Master's thesis

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

Sebastian Knedahl Hansen

Time and frequency domain calculations of the wind induced dynamic response of the Hålogaland Bridge

Master's thesis in Civil and Environmental Engineering
Supervisor: Ole André Øiseth
Co-supervisor: Aksel Fenerci
June 2023

NTNU
Norwegian University of
Science and Technology

Sebastian Knedahl Hansen

Time and frequency domain calculations of the wind induced dynamic response of the Hålogaland Bridge

Master's thesis in Civil and Environmental Engineering
Supervisor: Ole André Øiseth
Co-supervisor: Aksel Fenerci
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 19, 2023	61 + 52

TITLE:

Time and frequency calculations of the wind induced dynamic response of the Hålogaland Bridge

Tids- og frekvensplansberegninger av vindindusert dynamisk respons av Hålogalandsbrua

BY:

Sebastian Knedahl Hansen

SUMMARY:
In bridge engineering it is important to understand how wind affects bridges in order to obtain safe bridge designs. This is especially important for long-span bridges which are more prone to wind induced dynamic response. In this thesis the wind induced dynamic response of the Hålogaland Bridge have been studied and calculated both in time and frequency domain. By assuming an aeroelastic system the calculations can be performed in both domains which is useful for comparing and verification of the results.
The time domain response was performed with 20 time series simulations of the wind field generated by the Monte Carlo method. The time and frequency response were successfully obtained and coincided with a fairly high accuracy.

In addition to the wind induced dynamic response, the flutter stability have been calculated and an extreme value analysis of the response have been performed.
The critical mean wind velocity was obtained from the eigenvalue analysis of the state space model of the equation of motion and found to be 80.69 m/s.

RESPONSIBLE TEACHER: Ole André Øiseth

SUPERVISOR(S): Ole André Øiseth, Aksel Fenerci
CARRIED OUT AT: Departmen of Structural Engineering

Abstract

In bridge engineering it is important to understand how wind affects bridges in order to obtain safe bridge designs. This is especially important for long-span bridges which are more prone to wind induced dynamic response. In this thesis the wind induced dynamic response of the Hålogaland Bridge have been studied and calculated both in time and frequency domain. By assuming an aeroelastic system the calculations can be performed in both domains which is useful for comparing and verification of the results. The time domain response was performed with 20 time series simulations of the wind field generated by the Monte Carlo method. The time and frequency response were successfully obtained and coincided with a fairly high accuracy.

In addition to the wind induced dynamic response, the flutter stability have been calculated and an extreme value analysis of the response have been performed. The critical mean wind velocity was obtained from the eigenvalue analysis of the state space model of the equation of motion and found to be 80.69 m/s.

Sammendrag

I bruteknikk er det viktig å forstå hvordan vind påvirker bruer for å kunne lage trygge bruer. Det er spesielt viktig for lange bruer ettersom de er mer utsatt for vindindusert dynamisk respons. I denne oppgaven har vindindusert dynamisk respons av Hålogalandsbrua blitt studert og beregnet i tids- og frekevensplanet. Ved å anta et aeroelastisk system kan beregningene bli utført både i tids- og frekvensplanet som gjør at løsningene kan sammenlignes og verifiseres. Tidsplansløsningen ble beregnet med 20 tidsseriesimuleringer av vindfeltet ved bruk av Monte Carlo metoden og ga tilfredstillende samsvar med frekvensplansløsningen.

I tillegg til vindindusert dynamisk repsonse ble flutter stabiliteten bergenet og ekstremverdianalyse av den dynamiske responsen ble gjennomført. Den kritiske middelvinden ble funnet gjennom eigenverdianalyse av tilstandsmodellen av bevegelsesligningen og ble funnet til å være 80.69 m/s.

Preface

This master thesis in structural dynamics, written at the Department of Structural Engineering, concludes the five year MSc programme in Civil and Environmental Engineering at the Norwegian University of Science and Technology (NTNU). The thesis also mark the beginning of my next four years as a PhD Candidate at the Department of Structural Engineering.

I would like to thank my supervisors Professor Ole Øiseth and Associate Professor Aksel Fenerci for sharing their knowledge and helping me understand and implement the theory presented in this thesis. I look forward to learn more from them in the coming years.

I would also like to thank Associate Professor Øyvind Wiig Petersen for preparing the Abaqus model of the Hålogaland Bridge and a Python script which extract the appropriate information from the model.

Sebastian Knedahl Hansen
June 19, 2023

Table of Contents

1 Introduction

As bridges become longer they are more susceptible to wind induced dynamic response and it is important to understand how the wind affects the bridges and how the wind induced response can be calculated. The goal of this thesis is to present theory that are not presented in any existing courses at the Department of Structural Engineering at NTNU, and implement the theory in order to calculate the dynamic response of the Hålogaland Bridge both in time and frequency domain. In addition to the wind induced dynamic response, extreme value analysis of the response and stability limit of the system is investigated. The focus is on this new theory and it is assumed that the reader has a good understanding of the topics taught in the courses TKT4201 Structural Dynamics 1 and TKT4108 Structural Dynamics 2.

The contents of this thesis is presented in a systematic manner. First is the theoretical background, describing how the bridge is affected by the wind, the dynamic response in frequency and time domain, flutter stability and extreme value analysis. In the next chapter the methodology is presented, describing how the theory was implemented in order to obtain the buffeting response, stability limit and estimates of the extreme response, as well as a short introduction of the Hålogaland Bridge and its dynamic properties. The next chapter presents all the results obtained followed by a discussion of the results and a conclusion of the thesis.

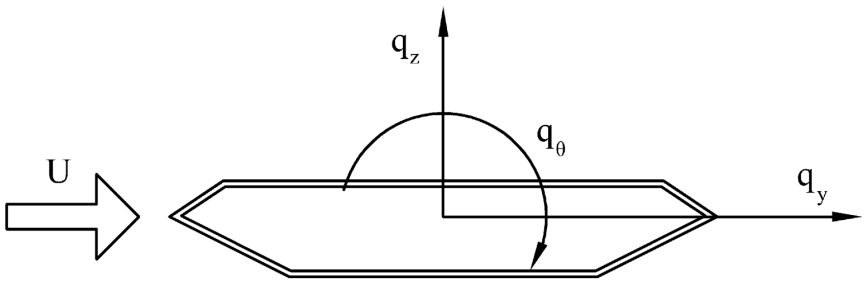


Figure 1: Aerodynamic forces acting on the cross section of the Hålogaland Bridge [1]

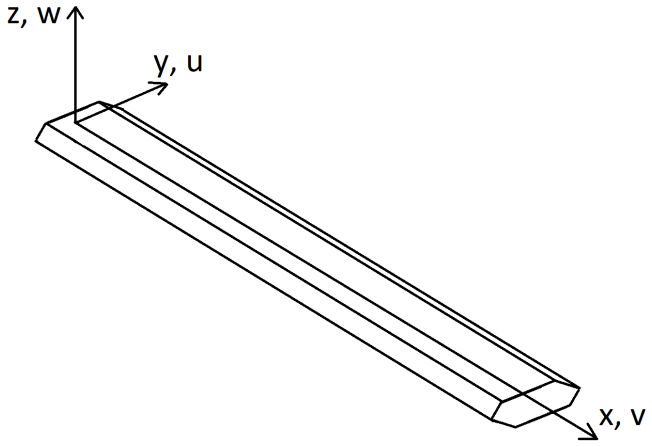


Figure 2: Coordinate system

2 Theoretical background

This chapter presents the theoretical background of wind induced dynamic response calculations in time and frequency domain, in addition to flutter stability and extreme value analysis of the dynamic response. The theory presented follows the so-called strip theory, which commonly used for long slender line-like structures. It is appropriate to apply the strip theory to the Hålogaland bridge as it is a long suspension bridge with a relatively constant cross section. In strip theory it is assumed that the wind loading acts as a concentrated force on a given section along the longitudinal direction. The shear centre of the cross section is commonly used as the reference point for the load. In order to obtain the forces acting on the cross section it is necessary to understand how the wind load is modelled.

2.1 Modelling of the wind load

Wind is the most contributing factor to the dynamic response of the Hålogaland Bridge, other factors such as contributions from traffic and seismic loading are neglected for the purpose of this thesis. Since wind is the lead causing effect of the dynamic response it is important to look at how the wind load is defined and calculated in order to calculate the dynamic response of the bridge.

Wind loading is most commonly evaluated in wind engineering today by the use of the Alan G. Davenport Wind Load Chain [2]. The wind load chain describes how the wind loading

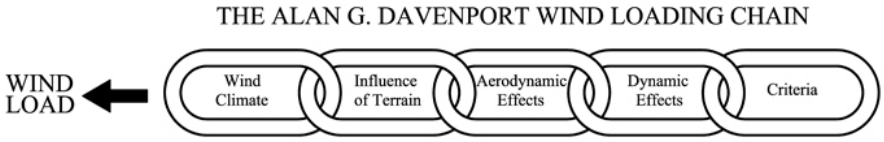


Figure 3: Davenport's wind load chain [4]

is a combined effect of the local wind climate, exposure, the aerodynamic properties of the structure and potential wind-induced resonant vibrations. The local wind climate must be described with statistical properties and the terrain roughness and topography surrounding the structure influences the local wind exposure. Davenport also recognised that there must be a clear criterion for judging the consequences of the wind action e.g., structural integrity and serviceability considerations for the users of the structure [3]. The wind load chain is illustrated in Figure 3.

Wind is a chaotic process, attacking a structure from all direction. The wind velocity vector U is defined as [5]

$$\mathrm{U}(x,\, y,\, z,\, t) = V(x,\, y,\, z) + u(x,\, y,\, z,\, t) + v(x,\, y,\, z,\, t) + w(x,\, y,\, z,\, t), \tag{2.1}$$

where V is the mean wind velocity over a given time interval T (usually equal to 10 minutes in bridge design) u, v and w is the turbulence components in the y-, x- and z-direction, respectively. The global coordinate system of the structure is shown in Figure 2. As seen from Eq. (2.1) the wind vector is defined with respect to x, y, z and t, meaning that it is defined in all of space and in time, this applies for the turbulence components as well, but not for the mean wind velocity which is only dependent on the spatial coordinates.

Two assumptions that are often made in bridge design are that the wind field is homogeneous and stationary. Assuming a homogeneous wind field implies that the probability distribution describing the wind field does not vary along any of the spatial coordinates. This is a fair assumption as weather conditions surrounding most civil engineering structures are considered homogeneous enough [5]. There may, however, be problems with the homogeneity if the wind field is influenced sufficiently by the surrounding terrain. The wind velocity generally increases with the height above ground. The wind is then stronger at the top of the towers than at the bridge deck which does not conform with homogeneity. However, in this thesis, all calculations are done purely on the bridge deck, towers, hangers and cables are neglected and the assumption of a homogeneous wind field is therefore appropriate. The assumption of a stationary wind field means that the statistical properties is invariant with respect to time, i.e. the probability distribution describing the wind field is equal for any time interval. The stationary assumption will generally not hold true for sufficiently long time series. The wind field is henceforth assumed to be stationary and homogeneous.

Because of the chaotic nature of the wind, it is assumed that the wind vector is perpendicular to the longitudinal direction of the bridge with zero angle of attack as shown in Figure 1. The velocity vector can then be simplified to [5]

$$\mathrm{U}(x,\, t) = V + u(x,\, t) + w(x,\, t). \tag{2.2}$$

The wind vector gives rise to the wind load components q_y, q_z and q_θ, also shown in Figure 1. These load components give rise to the corresponding displacements and rotations and are defined as positive in the same direction as the force components. The load components can be split into a mean and a fluctuating part [5]

$$\overline{\mathbf{q}}(x) + \mathbf{q}(x,\, t) = \begin{pmatrix} \overline{q}_y(x) \\ \overline{q}_z(x) \\ \overline{q}_\theta(x) \end{pmatrix} + \begin{pmatrix} q_y(x,\, t) \\ q_z(x,\, t) \\ q_\theta(x,\, t) \end{pmatrix}, \tag{2.3}$$

$$\overline{\mathbf{u}}(x) + \mathbf{u}(x,\, t) = \begin{pmatrix} \overline{u}_y(x) \\ \overline{u}_z(x) \\ \overline{u}_\theta(x) \end{pmatrix} + \begin{pmatrix} u_y(x,\, t) \\ u_z(x,\, t) \\ u_\theta(x,\, t) \end{pmatrix}, \tag{2.4}$$

where $\overline{\mathbf{q}}(x)$ and $\overline{\mathbf{u}}(x)$ are the mean load and response vector, respectively. These terms are invariant to time, i.e. the structure is in a state of static loading and subsequently exhibits static response. $\mathbf{q}(x,\, t)$ and $\mathbf{u}(x,\, t)$ is describing the fluctuating load and response. The static part is not of any interest in the buffeting response and flutter stability calculations and is therefore neglected henceforth.

