blue')

    ax.set_ylabel("Model order")
    ax.set_xlabel('$f$ [Hz]')
    ax.set_xticks(np.arange(0, 1, step=0.1))
    ax.set_xlim([0, 1])

    # Creating power spectral density in each direction of motion of the bridge
    Nwindow = np.ceil(len(acceleration) / Ndivisions)  # Length of window/segment

    Nfft_pow2 = 2 ** (np.ceil(np.log2(Nwindow)))  # Next power of 2 for zero padding
    dt = 1 / sampling_frequency  # Time step

    length = acceleration.shape[1]
    length = int(length / 3)
    n_channels = int(length / 2)

    B = 18.6
    # Transforming to find y, z, theta component for the sensor pair at the midspan
    y = (acceleration[:, length + int(np.floor(length / 4))] + acceleration[:, length + int(np.floor(length / 4))
    ↪  +
                                                                            n_channels]) / 2
    z = (acceleration[:, length + int(np.floor(length / 4)) + n_channels * 2] +
         acceleration[:, length + int(np.floor(length / 4)) + n_channels * 3]) / 2
    theta = (-acceleration[:, length + int(np.floor(length / 4)) + n_channels * 2] +
             acceleration[:, length + int(np.floor(length / 4)) + n_channels * 3]) / B

    # Call welch from scipy signal processing
    f, Sy_welch = signal.welch(y, fs=1 / dt, window='hann', nperseg=Nwindow, noverlap=None, nfft=Nfft_pow2,
                               detrend='constant', return_onesided=True, scaling='density', axis=- 1,
                               ↪  average='mean')
    f, Sz_welch = signal.welch(z, fs=1 / dt, window='hann', nperseg=Nwindow, noverlap=None, nfft=Nfft_pow2,
                               detrend='constant', return_onesided=True, scaling='density', axis=- 1,
                               ↪  average='mean')
    f, Stheta_welch = signal.welch(theta, fs=1 / dt, window='hann', nperseg=Nwindow, noverlap=None,
    ↪  nfft=Nfft_pow2,
                                   detrend='constant', return_onesided=True, scaling='density', axis=- 1,
                                   average='mean')

    ax2 = ax.twinx()
    ax2.plot(f, Sy_welch, color='black', label='$PSD\ y-direction$', lw=0.5)
    ax2.plot(f, Sz_welch, color='blue', label='$PSD\ z-direction$', lw=0.5)
    ax2.plot(f, Stheta_welch * 10, color='red', label=r'$PSD\ \theta-direction\ e1$', lw=0.5)
    ax2.set_ylabel("PSD")
    plt.legend()
    plt.grid()

    return fig


def plotModeShapeFEM(FEM_loader: FEM_result_loader, type='Vertical'):
    """
    Plots the mode shapes of the Finite Element Model (FEM) loaded using FEM_result_loader.

    Args:
        FEM_loader (FEM_result_loader): An instance of the FEM_result_loader class.
        type (str, optional): The type of mode shape to plot. Defaults to 'Vertical'.

    Returns:
        matplotlib.figure.Figure: The generated figure containing the mode shape plots.
    """

    f = FEM_loader.f
    phi_y = FEM_loader.phi_y
    phi_z = FEM_loader.phi_z
```

```
phi_t = FEM_loader.phi_t
x = FEM_loader.x_plot

num = FEM_loader.mode_type.count(type)

# Plot
fig, axs = plt.subplots(int(np.ceil(num / 2)), 2, figsize=(20, int(np.ceil(num / 2)) * 3), dpi=300)

j = 0
for i in range(len(f)):
    axs[int(np.floor(j / 2)), j % 2].set_xlabel('x[m]')
    axs[int(np.floor(j / 2)), j % 2].set_ylim([-1, 1])
    axs[int(np.floor(j / 2)), j % 2].set_xlim([-600, 600])
    axs[int(np.floor(j / 2)), j % 2].set_xticks([-600, -300, 0, 300, 600])
    axs[int(np.floor(j / 2)), j % 2].set_yticks([-1, -0.5, 0, 0.5, 1])

    if FEM_loader.mode_type[i] == 'Horizontal' and type == 'Horizontal':
        factor = 1 / np.max(np.abs(phi_y[:, i]))
        axs[int(np.floor(j / 2)), j % 2].plot(x, phi_y[:, i] * factor, color='black')
        axs[int(np.floor(j / 2)), j % 2].set_title(
            'Mode ' + str(i + 1) + ' - ' + type + '\n $f_n$ = ' + f"{f[i]:.3f}" + ' Hz')
        axs[int(np.floor(j / 2)), j % 2].grid()
        j += 1
    elif FEM_loader.mode_type[i] == 'Vertical' and type == 'Vertical':
        factor = 1 / np.max(np.abs(phi_z[:, i]))
        axs[int(np.floor(j / 2)), j % 2].plot(x, phi_z[:, i] * factor, color='black')
        axs[int(np.floor(j / 2)), j % 2].set_title(
            'Mode ' + str(i + 1) + ' - ' + type + '\n $f_n$ = ' + f"{f[i]:.3f}" + ' Hz')
        axs[int(np.floor(j / 2)), j % 2].grid()
        j += 1
    elif FEM_loader.mode_type[i] == 'Torsional' and type == 'Torsional':
        factor = 1 / np.max(np.abs(phi_t[:, i]))
        axs[int(np.floor(j / 2)), j % 2].plot(x, phi_t[:, i] * factor, color='black')
        axs[int(np.floor(j / 2)), j % 2].set_title(
            'Mode ' + str(i + 1) + ' - ' + type + '\n $f_n$ = ' + f"{f[i]:.3f}" + ' Hz')
        axs[int(np.floor(j / 2)), j % 2].grid()
        j += 1
    elif FEM_loader.mode_type[i] == 'Cable' and type == 'Cable':
        factor = 1 / np.max(np.abs(phi_y[:, i]))
        axs[int(np.floor(j / 2)), j % 2].plot(x, phi_y[:, i] * factor, color='black')
        axs[int(np.floor(j / 2)), j % 2].set_title(
            'Mode ' + str(i + 1) + ' - ' + type + '\n $f_n$ = ' + f"{f[i]:.3f}" + ' Hz')
        axs[int(np.floor(j / 2)), j % 2].grid()
        j += 1

if num % 2:
    fig.delaxes(axs[int(np.ceil(num / 2)) - 1, 1])

fig.tight_layout()

return fig
```



NTNU

Norwegian University of
Science and Technology