2.1.1 Buffeting load

When the air flow is exhibiting turbulence, the structure is subjected to a fluctuating pressure load. This load is commonly known as buffeting load in aerodynamics engineering and is written as

$$\mathbf{q}_{buff} = \mathbf{B}_q \mathbf{v} \tag{2.5}$$

where $\mathbf{v}$ is the turbulence vector and $\mathbf{B}_q$ is the buffeting load matrix defined as [5]

$$\mathbf{B}_q = \frac{\rho V B}{2} \begin{pmatrix} 2(D/B)\overline{C}_D & ((D/B)C'_D - \overline{C}_L) \\ 2\overline{C}_L & (C'_L + (D/B)\overline{C}_D) \\ 2B\overline{C}_M & BC'_M \end{pmatrix}, \tag{2.6}$$

where $\overline{C}_n$ $n \in \{D,\, L,\, M\}$ are the force coefficients at the mean angle of attack, while C'_n $n \in \{D,\, L,\, M\}$ are the derivative of the force coefficients at the mean angle of attack. D, L and M signifies the drag, lifting and pitching moment, respectively. However, as the buffeting load matrix is derived from the assumption of quasi-steady wind loads, it is common to add admittance functions [5] to each element in the buffeting load matrix such that Eq. (2.6) can be rewritten as

$$\mathbf{B}_q(\omega) = \frac{\rho V B}{2} \begin{pmatrix} 2(D/B)\overline{C}_D A_{yu}(\omega) & ((D/B)C'_D - \overline{C}_L)A_{yw}(\omega) \\ 2\overline{C}_L A_{zu}(\omega) & (C'_L + (D/B)\overline{C}_D)A_{zw}(\omega) \\ 2B\overline{C}_M A_{\theta u}(\omega) & BC'_M A_{\theta w}(\omega) \end{pmatrix}, \tag{2.7}$$

where $A_{mn}(\omega)$ $m \in \{y,\, z,\, \theta\}$, $n \in \{u,\, w\}$ are the cross sectional admittance functions, developed by Sears [6] and later made more practical, by e.g. Liepmann [7]. The admittance functions are filter functions that filters the load contribution for higher frequencies

[5]. For the sake of simplicity, all admittance functions are set equal to one and Eq. (2.6) will be the buffeting load matrix that is used in further calculations.

Another loading phenomenon that may occur is vortex shedding. Vortex shedding is an important part of bridge design. However, it occurs for reasonably low wind velocities while buffeting and motion induced forces occur for higher velocities. Therefore, it is common to calculate the response for the different effect separately [5]. Vortex shedding will not be discussed further.

2.1.2 Wind turbulence characteristics

In order to design the bridge it is necessary to obtain wind field characteristics which can help describe the loading on the structure. As wind is a fluctuating and irregular (random) process it is best to treat it as a stochastic process and look at its statistical properties. This is useful for enabling the use of auto-spectral and cross-spectral density functions (the auto- and cross-spectral density is also known as the power spectral density). Different spectra have been created over time by e.g. Davenport [8], Kaimal et al. [9], Solari [10] and von Karman [11] to name a few. They are all empirical formulations of the power spectra and have their strengths and weaknesses. Another formulation for the power spectra is given by the Norwegian Public Roads Administration (NPRA), namely the N400 Handbook [12] developed for engineering of public bridges in Norway. The power spectra chosen for this thesis is according to the N400 as it is the reigning design standard in Norway.

The auto-spectral density $S_i(n)$ of the turbulence components u, v and w is given by the following equation from N400

$$\frac{nS_i}{\sigma_i^2} = \frac{A_i \hat{n}_i}{(1 + 1.5A_i\hat{n}_i)^{5/3}} \quad \text{for } i = \{u,\, v,\, w\} \tag{2.8}$$

$$\hat{n}_i = \frac{n^x L_i(z)}{v_m(z)} \tag{2.9}$$

where n is the frequency in Hz, σ_i is the standard deviation of the turbulence component i, ${}^xL_i$ is the integral length scale, $v_m(z)$ is the mean wind velocity over a statistical averaging period of 10 minutes at a vertical distance z above the terrain. The coefficients A_i are given as: $A_u = 6.8$, $A_v = 9.4$ and $A_w = 9.4$.

The cross-spectral density of the turbulence component i_1 and i_2, $S_{i_1 i_2}$, is given on the normalised form

$$\frac{\mathrm{Re}\left[S_{i_1 i_2}(n,\, \Delta s_j)\right]}{\sqrt{S_{i_1} \cdot S_{i_2}}} = \exp\left(-C_{ij}\frac{n\Delta s_j}{v_m(z)}\right) \tag{2.10}$$

where Δs_j is the horizontal or vertical distance between the considered points along the respective axes. $i_1,\, i_2 \in \{u,\, v,\, w\}$, being the turbulence components and $j \in \{y,\, z\}$ denote the spatial direction. The decay coefficients C_{ij} have the corresponding values: $C_{uy} = C_{uz} = 10.0$, $C_{vy} = C_{vz} = C_{wy} = 6.5$ and $C_{wz} = 3.0$. The N400 also state that the turbulence components u and w is to be used for calculations regarding horizontal bridge elements, while for vertical elements the turbulence components u and y should be used.

The bridge deck is the element of interest in this thesis. Therefore, only the turbulence components u and w will be used in further calculations. The turbulence vector then reads:

$$\mathbf{v} = \begin{pmatrix} u \\ w \end{pmatrix}. \tag{2.11}$$

2.1.3 Self-excited forces

When the bridge is exposed to the wind loading, it is put into motion. The motion of the bridge gives rise to what is called motion induced forces, also known as self-excited forces and was first introduced in bridge engineering by Scanlan and Tomko [13]. The self-excited forces can be written as [1][14]

$$\mathbf{q}_{se}(x,\, t) = \mathbf{C}_{ae}(V,\, \omega)\dot{\mathbf{u}}(x,\, t) + \mathbf{K}_{ae}(V,\, \omega)\mathbf{u}(x,\, t), \tag{2.12}$$

$$\mathbf{C}_{ae} = \frac{\rho B^2}{2}\omega \begin{pmatrix} P_1^* & P_5^* & BP_2^* \\ H_5^* & H_1^* & BH_2^* \\ BA_5^* & BA_1^* & B^2A_2^* \end{pmatrix}, \tag{2.13}$$

$$\mathbf{K}_{ae} = \frac{\rho B^2}{2}\omega^2 \begin{pmatrix} P_4^* & P_6^* & BP_3^* \\ H_6^* & H_4^* & BH_3^* \\ BA_6^* & BA_4^* & B^2A_3^* \end{pmatrix}, \tag{2.14}$$

$$\mathbf{q}_{se} = \begin{pmatrix} q_{se,y} & q_{se,z} & q_{se,\theta} \end{pmatrix}^T, \tag{2.15}$$

$$\mathbf{u} = \begin{pmatrix} u_y & u_z & u_\theta \end{pmatrix}^T. \tag{2.16}$$

Here, $\mathbf{C}_{ae}$ and $\mathbf{K}_{ae}$ are known as the aerodynamic damping and stiffness matrix, ρ is the air density, B is the cross sectional width of the bridge deck, ω is the circular frequency and P_n^*, H_n^*, A_n^* $n \in \{1,\, 2,\, ...,\, 6\}$ are the dimensionless aerodynamic derivatives. $q_{se,y}$ and $q_{se,z}$ are the self-excited forces in the transverse and vertical direction while $q_{se,\theta}$ is the self-excited torsional load. $\mathbf{u}$ is the displacement along the axis of the corresponding self-excited force and is positive in the same direction as the corresponding force. By introducing the self-excited forces, the equation of motion reads:

$$\mathbf{M}\ddot{\mathbf{u}}(t) + (\mathbf{C} - \mathbf{C}_{ae})\,\dot{\mathbf{u}}(t) + (\mathbf{K} - \mathbf{K}_{ae})\,\mathbf{u}(t) = \mathbf{q}_{buff}(t), \tag{2.17}$$

where $\mathbf{M}$, $\mathbf{C}$ and $\mathbf{K}$ are the mass, damping and stiffness matrix and $\mathbf{q}_{buff}$ is the buffeting load. Note that the aerodynamic damping and stiffness matrices are frequency dependent. This implies that the equation of motion contains both time and frequency dependent terms, which creates some challenges in solving the equation of motion. The following sections will show how the equation of motion can be solved, both in time and frequency domain, in order to obtain the buffeting response.

2.1.4 Aerodynamic derivatives

The dimensionless aerodynamic derivatives are characteristic cross sectional properties where P_n^* is related to the horizontal loading while H_n^* is related to the vertical loading and A_n^* is related to the torsional loading, this can readily be seen by looking at each row of Eq. (2.12). The aerodynamic derivatives are found experimentally through wind tunnel testing. The section model used in the wind tunnel testing is (usually) quite small compared to the full-scale bridge and the results obtained from the wind tunnel test is not directly applicable for the full-scale bridge. Because of this it is necessary to work with dimensionless variables which are applicable regardless of section width or the mean wind velocity. The dimensionless aerodynamic derivatives are therefore a function of the reduced velocity, which is defined as the mean wind velocity divided by the product of the section width and the circular frequency: $\hat{V} = V/(B \cdot \omega)$.

2.2 Buffeting response in frequency domain

In order to carry out the buffeting response calculations in the frequency domain, it is necessary to transfer the equation of motion entirely to frequency domain. But first, by introducing the modal transformation, the response $\mathbf{u}(x,\, t)$ can be written as a product of the mode shapes $\mathbf{\Phi}(x)$ and the generalised coordinates $\boldsymbol{\eta}(t)$ such that

$$\mathbf{u}(x,\, t) = \mathbf{\Phi}(x)\boldsymbol{\eta}(t). \tag{2.18}$$

By utilising the modal transformation, the frequency domain representation of the equation of motion subjected to self-excited forces and buffeting loads, can be found by applying the Fourier transformation to the equation of motion. The frequency domain representation of the equation of motion then reads as follows [14]

$$\widetilde{\mathbf{M}}\mathbf{G}_{\ddot{\boldsymbol{\eta}}}(\omega) + \left(\widetilde{\mathbf{C}} - \widetilde{\mathbf{C}}_{ae}(V,\, \omega)\right)\mathbf{G}_{\dot{\boldsymbol{\eta}}}(\omega) + \left(\widetilde{\mathbf{K}} - \widetilde{\mathbf{K}}_{ae}(V,\, \omega)\right)\mathbf{G}_{\boldsymbol{\eta}}(\omega) = \mathbf{G}_{\widetilde{\mathbf{q}}_{buff}}(\omega), \tag{2.19}$$

$$\widetilde{\mathbf{M}} = \int_0^L \mathbf{\Phi}^T\mathbf{M}\mathbf{\Phi}dx, \tag{2.20}$$

$$\widetilde{\mathbf{C}} = \int_0^L \mathbf{\Phi}^T\mathbf{C}\mathbf{\Phi}dx, \tag{2.21}$$

$$\widetilde{\mathbf{K}} = \int_0^L \mathbf{\Phi}^T\mathbf{K}\mathbf{\Phi}dx, \tag{2.22}$$

$$\widetilde{\mathbf{C}}_{ae} = \int_0^L \mathbf{\Phi}^T\mathbf{C}_{ae}\mathbf{\Phi}dx, \tag{2.23}$$

$$\widetilde{\mathbf{K}}_{ae} = \int_0^L \mathbf{\Phi}^T\mathbf{K}_{ae}\mathbf{\Phi}dx, \tag{2.24}$$

$$\mathbf{G}_{\widetilde{\mathbf{q}}_{buff}} = \int_0^L \mathbf{\Phi}^T(x)\mathbf{B}_q(\omega)\mathbf{G}_v(x,\,\omega)dx, \tag{2.25}$$

where $\widetilde{\mathbf{M}}$, $\widetilde{\mathbf{C}}$ and $\widetilde{\mathbf{K}}$ are the modal mass, damping and stiffness matrices, respectively, while $\widetilde{\mathbf{C}}_{ae}(V,\,\omega)$ and $\widetilde{\mathbf{K}}_{ae}(V,\,\omega)$ are the modal aerodynamic damping and stiffness matrix, respectively. $\mathbf{G}_{\boldsymbol{\eta}}(\omega)$ and $\mathbf{G}_{\widetilde{\mathbf{q}}_{buff}}(\omega)$ are the Fourier transform of the generalised response and buffeting load, respectively and $\mathbf{G}_v(x,\,\omega)$ is the Fourier transform of the turbulence components u and w:

$$\mathbf{G}_v(x,\,\omega) = \begin{pmatrix} G_u & G_w \end{pmatrix}^T. \tag{2.26}$$

The cross-spectral density of the generalised buffeting load can be found by taking the Fourier transform of the cross-correlation function (see e.g. [15] for procedure) of the generalised buffeting load which reads as follows [14]

$$\mathbf{S}_{\widetilde{\mathbf{q}}_{buff}}(\omega) = \int_L \int_L \mathbf{\Phi}^T(x_1)\mathbf{B}_q(\omega)\mathbf{S}_V^+(\Delta x,\,\omega)\mathbf{B}_q^T\mathbf{\Phi}(x_2)dx_1dx_2, \tag{2.27}$$

$$\mathbf{S}_V^+(\Delta x,\,\omega) = \begin{pmatrix} S_{uu}^+(\Delta x,\,\omega) & S_{uw}^+(\Delta x,\,\omega) \\ S_{wu}^+(\Delta x,\,\omega) & S_{ww}^+(\Delta x,\,\omega) \end{pmatrix}, \tag{2.28}$$

where $S_{nm}^+(\Delta x,\,\omega)$ $n,\,m \in \{u,\,w\}$ are the one-sided cross-spectral densities of the turbulence components along the structure. The co-spectra of u and w is often neglected, e.g. [5][16][17][18] and the effects of the co-spectra have been studied by Øiseth et al. [19] and shown that the contribution from the co-spectra was relative small. However, it was noted that there are uncertainties in the modelling of the wind field and that the importance of the co-spectra may increase for long-span bridges where the natural frequency are lower. For the purpose of this thesis the cross-spectra of the turbulence components are assumed negligible, Eq. (2.28) then reads

$$\mathbf{S}_V^+(\Delta x,\,\omega) = \begin{pmatrix} S_{uu}^+(\Delta x,\,\omega) & 0 \\ 0 & S_{ww}^+(\Delta x,\,\omega) \end{pmatrix}. \tag{2.29}$$

The cross-spectral density of the response at a given point x_r along the structure can be calculated with the following equation [20]

$$\mathbf{S}_R(\omega,\,x_r) = \mathbf{\Phi}(x_r)\overline{\mathbf{H}}_\eta(\omega)\mathbf{S}_{\widetilde{\mathbf{q}}_{buff}}(\omega)\mathbf{H}_\eta^T(\omega)\mathbf{\Phi}(x_r) \tag{2.30}$$

here $\mathbf{H}_\eta(\omega)$ is the frequency response matrix of the equation of motion in generalised coordinates defined as

$$\mathbf{H}_\eta(\omega) = \left[-\omega^2\widetilde{\mathbf{M}} + i\omega\left(\widetilde{\mathbf{C}} - \widetilde{\mathbf{C}}_{ae}(V,\,\omega)\right) + \left(\widetilde{\mathbf{K}} - \widetilde{\mathbf{K}}_{ae}(V,\,\omega)\right)\right]^{-1} \tag{2.31}$$

where i denote the imaginary unit.

2.3 Buffeting response in time domain

Calculating the buffeting response in the time domain is commonly carried out when nonlinearities must be considered [21], as there are difficulties in applying them in the frequency domain analysis. Even though there are no such nonlinearities accounted for in this thesis and the fact that frequency domain analysis is less computationally expensive than the time domain analysis, it is still useful to perform the time domain analysis in order to compare the results obtained in both domains.

2.3.1 Self-excited forces in time domain

As seen from Eq. (2.12), the self-excited forces are functions of the spatial coordinate corresponding to the longitudinal direction of the bridge and time, while the aerodynamic damping and stiffness matrices are dependent on the mean wind velocity and frequency. This means that Eq. (2.12) is only valid for one single-frequency harmonic motion. This can, however, be overcome by introducing the principle of superposition. Eq. (2.12) can then be extended to any periodic or aperiodic motion by applying Fourier integral representation [1]

$$\mathbf{G_q} = \mathbf{F}(\omega)\mathbf{G_u}(\omega), \tag{2.32}$$

$$\mathbf{F}(\omega) = \frac{1}{2}\rho V^2 \begin{pmatrix} K^2(P_1^*i + P_4^*) & K^2(P_5^*i + P_6^*) & K^2B(P_2^*i + P_3^*) \\ K^2(H_5^*i + H_6^*) & K^2(H_1^*i + H_4^*) & K^2B(H_2^*i + H_3^*) \\ K^2B(A_5^*i + A_6^*) & K^2B(A_1^*i + A_4^*) & K^2B^2(A_2^*i + A_3^*) \end{pmatrix}, \tag{2.33}$$

where $\mathbf{G_u}(\omega)$ are the Fourier transformation of the self-excited forces and the response, respectively. $\mathbf{F}(\omega)$ is the frequency response matrix which contains the transfer functions defined in terms of the aerodynamic derivatives and is a function of the reduced circular frequency of motion $K = (B \cdot \omega)/V$. The transfer functions are treated as continuous functions [1].

The self-excited forces in the time domain can be found through Eq. (2.32) by the use of the convolution theorem. The convolution theorem states that a multiplication in frequency domain is equal to a convolution in time domain [22]. Subsequently, the time domain representation of Eq. (2.32) can be written as [23]

$$\mathbf{q}_{se}(x,\, t) = \int_{-\infty}^{\infty} \mathbf{f}(t-\tau)\mathbf{u}(x,\, \tau)d\tau \tag{2.34}$$

where $\mathbf{f}$ is the matrix containing the aerodynamic impulse-response functions and is obtain by the inverse Fourier transform of the aerodynamic transfer functions in matrix $\mathbf{F}(\omega)$.

2.3.2 Curve fitting of the aerodynamic derivatives

As mentioned earlier, the aerodynamic derivatives are found through wind tunnel testing and are only known at a limited number of reduced velocities. Curve fitting is therefore necessary in order to obtain continuous aerodynamic functions. Note that the curve

fitting function must be appropriate for the inverse Fourier transform in order to obtain $\mathbf{f}$. The following rational function have been used frequently in the literature (e.g. [1][14][23][24][25][26][27])

$$\begin{aligned}\mathbf{F}(\omega) &= \frac{1}{2}\rho V^2\left(\mathbf{a}_1 + \mathbf{a}_2\frac{i\omega B}{V} + \mathbf{a}_3\left(\frac{i\omega B}{V}\right)^2 + \sum_{l=1}^{N-3}\mathbf{a}_{l+3}\frac{i\omega B/V}{i\omega B/V + d_l}\right)\\ &= \frac{1}{2}\rho V^2\left(\mathbf{a}_1 + \mathbf{a}_2 iK + \mathbf{a}_3(iK)^2 + \sum_{l=1}^{N-3}\mathbf{a}_{l+3}\frac{iK}{iK + d_l}\right).\end{aligned} \tag{2.35}$$

Here $\mathbf{a}_i$ $i \in \{1,\ 2,\ ...,\ N\}$ and d_l $l \in \{1,\ 2,\ ...,\ N-3\}$ are frequency independent matrices and coefficients that need to be determined through curve fitting. $\mathbf{a}_1$ represent non-circulatory static aerodynamics, $\mathbf{a}_2$ represent the aerodynamic damping while $\mathbf{a}_3$ represent the additional aerodynamic mass, which is normally negligible [27], $\mathbf{a}_3$ is assumed negligible from here on out. The rational terms containing $\mathbf{a}_{l+3}$ and d_l represent the unsteady part of the self-excited forces.

By looking at the real and imaginary part of the transfer functions in Eq. (2.33) it can be seen that the real and imaginary part of the transfer functions correspond to the aerodynamic derivatives in the aerodynamic stiffness and damping matrix, respectively (see Eq. (2.12)). It is therefore convenient to write Eq. (2.35) in terms of the real and imaginary part of the rational functions on the following form

$$\frac{\mathrm{Re}(\mathbf{F}(\omega))}{\frac{1}{2}\rho V^2 K^2} = \hat{V}^2\left(\mathbf{a}_1 + \sum_{l=3}^{N-3}\mathbf{a}_{l+3}\frac{1}{\left[\left(d_l\hat{V}\right)^2 + 1\right]}\right), \tag{2.36}$$

$$\frac{\mathrm{Im}(\mathbf{F}(\omega))}{\frac{1}{2}\rho V^2 K^2} = \hat{V}\left(\mathbf{a}_2 + \hat{V}^2\sum_{l=3}^{N-3}\mathbf{a}_{l+3}\frac{d_l}{\left[\left(d_l\hat{V}\right)^2 + 1\right]}\right). \tag{2.37}$$

Looking at Eq. (2.35) it can be noted that a nonlinear curve fitting is needed. Finding proper coefficients for the rational functions and deciding how many terms to include can be a tricky process if the experimental data describing the aerodynamic derivatives is only known for a limited range of reduced velocities. This is because the entire frequency domain is needed when taking the inverse Fourier transform [28]. This implies that extrapolating out of the known range of reduced velocities may be problematic as it can yield unrealistic values. It can therefore be convenient to force the curves to quasi-steady asymptotes [24]. Algorithm 1 have been recommended in order to determine the unknown coefficients [28]

Having obtained the rational function through curve fitting, the inverse Fourier transform can be applied to Eq. (2.35) and inserted it into Eq. (2.34). This renders the following expression for the self-excited forces

Algorithm 1 Curve fitting

Step1: Make an initial guess for d_l and use linear regression to find $\mathbf{a}_i$
Step2: Use nonlinear regression to optimise d_l and use linear regression to find a new coefficients for $\mathbf{a}_i$
Step3: Perform nonlinear regression to determine all coefficients d_l and $\mathbf{a}_i$ using the results from the previous step as an initial guess

$$\mathbf{q}_{se}(x,\,t) = \frac{1}{2}\rho V^2 \left(\mathbf{a}_1 \mathbf{u}(x,\,t) + \frac{B}{V}\mathbf{a}_2 \dot{\mathbf{u}}(x,\,t) + \sum_{l=3}^{N-3} \mathbf{a}_{l+3} \int_{-\infty}^{\infty} \left(\mathbf{u}(x,\,t) - \frac{d_l V}{B} \int_{-\infty}^{t} e^{-\frac{d_l V}{B}(t-\tau)} \mathbf{u}(x,\,\tau) d\tau \right) \right) \tag{2.38}$$

By transforming the system into generalised coordinates, the generalised self-excited forces can be written as the following

$$\begin{aligned} \widetilde{\mathbf{q}}_{se}(t) &= \int_0^L \mathbf{\Phi}^T(x) \mathbf{q}_{se}(x,\,t) dx \\ &= \widetilde{\mathbf{a}}_1 \boldsymbol{\eta}(t) + \widetilde{\mathbf{a}}_2 \boldsymbol{\eta}(t) + \widetilde{\mathbf{Z}}(t) \end{aligned} \tag{2.39}$$

$$\widetilde{\mathbf{a}}_1 = \frac{1}{2}\rho V^2 \int_0^L \mathbf{\Phi}^T \mathbf{a}_1 \mathbf{\Phi} dx, \tag{2.40}$$

$$\widetilde{\mathbf{a}}_2 = \frac{1}{2}\rho V^2 \int_0^L \mathbf{\Phi}^T \mathbf{a}_2 \mathbf{\Phi} dx \tag{2.41}$$

$$\widetilde{\mathbf{Z}}(t) = \frac{1}{2}\rho V^2 \int_0^L \sum_{l=3}^{N-3} \mathbf{\Phi}^T \mathbf{a}_{l+3} \mathbf{\Phi} \cdot \int_{-\infty}^{\infty} \left(\boldsymbol{\eta}(t) - \frac{d_l V}{B} \int_{-\infty}^{t} e^{-\frac{d_l V}{B}(t-\tau)} \boldsymbol{\eta}(\tau) d\tau \right) \tag{2.42}$$

$\widetilde{\boldsymbol{Z}}$ can be expressed in terms of a matrix multiplication such as

$$\widetilde{\boldsymbol{Z}} = \boldsymbol{Q}\boldsymbol{X} \tag{2.43}$$

$$\boldsymbol{Q} = \begin{pmatrix} \widetilde{\mathbf{a}}_4 & \widetilde{\mathbf{a}}_5 & \dots & \widetilde{\mathbf{a}}_N \end{pmatrix} \tag{2.44}$$

$$\boldsymbol{X} = \begin{pmatrix} \mathbf{x}_1^T & \mathbf{x}_2^T & \dots & \mathbf{x}_{N-3}^T \end{pmatrix}^T \tag{2.45}$$

$$\mathbf{x}_l = \left(\boldsymbol{\eta}(t) - \frac{d_l V}{B} \int_{-\infty}^{t} e^{-\frac{d_l V}{B}(t-\tau)} \boldsymbol{\eta}(\tau) d\tau \right). \tag{2.46}$$

The convolution integral in Eq. (2.46) is a computationally expensive calculation and it is advantageous to create a state-space model in order to avoid calculating the convolution integral, this has been shown by e.g [1][16].

2.3.3 State-space representation of the equation of motion

This section describes the steps needed to create a state space model which can be used to calculate the buffeting response. Starting with the derivative of Eq. (2.46) and utilising partial integration of the convolution integral, it can be shown that the derivative of Eq. (2.46) reads

$$\dot{\mathbf{x}}_l = \boldsymbol{\eta}(t) - \frac{d_l V}{B}\mathbf{x}_l. \tag{2.47}$$

Writing Eq. (2.47) on matrix form and moving everything over to the left hand side of the equation renders the following expression

$$\boldsymbol{B}\dot{\boldsymbol{X}} + \boldsymbol{D}\boldsymbol{X} - \boldsymbol{E}\dot{\boldsymbol{\eta}} = \mathbf{0}, \tag{2.48}$$

$$\boldsymbol{B} = \begin{pmatrix} \mathbf{I} & & & \\ & \mathbf{I} & & \\ & & \ddots & \\ & & & \mathbf{I} \end{pmatrix}, \quad \boldsymbol{E} = \begin{pmatrix} \mathbf{I} \\ \mathbf{I} \\ \vdots \\ \mathbf{I} \end{pmatrix}, \quad \boldsymbol{D} = \frac{V}{B}\begin{pmatrix} d_1\mathbf{I} & & & \\ & d_2\mathbf{I} & & \\ & & \ddots & \\ & & & d_{N-3}\mathbf{I} \end{pmatrix}, \tag{2.49}$$

where $\mathbf{I}$ is the identity matrix and each $\mathbf{I}$ has the same number of rows and columns as there are generalised coordinates. The equation of motion in generalised coordinates reads

$$\widetilde{\mathbf{M}}\ddot{\boldsymbol{\eta}}(t) + \widetilde{\mathbf{C}}\dot{\boldsymbol{\eta}}(t) + \widetilde{\mathbf{K}}\boldsymbol{\eta}(t) = \widetilde{\mathbf{Q}}_{tot}(t), \tag{2.50}$$

$$\widetilde{\mathbf{Q}}_{tot}(t) = \widetilde{\mathbf{q}}_{buff}(t) + \widetilde{\mathbf{q}}_{se}(t). \tag{2.51}$$

Inserting the generalised self-excited forces from Eq. (2.39) with the matrix notation in Eq. (2.43) into the equation of motion in Eq. (2.50) yields the following equation of motion

$$\widetilde{\mathbf{M}}\ddot{\boldsymbol{\eta}}(t) + \left(\widetilde{\mathbf{C}} - \widetilde{\mathbf{a}}_2\right)\dot{\boldsymbol{\eta}}(t) + \left(\widetilde{\mathbf{K}} - \widetilde{\mathbf{a}}_1\right)\boldsymbol{\eta}(t) - \boldsymbol{Q}\boldsymbol{X} = \widetilde{\mathbf{q}}_{buff}(t). \tag{2.52}$$

By pre-multiplying with the inverse of the generalised mass matrix $\widetilde{\mathbf{M}}^{-1}$ in Eq. (2.52), the equation, together with Eq. (2.48) can be expressed as the following state space model

$$\begin{pmatrix} \dot{\boldsymbol{\eta}} \\ \ddot{\boldsymbol{\eta}} \\ \dot{\boldsymbol{X}} \end{pmatrix} + \begin{pmatrix} \mathbf{0} & -\mathbf{I} & \mathbf{0} \\ \widetilde{\mathbf{M}}^{-1}\left(\widetilde{\mathbf{K}} - \widetilde{\mathbf{a}}_1\right) & \widetilde{\mathbf{M}}^{-1}\left(\widetilde{\mathbf{C}} - \widetilde{\mathbf{a}}_2\right) & -\widetilde{\mathbf{M}}^{-1}\boldsymbol{Q} \\ \mathbf{0} & -\boldsymbol{E} & \boldsymbol{D} \end{pmatrix} \begin{pmatrix} \boldsymbol{\eta} \\ \dot{\boldsymbol{\eta}} \\ \boldsymbol{X} \end{pmatrix} = \begin{pmatrix} \mathbf{0} \\ \widetilde{\mathbf{M}}^{-1}\widetilde{\mathbf{q}}_{buff}(t) \\ \mathbf{0} \end{pmatrix}. \tag{2.53}$$

Eq. (2.53) can be solved by discretisation of the state space model (see e.g. [29]). A general state space representation for a linear system can be written as [29]

$$\begin{aligned} \dot{\mathbf{x}}(t) &= \mathbf{A}_c\mathbf{x}(t) + \mathbf{B}_c\mathbf{u}(t) \\ \mathbf{y}(t) &= \mathbf{C}_c\mathbf{x}(t) + \mathbf{D}_c\mathbf{u}(t) \end{aligned} \tag{2.54}$$

where $\mathbf{x}$, $\mathbf{u}$ and $\mathbf{y}$ are known as the state, input and output vector, respectively. $\mathbf{A}_c$, $\mathbf{B}_c$, $\mathbf{C}_c$ and $\mathbf{D}_c$ are known as the state, input, output and feedthrough matrices. The sub index c denotes that this is a continuous system in time. However, we are usually working with discretised models and discretised time steps Δt, and therefore, in need to represent the state space model in discretised time. The continuous state space model can be transform to the following discretised model assuming zero-order hold [29]

$$\begin{aligned} \dot{\mathbf{x}}_{k+1} &= \mathbf{A}_d \mathbf{x}_k + \mathbf{B}_d \mathbf{u}_k \\ \mathbf{y}_k &= \mathbf{C}_d \mathbf{x}_k + \mathbf{D}_d \mathbf{u}_k \end{aligned} \tag{2.55}$$

$$\mathbf{A}_d = e^{\mathbf{A}_c \Delta t}, \quad \mathbf{B}_d = [\mathbf{A}_d - \mathbf{I}]\, \mathbf{A}_c^{-1} \mathbf{B}_c, \quad \mathbf{C}_d = \mathbf{C}_c, \quad \mathbf{D}_d = \mathbf{D}_c \tag{2.56}$$

where k denotes the current time increment t_k. Now, applying the state space representation of the equation of motion in Eq. (2.53) to the aforementioned discretised state space model, Eq. (2.55), the following yields true

$$\begin{aligned} \dot{\mathbf{x}}_{k+1} &= \mathbf{A}_d \mathbf{x}_k + \mathbf{B}_d \tilde{\mathbf{q}}_{buff,k} \\ \mathbf{y}_k &= \mathbf{C}_d \mathbf{x}_k \end{aligned} \tag{2.57}$$

$$\mathbf{A}_c = -\begin{pmatrix} \mathbf{0} & -\mathbf{I} & \mathbf{0} \\ \widetilde{\mathbf{M}}^{-1}\left(\widetilde{\mathbf{K}} - \widetilde{\mathbf{a}}_1\right) & \widetilde{\mathbf{M}}^{-1}\left(\widetilde{\mathbf{C}} - \widetilde{\mathbf{a}}_2\right) & -\widetilde{\mathbf{M}}^{-1} Q \\ \mathbf{0} & -\boldsymbol{E} & \boldsymbol{D} \end{pmatrix}, \quad \mathbf{B}_c = \begin{pmatrix} \mathbf{0} \\ \widetilde{\mathbf{M}}^{-1} \\ \mathbf{0} \end{pmatrix}, \quad \mathbf{C}_c = \mathbf{I} \tag{2.58}$$

where $\mathbf{y}_k = \begin{pmatrix} \boldsymbol{\eta}_k^T & \dot{\boldsymbol{\eta}}_k^T & \boldsymbol{X}_k^T \end{pmatrix}^T$ and the matrices $\mathbf{A}_d$, $\mathbf{B}_d$ and $\mathbf{C}_d$ can be established through Eq. (2.56). The buffeting response can be obtained through this state space model by calculating the generalised response $\boldsymbol{\eta}$ and pre-multiplying with the mode shapes $\boldsymbol{\Phi}$. However, the generalised buffeting load must be properly defined before the calculations can be carried out. The buffeting load is dependent on the buffeting load matrix and the turbulence components. The buffeting load matrix have been defined previously (Eq. (2.6)). The turbulence components can be obtained through simulated time series.

2.3.4 Time series simulations

By utilising the Monte Carlo method, as done by e.g. [1][24][30][31][32], the wind field characteristics can be used to create a simulated wind field which describes the turbulence along the bridge. The simulated time series for a turbulence component $x \in \{u,\, v,\, w\}$ at a point m along the bridge reads [5]

$$x_m(t) = \sum_{n=1}^{m} \sum_{j=1}^{N} \left| G_{mn}(\omega_j) \right| \cdot \sqrt{2\Delta\omega} \cdot \cos\left(\omega_j \cdot t + \psi_{nj}\right) \tag{2.59}$$

where $\psi_{nj} \in [0,\, 2\pi]$ is an arbitrary phase angle, ω_j is the frequency segment j and $\Delta\omega = \omega_{j+1} - \omega_j$ is the distance between two frequency segments. $G_{mn}(\omega_j)$ corresponds to

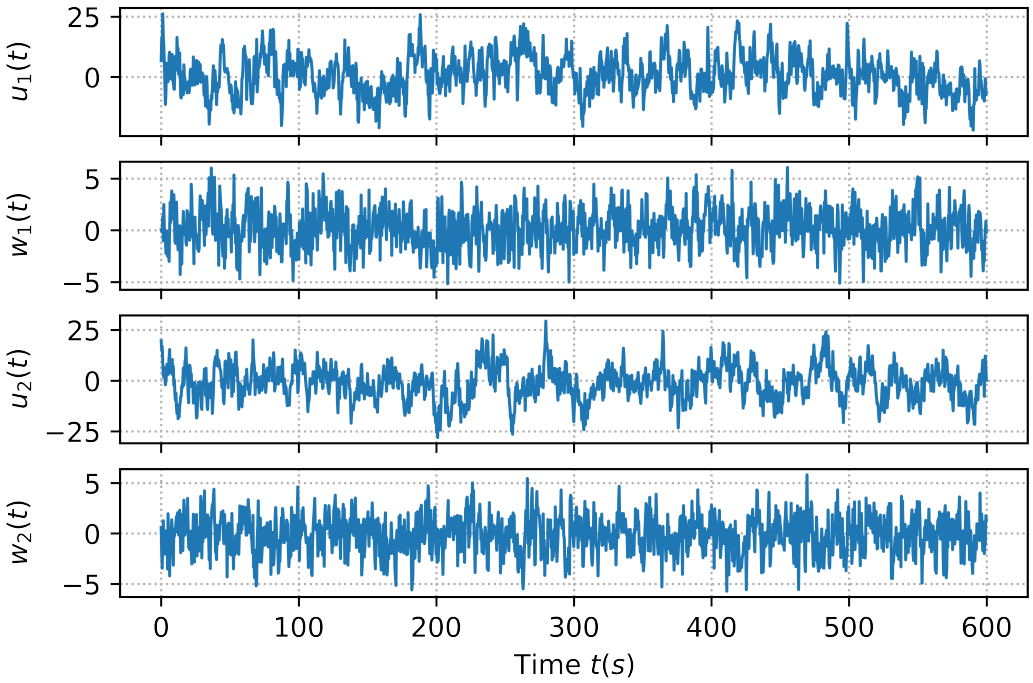


Figure 4: Simulated fluctuating wind velocities at the quarter-span (1) and midspan (2) over a 600 s window; u denoting the along-wind turbulence component while w denote the vertical turbulence component.

the elements of the matrix $\boldsymbol{G}(\omega)$ at frequency segment j. An example of simulated time series of the turbulence components u and w at two points along the bridge is shown in Figure 4. $\boldsymbol{G}(\omega)$ denote the Cholesky decomposition [33] of the cross-spectral density of the turbulence components, $\mathbf{S}(\omega)$, along the bridge. The Cholesky decomposition have the property that $\mathbf{S}(\omega) = \boldsymbol{G}(\omega)\boldsymbol{G}^H(\omega)$ where $\boldsymbol{G}$ is a lower triangular matrix and H denotes the Hermitian transpose. A nxn lower triangular matrix has $(n^2 - n)/2$ zero entries and therefore reduces the number of calculations needed by the same amount, which is why Cholesky decomposition is frequently used, see e.g. [1][24][32][34][35][36].

The inner sum of Eq. (2.59) can rewritten on exponential form by the use of Euler's formula:

$$x_m(t) = \mathrm{Re}\left(\sum_{n=1}^{m}\sum_{j=1}^{N}\left|G_{mn}(\omega_j)\right| \cdot \sqrt{2\Delta\omega} \cdot e^{\psi_{nj}} e^{i\omega_j \cdot t}\right) \tag{2.60}$$

Again by looking at the inner sum, the inner sum can be recognising as the inverse discrete Fourier transform of $|G|\sqrt{2\Delta\omega}e^{\psi}$. Subsequently, the inverse FFT (IFFT) can be applied, as shown by Shinozuka [31], and Eq. (2.60) can be rewritten to

$$x_m(t) = \mathrm{Re}\left(\sum_{n=1}^{m}\mathrm{IFFT}\left(|\boldsymbol{G}_{mn}| \cdot \sqrt{2\Delta\omega} \cdot e^{\Psi_n}\right)\right) \tag{2.61}$$

The turbulence vector $\mathbf{v}$ can be found by performing the time series simulations according to Eq. (2.61) for the turbulence components u and w such that

$$\mathbf{v} = \begin{pmatrix} u(x,\, t) \\ w(x,\, t) \end{pmatrix}. \tag{2.62}$$

Going from simulating time series with double summation in Eq. (2.59) to single summation with the IFFT reduces the computational costs drastically. However, the computational cost can be reduced even further. One of the most time consuming calculations in the simulation is the Cholesky decomposition. The problem have been circumvented, by e.g. [37], by calculating the Cholesky decomposition, $\mathbf{G}(\omega_{red})$, for a coarser frequency axis and then interpolate $\mathbf{G}(\omega_{red})$ onto the main frequency axis.

2.3.5 Buffeting load

Now everything is accounted for, except the buffeting load $\mathbf{q}_{buff}$. As seen by the frequency domain representation of the buffeting load in Eq. (2.25), the buffeting load is defined through a multiplication of the buffeting load matrix $\mathbf{B}_q(\omega)$ and the turbulence components $\mathbf{G}_v(x,\, \omega)$. As stated earlier (regarding the self-excited forces) a multiplication in the frequency domain corresponds to a convolution integral in time domain. The buffeting load in time domain then reads

$$\mathbf{q}_{buff}(x,\, t) = \int_{-\infty}^{\infty} \mathbf{b}_q(\tau)\mathbf{v}(x,\, t-\tau)d\tau \tag{2.63}$$

where $\mathbf{b_q}(\tau)$ is the inverse Fourier transform of $\mathbf{B_q}(\omega)$. By applying the modal transformation, the generalised buffeting load reads

$$\tilde{\mathbf{q}}_{buff}(t) = \int_0^L \mathbf{\Phi}^T(x)\mathbf{q}_{buff}(x,\, t)dx \tag{2.64}$$

The convolution integral in Eq. (2.63) is a computational expensive calculation and it is beneficial to avoid calculating the integral directly. The convolution integral can be evaluated in Fourier space. This can be done by taking the Fourier transform of the turbulence vector $\mathbf{v}$ obtained through the time series simulation, which yields $\mathbf{G}_v(\omega)$. The buffeting load matrix $\mathbf{B}_q$ and $\mathbf{G}_v$ can then be multiplied together, according to Eq. (2.25). The inverse Fourier transform of the result then yields the buffeting load $\mathbf{q}_{buff}(t)$.

2.4 Flutter stability

Flutter is a phenomenon in which the structure exhibits unsteady dynamic behaviour due self-excited vibrations. The flutter response is govern by vertical or torsional response, or both vertical and torsional response in a coupled configuration. The stability limit is an important aspect of bridge design, which tells us for which frequency and mean wind velocities the structure becomes unstable. The stability limit of an aeroelastic system can be determined by solving the following complex polynomial [14][38]

$$\left|\det\left(\mathbf{E}_\eta(V,\, \omega)\right)\right| = 0 \tag{2.65}$$

where **E** is the impedance matrix and is equal to the inverse of the frequency response matrix. Because of the relation between the impedance matrix and the frequency response matrix, the solution of Eq. (2.65) can readily be obtained graphically by plotting the absolute value of the determinant of the frequency response matrix and seeing where it tends to infinity.

Another way to determine the stability limit of an aeroelastic system is through eigenvalue analysis of the state space model, as done by e.g. [1][24][39][40]. This is done by calculating the eigenvalues of $\mathbf{A}_c$, defined in Eq. (2.58). The eigenvalues of $\mathbf{A}_c$ will generally be on the form $S_n = \mu_n + \omega_n i$. The eigenvalues S_n have the properties that the imaginary part correspond to the natural frequency of the system and the damping ratio can be found by the following expression [28]

$$\xi_n = -\frac{\mathrm{Re}\,(S_n)}{|S_n|}. \tag{2.66}$$

From these properties it is seen that a negative real part implies positive damping ratio and a stable system, while a positive real part implies negative damping and an unstable system. If the imaginary part of S_n is zero, $\omega_n = 0$, the response is non-periodic, while a nonzero imaginary part implies an oscillatory response.

2.5 Extreme value analysis time domain

Extreme value analysis is an important part of structural safety and design. The results from the buffeting response can be used to fit the extreme values of the response to a probability distribution function (PDF). Generally, the probability distribution is not known. However, since the extreme response is the matter of interest, it will follow an extreme value distribution. There are only three types of extreme value distributions [41], namely, Gumbel [42], Fréchet [43] and Weibull [44] distribution. All three distribution can be described on one parametric form called the generalised extreme value (GEV) distribution. Therefore, the GEV distribution could be used in order to find the best fitting extreme value distribution. However, there are problems with this approach as discussed by e.g. Næss [41]. Therefore, the Gumbel distribution will only be discussed henceforth.

2.5.1 Gumbel distribution

The cumulative distribution function (CDF) of the Gumbel distribution reads as follows

$$F(x;\alpha,\,\beta) = e^{-e^{-\frac{(x-\alpha)}{\beta}}} \tag{2.67}$$

where α and $\beta > 0$ are parameters that is found through the simulated data by e.g. the method of moments. It has been shown that the parameters α and β can be expressed by the mean value μ and the standard deviation σ [45] such that

$$\begin{aligned} \beta &= \sqrt{6}\sigma/\pi \\ \alpha &= \mu - \beta\gamma, \end{aligned} \tag{2.68}$$

where γ denotes Euler's constant. The PDF of the Gumbel distribution is found by taking the derivative of the CDF with respect to x. The PDF of the Gumbel distribution then reads

$$f(x;\alpha,\,\beta) = \frac{1}{\beta} e^{-\left(\frac{(x-\alpha)}{\beta} + e^{-\frac{(x-\alpha)}{\beta}}\right)}. \tag{2.69}$$

After the response calculation are done through time series simulations with N simulations, there are N maximum values (for each degree of freedom). Let $x_1,\, x_2,\, ...,\, x_N$ be the (absolute) maximum realisation for each simulation in N in increasing order, such that $x_{(1)} \leq x_{(2)} \leq ... \leq x_{(N)}$. Let $X_{(m)}$ be a random variable corresponding to $x_{(m)}$ $m \in \{1,\, 2,\, ...,\, N\}$. Then, a realisation of $X_{(m)} = x$ implies that there are $N - m$ values greater than x, and there are $m - 1$ values less than or equal to x. Using the results above it can be shown that the expected value of the CDF of a realisation $X_{(m)}$ is written as [46][47]

$$\mathrm{E}[F(X_{(m)})] = \frac{m}{N+1}. \tag{2.70}$$

This result can be used to estimate the parameters of the Gumbel distribution by plotting all realisations on the following form $(-\log(-\log(\frac{m}{N+1})),\, x_m)$ and utilising that the negative logarithm of the negative logarithm of the CDF of the Gumbel distribution reads

$$-\log\left(-\log(F(x;\alpha,\,\beta))\right) = \frac{1}{\beta}x - \frac{\alpha}{\beta}. \tag{2.71}$$

Taking twice the negative logarithm of F renders a linear function which can be found by linear regression of the points $(-\log(-\log(\frac{m}{N+1})),\, x_m)$. The CDF of F can be determined after the estimated parameters $\hat{\alpha}$ and $\hat{\beta}$ have been obtained. Subsequently, an estimated response $\hat{R}$ corresponding to a prescribed percentile, $p \in [0,\, 1]$, can be obtained. However, there are uncertainties related to the aforementioned result as it is based on a limited number of simulations. Therefore, it is useful to employ the parametric bootstrapping method [48][49] in order to calculate an estimated confidence interval for the estimated response $\hat{R}$.

2.5.2 Confidence interval

The parametric bootstrapping method uses all the data points obtained earlier, i.e. $\mathbf{x} = (x_1,\, x_2,\, ...,\, x_N)$, or rather, it uses the estimated parameters $\hat{\alpha}$ and $\hat{\beta}$ and the corresponding Gumbel distribution $F(x;\hat{\alpha},\,\hat{\beta})$. New samples $\mathbf{x}_j^*$ $j \in \{1,\, 2,\, ...,\, n\}$ are then generated from a new random variable X^* with the distribution function $F(x;\hat{\alpha},\,\hat{\beta})$. Each new sample $\mathbf{x}_j^*$ contains N independent observations of X^* and each $\mathbf{x}_j^*$ is then fitted to a new Gumbel distribution $F(x;\hat{\alpha}_j^*,\,\hat{\beta}_j^*)$ where a new estimate of the response $\hat{R}_j^*$ can be obtained. An approximation of the confidence interval is then given by

$$\left(\, \hat{R} - w_{q/2}\sigma_R^*,\quad \hat{R} + w_{q/2}\sigma_R^* \,\right) \tag{2.72}$$

where $w_{q/2}$ denotes the $100(1 - q/2)\%$ standard normal fractile and σ_R^* is the standard deviation of the new response samples

$$\sigma_R^* = \sqrt{\frac{1}{n-1}\sum_{j=1}^{n}\left(\hat{R}_j^* - \bar{R}^*\right)^2} \tag{2.73}$$

$$\bar{R}^* = (1/n)\sum_{j=1}^{n}\hat{R}_j^* \tag{2.74}$$

However, the true distribution may be generated by using several thousands bootstrap samples [46]. By ordering the estimates $\hat{R}_j^*$ in increasing order, a $100(1-q)\%$ confidence interval is then given as

$$\left(\hat{R}_l^*, \quad \hat{R}_u^*\right) \tag{2.75}$$

where $l = [qn/2]$ and $u = [(1-q/2)n]$ ($[\cdot]$ denotes the integer part of $\cdot$).

2.6 Extreme value analysis frequency domain

The frequency domain response does not have any realisations of the actual response of the bridge, rather it describes the response in statistical terms. The probability distribution of the largest peak for a time interval T follows a Poisson distribution. The CDF of the largest peak is equal to the probability of the maximum value of a function $y(t)$, where t goes from 0 to T, is less or equal to some threshold a, which can be expresses as:

$$\begin{aligned} P_{Ma}(a) &= \text{Prob}\{\max\{y(t),\, 0 \le t \le T\} \le a\} \\ &= \exp\left(-v_y^+(a)T\right) \end{aligned} \tag{2.76}$$

where $v_y^+(a)$ is the up-crossing rate defined as:

$$v_y^+(a) = \frac{1}{2\pi}\frac{\sigma_{\dot{y}}}{\sigma_y}\exp\left(-\frac{1}{2}\left(\frac{a}{\sigma_y}\right)^2\right). \tag{2.77}$$

Here, σ_y and $\sigma_{\dot{y}}$ is the standard deviation of the function $y(t)$ and its derivative $\dot{y}$, respectively. The standard deviation of the frequency response can be obtained by utilising one of the most important property of the power spectra. That is, for some process y, the integral of the power spectra $S_y(\omega)$ over the entire frequency domain is equal to the variance, or $E[y^2]$, of the process [15]. The standard deviation of y can be obtained from the following:

$$\sigma_y = \sqrt{\int_{-\infty}^{\infty} S_y(\omega)d\omega}. \tag{2.78}$$

The standard deviation of the derivative of the frequency response can be obtained by utilising another property of the power spectra:

$$S_{\dot{y}}(\omega) = \omega^2 S_y(\omega), \tag{2.79}$$

such that the standard deviation of the derivative of the response reads as follows:

$$\sigma_{\dot{y}} = \sqrt{\int_{-\infty}^{\infty} \omega^2 S_y(\omega) d\omega}. \tag{2.80}$$

The up-crossing rate of any response given by the power spectra $\mathbf{S}_R(\omega)$ can be determined through Eq. (2.78) and Eq. (2.80). Subsequently, the CDF of the largest peak can be established for a given time interval T. The CDF of the largest peak can then be used to obtain a response $\hat{R}$ for a given percentile $p \in [0, 1]$ such that for an arbitrary peak response, R, the probability of exceeding $\hat{R}$ is $100(1-p)$ %.

Figure 5: The Hålogaland Bridge viewed from the South-East, photo from Statens Vegvesen [50].

3 Methodology

The theory presented in the previous chapter is the theoretical foundation used to calculate the buffeting response of the Hålogaland Bridge in frequency and time domain, in addition to the flutter stability and the extreme value analysis of the buffeting response. This chapter outlines how the theory was used in order to obtain the desired results as well as a short introduction of the Hålogaland bridge and its properties.

3.1 Introduction

The Hålogaland Bridge is a suspension bridge located in the vicinity of Narvik in northen Norway. The bridge, which was finished in the end of 2018, crosses the Rombak fjord and as a result, the European route E6 was shortened by 18 km and traffic was redirected away from an area prone to accidents. The Hålogaland Bridge is currently the second longest bridge in Norway, with a main span of 1145 m [50]. The bridge deck is made of a steel box girder and the cross section, which is similar to the one shown in Figure 1, has a total width of 18.6 m and a height of 3 m [51]. The girder is carried by hangers connected to two suspension cables which are supported by a tower in each end. Figure 5 shows the finished Hålogaland Bridge.

3.2 Abaqus model

A finite element model of the Hålogaland Bride have been modelled in Abaqus [52] and was provided by Øyvind Wiig Petersen. The model is shown in Figure 6. The model was used to extract the modal features of the Hålogaland Bridge, i.e. modal mass, mode shapes and the corresponding natural frequencies. The first eighth mode shapes of the horizontal, vertical and torsional mode shapes are illustrated in Figure 7 and the corresponding dynamic properties are listed in Table 1. Note that the eighth torsional mode correspond to the

80th mode shape while the eighth horizontal and vertical modes correspond to the 19th and 13th mode shape, respectively. Subsequently, many mode shapes need to calculated in order to obtain sufficiently many torsional modes. Therefore, a total of 100 mode shapes was extracted from the Abaqus model and used in the buffeting response calculations.

The Abaqus model contains 573 nodes along the entire length of the bridge girder. This corresponds to approximately 2 m long elements. The number of nodes were however reduced to 120 nodes along the main span, which gives an approximate of 10 m per element, in order to reduce the computational costs of the buffeting response calculations.

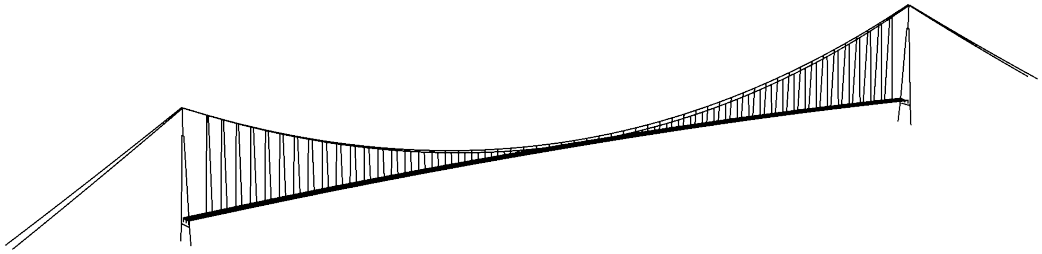


Figure 6: Abaqus model of the Hålogaland Bridge

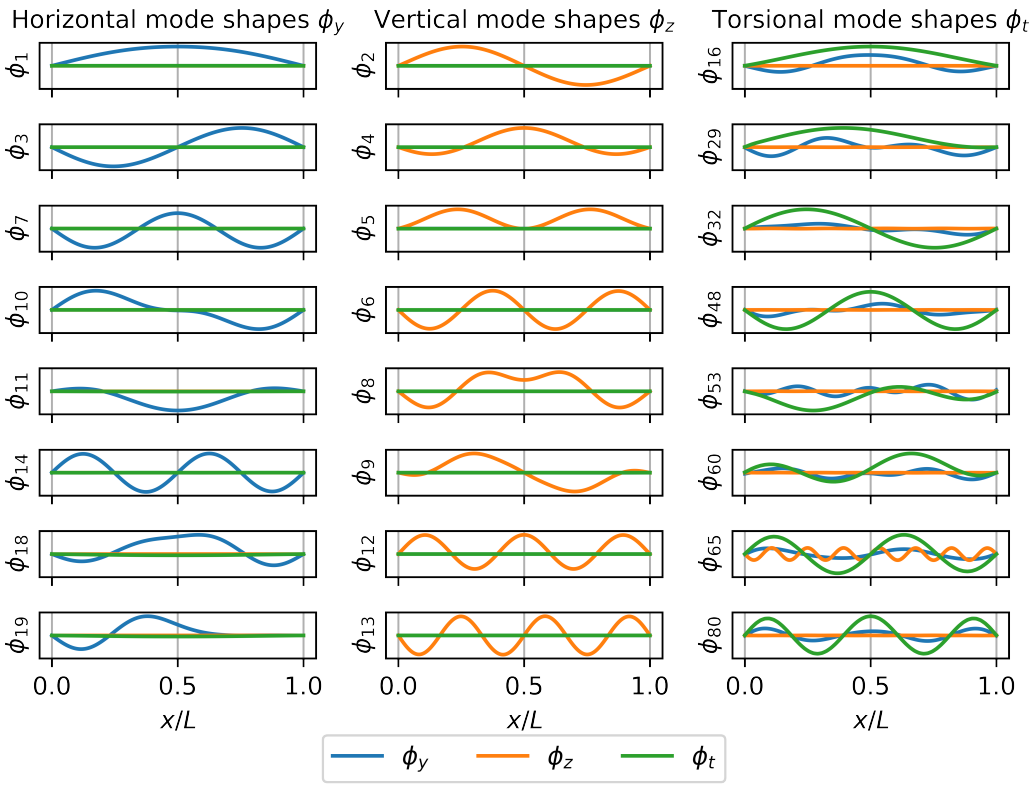


Figure 7: The eighth first horizontal, vertical and torsional mode shapes of the Hålogaland Bridge

Table 1: Dynamic properties of the Hålogaland Bridge

Mode nr. i	Mode type	Natural frequency ω_i (rad/s)	(Assumed) damping ratio ξ_i (-)
1	Horizontal	0.338	0.01
2	Vertical	0.728	0.01
3	Horizontal	0.752	0.01
4	Vertical	0.905	0.01
5	Vertical	1.293	0.01
6	Vertical	1.367	0.01
7	Horizontal	1.433	0.01
8	Vertical	1.680	0.01
9	Vertical	1.737	0.01
10	Horizontal	1.755	0.01
11	Horizontal	1.782	0.01
12	Vertical	1.798	0.01
13	Vertical	2.187	0.01
14	Horizontal	2.497	0.01
16	Torsional	2.752	0.01
18	Horizontal	2.835	0.01
19	Horizontal	2.908	0.01
29	Torsional	3.324	0.01
32	Torsional	3.676	0.01
48	Torsional	5.547	0.01
53	Torsional	5.965	0.01
60	Torsional	6.557	0.01
65	Torsional	7.216	0.01
80	Torsional	8.899	0.01

3.3 Wind tunnel testing

As mentioned earlier, in order to obtain the aerodynamic derivatives and the load coefficients it is necessary to perform wind tunnel testing of a down-scale section model of the bridge. The wind tunnel data presented in this thesis was generated at the wind tunnel testing laboratory at NTNU by Solstad and Onstad [53] in 2022. They performed the testing on a section model with a scale of 1:50. The down-scaled Hålogaland model had a height of 0.06 m, width of 0.372 m and was 2.68 m long. Figure 8 show the wind tunnel testing of a bridge girder section model similar to the Hålogaland Bridge. As the wind only comes from one direction in the wind tunnel, the section model has to be rotated in order to simulate different angle of attacks. Thus it is possible to measure the aerodynamic forces acting on the section model for different angles of attack.

3.3.1 Aerodynamic derivatives

The file containing the processed data from the wind tunnel testing regarding the aerodynamic derivatives contains the values of the dimensionless aerodynamic derivatives and the corresponding mean wind velocity and reduced velocity for which they are obtained at. The data consist of discrete values of the aerodynamic derivatives which are within a limited range of reduced velocities. It is necessary to obtain the aerodynamic derivatives as continuous functions in order to perform the buffeting response calculations. This can be

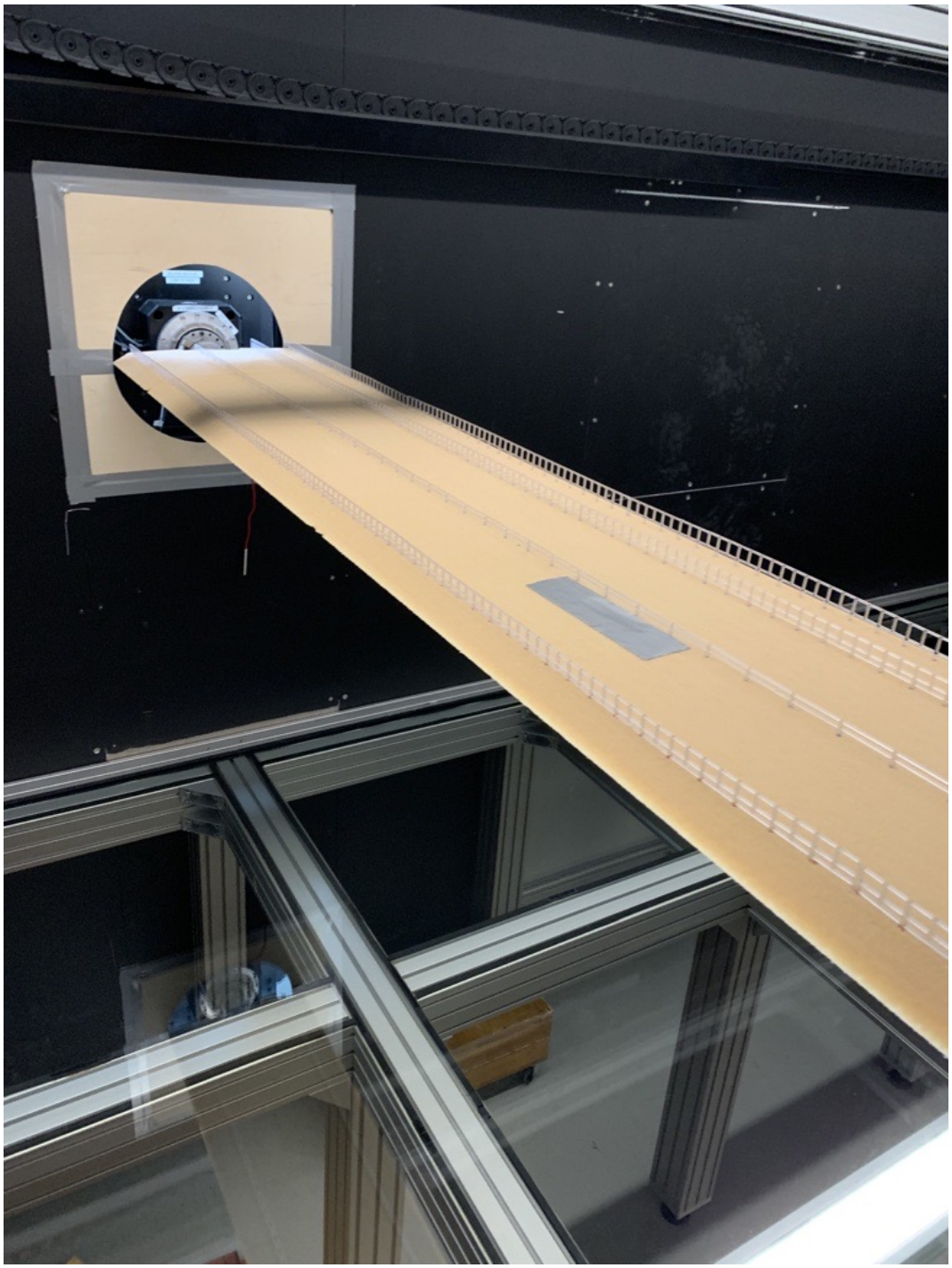


Figure 8: Wind tunnel testing of a bridge girder section model. Courtesy of NTNU.

done by fitting the rational functions in Eq. (2.35) to the transfer functions in Eq. (2.33). The first step in this process is to determine the number of rational terms to include. As discussed earlier, this curve fitting can be a tricky process due to the consequences the number of included terms have and the limited amount of experimental data. Investigations into the optimal number of terms to include have not been presented nor pursued as it is out of scope of this thesis. Therefore, the number of terms were simply set to three terms for describing the rational functions, i.e. $\mathbf{a}_1$, $\mathbf{a}_2$, $\mathbf{a}_4$ and d. In addition, d was assumed equal to one by recommendation from Ole Øiseth. This turned out to give a good fit and further investigation to optimise d were not made.

Now, lets take a closer look as to how $\mathbf{a}_1$, $\mathbf{a}_2$ and $\mathbf{a}_4$ can be determined through the method of least squares by employing the Moore-Penrose inverse, also known as the pseudoinverse. First, note that the coefficient matrices $\mathbf{a}_i$ $i \in \{1, 2, 4\}$ are 3x3 matrices such that they can be expressed as follows

$$\mathbf{a}_i = \begin{pmatrix} a_i^{11} & a_i^{12} & a_i^{13} \\ a_i^{21} & a_i^{22} & a_i^{23} \\ a_i^{31} & a_i^{32} & a_i^{33} \end{pmatrix}. \tag{3.1}$$

Each element in the coefficient matrices are connected to the corresponding element in the frequency response matrix (see Eq. (2.33)). This implies that in order to find the coefficients a_i^{11}, only the data points from P_1^* and P_4^* need to be considered, for a_i^{12} the data points P_5^* and P_6^*, and so forth. Lets take a closer look as to how a_i^{11} can be determined. Assuming there are N data points obtained through the wind tunnel testing, the following matrix equation can be written with respect to the first elements in the rational functions (see Eq. (2.35)) and the frequency response matrix (the factor $1/2\rho V^2$ is a common factor of both equations and cancels each other out):

$$\begin{pmatrix} (K^2P_1^*)_1 \\ (K^2P_1^*)_2 \\ \vdots \\ (K^2P_1^*)_N \\ (K^2P_4^*)_1 \\ (K^2P_4^*)_2 \\ \vdots \\ (K^2P_4^*)_N \end{pmatrix} = \begin{pmatrix} 0 & 1 & \mathrm{Im}\left(\frac{iK_1}{iK_1+d}\right) \\ 0 & 1 & \mathrm{Im}\left(\frac{iK_2}{iK_2+d}\right) \\ \vdots & \vdots & \vdots \\ 0 & 1 & \mathrm{Im}\left(\frac{iK_N}{iK_N+d}\right) \\ 1 & 0 & \mathrm{Re}\left(\frac{iK_1}{iK_1+d}\right) \\ 1 & 0 & \mathrm{Re}\left(\frac{iK_2}{iK_2+d}\right) \\ \vdots & \vdots & \vdots \\ 1 & 0 & \mathrm{Re}\left(\frac{iK_N}{iK_N+d}\right) \end{pmatrix} \begin{pmatrix} a_1^{11} \\ a_2^{11} \\ a_4^{11} \end{pmatrix} \tag{3.2}$$

The equation can be written in a tidier manner:

$$\mathbf{y} = \mathbf{A}\mathbf{x} \tag{3.3}$$

where $\mathbf{x}$ denotes the unknown coefficients to be determined. For a matrix $\mathbf{P}$, the pseudoinverse is defined as follows:

$$\mathbf{P}^+ = \mathbf{P}^T \left(\mathbf{P}\mathbf{P}^T\right)^{-1}. \tag{3.4}$$

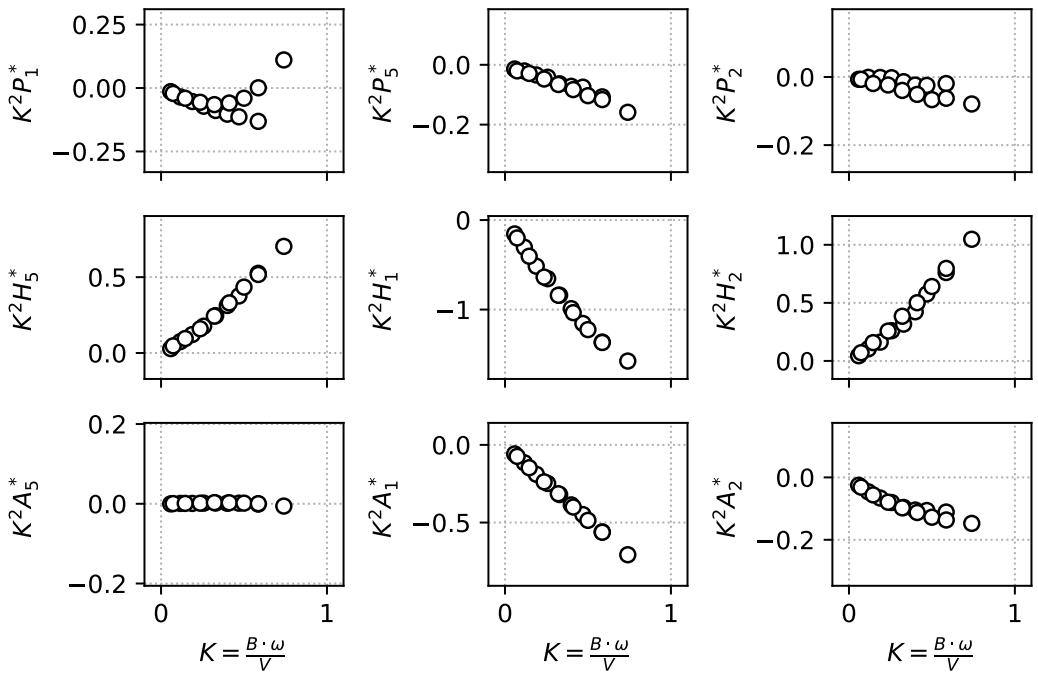


Figure 9: Dimensionless aerodynamic derivatives, as function of reduced circular frequency, obtained from wind tunnel testing. These aerodynamic derivatives are associated with the aerodynamic damping.

Applying the pseudoinverse to Eq. (3.3) then renders the following expression for $\mathbf{x}$:

$$\mathbf{x} = \mathbf{A}^T \left(\mathbf{A}\mathbf{A}^T \right)^{-1} \mathbf{y}. \tag{3.5}$$

The exact same procedure can be followed in order to obtain the other coefficients. The experimental data obtained from the wind tunnel testing of the aerodynamic derivatives related to the aerodynamic damping and stiffness is shown in Figure 9 and Figure 10, respectively. The figures show the aerodynamic derivatives multiplied by the reduced frequency squared as functions of reduced frequency, which is convenient for the method described for fitting the rational functions.

3.3.2 Load coefficients

The experimental data obtained from the wind tunnel testing regarding the drag, lift and moment coefficients is presented in Figure 14. The mean angle of attack is assumed equal to zero, such that the load coefficients $\overline{C}_D$, $\overline{C}_L$ and $\overline{C}_M$ are obtained $\alpha = 0$, while C'_D, C'_L and C'_M are obtained as the slope of the derivative of the load coefficients at $\alpha = 0$.

3.4 Verification of the modelling of self-excited forces

It is possible to check if the curve fitted expressions for the rational functions is properly fitted by the use of a state space model to obtain the self-excited forces. By prescribing an arbitrary motion for a single reduced circular frequency, the self-excited forces can be obtain through a state space model and through to Eq. (2.12).

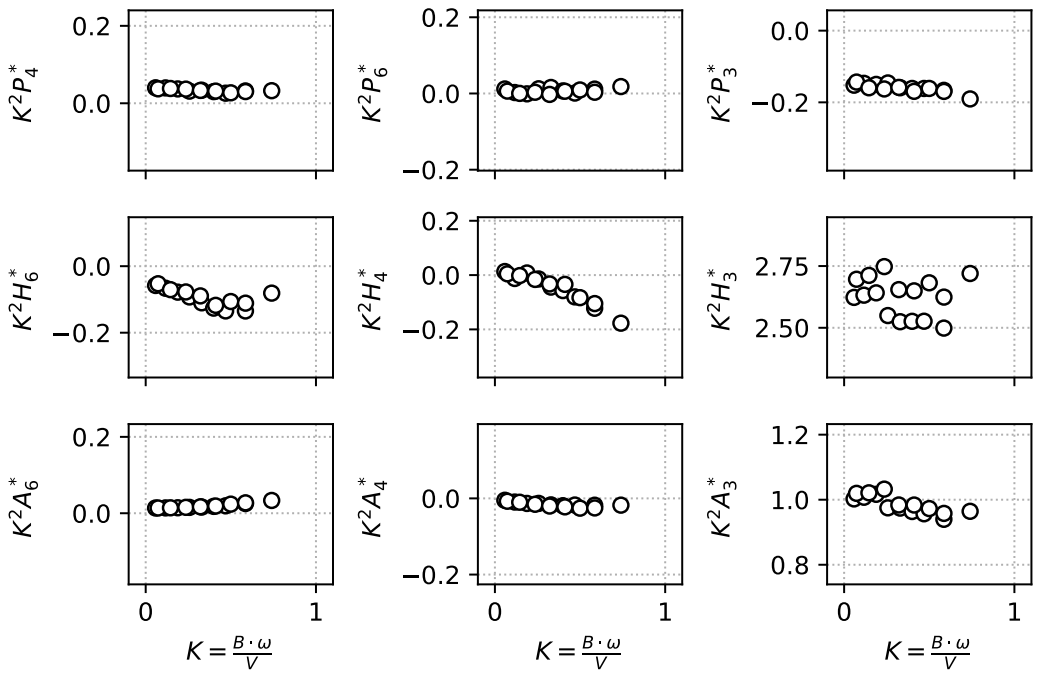


Figure 10: Dimensionless aerodynamic derivatives, as function of reduced circular frequency, obtained from wind tunnel testing. These aerodynamic derivatives are associated with the aerodynamic stiffness.

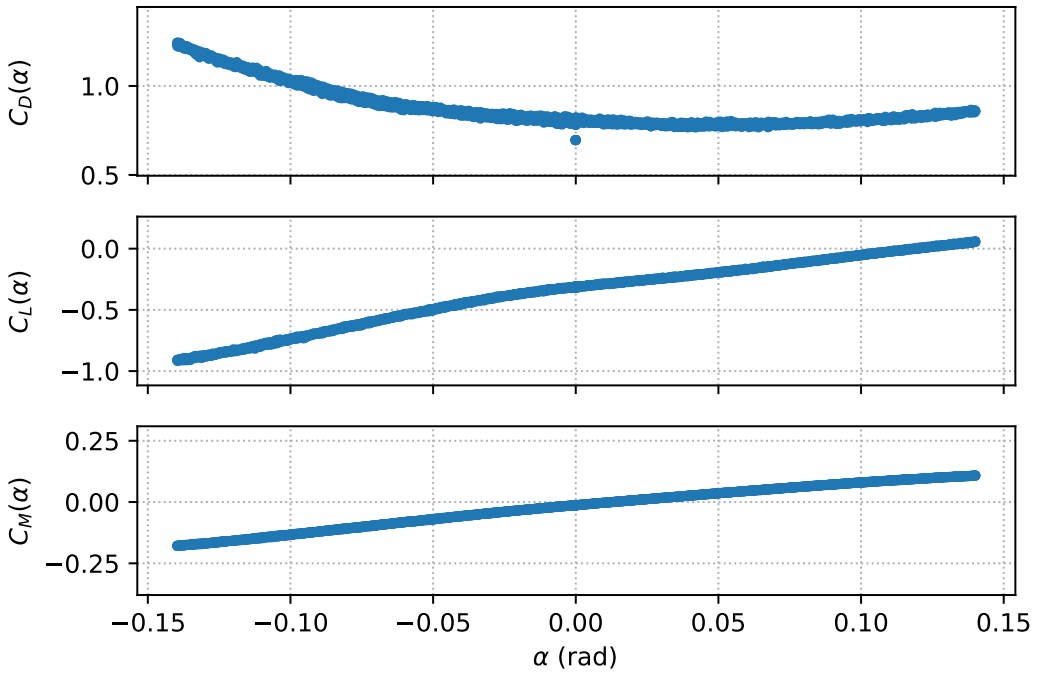


Figure 11: Load coefficients obtained from wind tunnel testing of different angles α. Top figure show the drag coefficient, middle figure show the lift coefficient, while the bottom figure show the pitching moment coefficient.

In order to obtain the state space model for describing the self-excited forces, a procedure similar to the one presented in Section 2.3.2 can be applied. The state space model can be set up using the general state space model, given in Eq. (2.54), to find the unsteady part of the of the self-excited forces. As $\mathbf{a}_i$ and d are known from the curve fitting and a random motion is prescribed, the following set of equations can be used to describe the self-excited forces as a continuous function:

$$\begin{aligned} \dot{\mathbf{x}}(t) &= \mathbf{D}_c\mathbf{x}(t) + \mathbf{E}_c\dot{\mathbf{u}}(t) \\ \mathbf{z}(t) &= \mathbf{Q}_c\mathbf{x}(t) \end{aligned} \tag{3.6}$$

$$\mathbf{q}_{se}(t) = \frac{1}{2}\rho V^2 \left(\mathbf{a}_1\mathbf{u}(t) + \mathbf{a}_2\dot{\mathbf{u}}(t)\right) + \mathbf{z}(t) \tag{3.7}$$

where,

$$\mathbf{D}_c = -\frac{dV}{B}\mathbf{I} \tag{3.8}$$

$$\mathbf{E}_c = \mathbf{I} \tag{3.9}$$

$$\mathbf{Q}_c = \frac{1}{2}\rho V^2\mathbf{a}_4 \tag{3.10}$$

$$\mathbf{x} = \left(\mathbf{u}(t) - \frac{dV}{B}\int_{-\infty}^{t} e^{-\frac{dV}{B}(t-\tau)}\mathbf{u}(\tau)d\tau\right). \tag{3.11}$$

Here $\mathbf{I}$ is the (3x3) identity matrix. The discretised state space model can be established similar to the discretisation of the state space model of the equation of motion:

$$\begin{aligned} \dot{\mathbf{x}}_{k+1} &= \mathbf{D}_d\mathbf{x}_k + \mathbf{E}_d\dot{\mathbf{u}}_k \\ \mathbf{z}_k &= \mathbf{Q}_d\mathbf{x}_k \end{aligned} \tag{3.12}$$

$$\mathbf{q}_k = \frac{1}{2}\rho V^2 \left(\mathbf{a}_1\mathbf{u}_k + \mathbf{a}_2\dot{\mathbf{u}}_k\right) + \mathbf{z}_k \tag{3.13}$$

where $\mathbf{D}_d$, $\mathbf{E}_d$ and $\mathbf{Q}_d$ can be established through Eq. (2.56). Eq. (3.12) and Eq. (3.13) can then be used to simulate the self-excited forces for an arbitray motion.

3.5 Calculating the buffeting response

The buffeting response of the Hålogaland Bridge was calculated using 100 mode shapes. All calculations have been carried out in Python and are available in the appendix. The buffeting response is calculated for horizontal, vertical and rotational displacement at the midspan of the bridge.

Table 2: Wind field parameters

ρ	A_u	A_w	C_{uy}	C_{wy}	L_1(m)	z_1(m)	z(m)	${}^xL_u$(m)	${}^xL_w$(m)	I_u	I_w
1.25	6.8	9.4	10.0	6.5	100.0	10.0	50.0	$L_1(z/z_1)^{0.3}$	${}^xL_u/12$	0.15	$I_u/4$

The modal mass, natural frequencies and corresponding mode shapes are obtained through the Python script prepared by Øyvind Wiig Petersen. The Python script (see Appendix A.4) reads the structural properties from a HDF5 data file which contains the relevant information from the ODB (output database) file from the Abaqus model. The modal properties are readily obtained in NumPy arrays [54]. There are in total 573 nodes along the bridge deck from the Abaqus model and six degrees of freedom (DOF) in each node, namely, longitudinal , horizontal and vertical translation, in addition to rotations, i.e. torsion, vertical bending and horizontal bending. However, the script sorts out the relevant DOF, i.e. horizontal translation, vertical translation and torsion. The number of nodes were reduces from 573 to 120 in order to reduce the computational time. This results in roughly 10 m between each node rather than the original 2 m.

Having the modal mass and natural frequencies, the modal stiffness matrix was obtained by the following equation

$$\widetilde{\mathbf{K}} = diag\left(\omega_n^2 \widetilde{M}_n\right), \tag{3.14}$$

where ω_n $n \in \{1, 2, ..., 100\}$ is the natural circular frequency and $\widetilde{M}_n$ $n \in \{1, 2, ..., 100\}$ denote the diagonal terms in the modal mass matrix. The damping ratio was assumed $\xi = 1$ % in all modes, and such the modal damping matrix was found through the following equation

$$\widetilde{\mathbf{C}} = diag\left(2\widetilde{M}_n \omega_n \xi\right). \tag{3.15}$$

Using the static load coefficients and the sectional dimensions of the Hålogaland Bridge, the buffeting load matrix $\mathbf{B}_q$ (Eq. (2.6)) can be obtained for an arbitrary mean wind velocity V. For the same arbitrary V, the cross-spectral density of the turbulence components $\mathbf{S}_V^+$ (Eq. (2.28)) can readily obtained by inserting the assumed wind field parameters, given in Table 2, into Eq. (2.8) and Eq. (2.10).

3.5.1 Calculations in frequency domain

Having obtained the mode shapes $\mathbf{\Phi}$, the buffeting load matrix $\mathbf{B}_q$ and the cross-spectral density of the turbulence components $\mathbf{S}_V^+$, the cross-spectral density of the generalised buffeting load (Eq. (2.27)) can be calculated by numerical integration. The integration is performed with a coarser frequency axis and then the co-spectra is interpolated back onto the original frequency axis. This reduces the number of calculations needed and subsequently the computational time. This can be done as the frequency steps $\Delta\omega$ needed to capture the narrow peaks in the power spectra of the buffeting response is much smaller than what is needed for the power spectra of the buffeting load.

In order to obtain the frequency response matrix (Eq. (2.31)), the modal aerodynamic damping (Eq. (2.23)) and stiffness (Eq. (2.24)) matrices need to be determined. The

aerodynamic damping and stiffness matrix can be established with the mode shapes $\mathbf{\Phi}$ and previously obtained curve fits of the aerodynamic derivatives. The frequency response matrix can then readily be obtained having established the modal aerodynamic damping and stiffness matrices. The cross-spectral density of the response $\mathbf{S}_R$ can then be calculated as in Eq. (2.30).

Note that the frequency response matrix and the co-spectra of the generalised buffeting load are both matrices as functions of the mean wind velocity V and the circular frequency ω, and subsequently, the co-spectra of the response is also a function of V and ω. This means that these matrix multiplications and numerical integrations need to happen for all frequency components ω_k and all mean wind velocity components V_k. This is very computationally expensive and optimisations, such as the integration of a coarser frequency axis to find the co-spectra of the generalised buffeting load is rather convenient.

Integrating the co-spectra of the response over the frequency axis renders the covariance matrix of the response. The covariance matrix can be used to calculate the standard deviation and correlation coefficient between the different responses as the standard deviation is equal to the square root of the variance. The correlation factor, ρ_{xy}, of two processes x and y is equal to the covariance of x and y divided by the product of the standard deviation of x and y:

$$\rho_{xy} = \frac{\mathrm{cov}(x,\, y)}{\sigma_x \sigma_y}. \tag{3.16}$$

3.5.2 Calculations in time domain

The buffeting response in time domain also needs to be calculated for an array of mean wind velocities. Additionally, time series of the wind field must be simulated for each mean wind velocity in order to obtain the turbulence vector $\mathbf{v}$ and calculate the buffeting response.

The buffeting response can be calculated through the discretised state space model given in Section 2.3.3. First, the state space matrices in Eq. (2.58) need to be established. From Section 3.3.1 it was determined that the transfer functions would be described by the use of the coefficient matrices $\mathbf{a}_1$, $\mathbf{a}_2$ and $\mathbf{a}_4$. The matrices, which have been established through the curve fitting of the aerodynamic derivatives, are then transformed to generalised matrices, $\widetilde{\mathbf{a}}_1$, $\widetilde{\mathbf{a}}_2$ and $\widetilde{\mathbf{a}}_4$ according to Eq. (2.40). The state space matrix $\mathbf{A}_c$, given in Eq. (2.58), then reads:

$$\mathbf{A}_c = -\begin{pmatrix} \mathbf{0} & -\mathbf{I} & \mathbf{0} \\ \widetilde{\mathbf{M}}^{-1}\left(\widetilde{\mathbf{K}} - \widetilde{\mathbf{a}}_1\right) & \widetilde{\mathbf{M}}^{-1}\left(\widetilde{\mathbf{C}} - \widetilde{\mathbf{a}}_2\right) & -\widetilde{\mathbf{M}}^{-1}\,\widetilde{\mathbf{a}}_4 \\ \mathbf{0} & -\boldsymbol{E} & \boldsymbol{D} \end{pmatrix}, \tag{3.17}$$

while $\mathbf{B}_c$ and $\mathbf{C}_c$ reads the same. These matrices can be transformed to the discretised state space model according to Eq. (2.56).

The Cholesky decomposition of the co-spectra of the turbulence components $\mathbf{S}_V^+$ are found through the SciPy function *cholesky* [55]. The Cholesky decomposition is found for a coarser frequency axis than the main simulation frequency axis, and then interpolated back to the main axis, as described in Section 2.3.4, in order to reduce the computational

cost. Another way to increase the computational speed is to zero-pad the inverse FFT to the next power of two as the FFT computes arrays with length equal to exponents of two faster than arrays of other lengths.

The number of Monte Carlo simulations was set to 20, and such, for each simulation, a new set of random phase angles between 0 and 2π was generated by the help of the NumPy function *random.rand*. With the random phase angles and the Cholesky decomposed matrix in place, the turbulence components, u_m and w_m, can readily be obtained according to Eq. (2.61). The turbulence components are then calculated for all 120 nodes along the main span in order to establish the turbulence vector $\mathbf{v}$. As no admittance functions are applied, the buffeting load matrix is not dependent on the circular frequency such that the buffeting load can be expressed as $\mathbf{q}_{buff}(t) = \mathbf{B}_q \mathbf{v}(t)$ and the generalised buffeting load can be calculated according to Eq. (2.64). The discretised state space model presented in Eq. (2.57) is then used, assuming $\mathbf{x}_0 = \mathbf{0}$, to calculate the buffeting response.

3.6 Stability limit

The flutter stability of the system was estimated by the eigenvalue analysis of the state space model, i.e. the second approach shown in Section 2.4. The eigenvalues of the state space matrix $\mathbf{A}_c$, which were established in the last section, are then calculated by iterating through different mean wind velocities V. The iterations was done in a while-loop, starting with relative small values of V and calculating the eigenvalues S_n of $\mathbf{A}_c$. The system is stable if all the real eigenvalues are negative, i.e. positive damping. If this is the case, V is slightly increased and a new set of eigenvalues are calculated for the new V. The process of increasing V continues until the maximum of the real eigenvalues becomes positive. The system have then become unstable and exhibit negative damping. When this happens the increase in V is reverted to the previous value and instead increased with half amount as compared to earlier. By doing this every time the maximum real eigenvalue is greater than zero, we get closer and closer to the critical mean wind velocity V_{cr}, which is readily obtained with sufficiently many iterations or a stop criteria.

3.7 Extreme response time domain

3.7.1 Fitting the extreme value distribution

After calculating 20 realisations of the buffeting response in time domain, the maximum response (in each DOF) is selected for each realisation and a specific mean wind velocity and put into an array. These values are then sorted in ascending order, $x_1 \leq ... \leq x_m \leq ... \leq x_{N=20}$. These values can then be plotted on the form presented in Section 2.5.1: $(-\log(-\log(\frac{m}{N+1})), x_m)$, and such, the parameters of the Gumbel distribution can be found by linear regression, in accordance with Eq. (2.71). Another way to determine the Gumbel parameters is through method of moments, using the mean and standard deviation of the array of the maximum responses. With the mean and standard deviation, the Gumbel parameters are readily obtainable using Eq. (2.68). Both methods are equally viable. However, the latter method was used to obtain the Gumbel parameters. The PDF and the CDF of the Gumbel distribution can then be calculated according to Eq. (2.69) and Eq. (2.67), respectively.

By setting the CDF of the Gumbel distribution $F(R)$ equal to some percentile p, an estimator of the peak response $\hat{R}$ can be obtained for which the exceedance rate of $\hat{R}$ is

equal to $100\,(1-p)$ %. The 95th percentile was used in order to obtain the results in this thesis, i.e. $p = 0.95$. However, since the Gumbel parameter are based on a relative small number of samples, there are (quite the) uncertainty in the calculated response. For this reason it is important to find the confidence interval of the peak response estimator $\hat{R}$.

3.7.2 Confidence interval

The confidence interval of the estimated peak response, $\hat{R}$, was estimated using the bootstrap method presented in Section 2.5.2. For the bootstrap method, 100 000 arrays with 20 elements each were randomly generated from the CDF of the Gumbel distribution with the estimated Gumbel parameters. The mean and the standard deviation of the arrays were then calculated and a new set of Gumbel parameters can be estimated for the corresponding arrays. With these new estimated parameters, a new estimator of the 95th percentile response $\hat{R}_j^*$ $j \in \{1,\ 2,\ ...,\ 100\ 000\}$ can obtained.

By sorting all the new estimators $\hat{R}_j^*$ in ascending order, $\hat{R}^*_{(1)} \leq \hat{R}^*_{(2)} \leq ... \leq \hat{R}^*_{(100\,000)}$, the $100\,(1-q)$ % confidence interval can be found according to Eq. (2.75). The confidence interval is chosen to be 95 % in this thesis, i.e. $q = 0.05$, meaning that the 95 % confidence interval is given by:

$$\left(\hat{R}^*_{(2\,500)},\quad \hat{R}^*_{(97\,500)} \right) \tag{3.18}$$

3.8 Extreme response frequency domain

The standard deviation of the response and the derivative of the response can readily be obtained after the co-spectra of the response $\mathbf{S}_R$ have been established, according to Eq. (2.78) and Eq. (2.80), respectively. With the standard deviation in place, the CDF of the largest peak P_{Ma} can then be obtained through Eq. (2.76). After P_{Ma} was established, the 95th percentile responses were obtained by solving the equation $P_{Ma} = 0.95$.

4 Results

This chapter presents the results derived from the theory, described in Section 2, and applied to the Hålogaland Bridge following the methodology described in Section 3.

4.1 Wind tunnel data

This section presents the result regarding the aerodynamic properties, i.e. the curve fitting of the aerodynamic derivatives, the load coefficients and the verification of the fitted rational functions.

4.1.1 Aerodynamic derivatives

Figure 12 and Figure 13 show the experimental data of the aerodynamic derivatives from the wind tunnel testing together with the curve fitting of the respective aerodynamic derivative. Figure 12 show the aerodynamic derivatives associated with the aerodynamic damping while Figure 13 show the aerodynamic derivatives associated with aerodynamic stiffness. A summary of the rational function coefficients describing the curve fits are presented in Table 3.

Table 3: Rational function coefficients obtained through curve fitting of the experimental aerodynamic derivative data.

i	a_i^{11}	a_i^{12}	a_i^{13}	a_i^{21}	a_i^{22}	a_i^{23}	a_i^{31}	a_i^{32}	a_i^{33}
1	0.056	0.003	-0.152	-0.066	0.030	2.657	0.012	-0.010	1.014
2	0.052	-0.219	-0.031	1.036	-1.875	1.464	0.041	-0.928	-0.057
4	-0.190	0.030	-0.070	-0.238	-0.647	-0.260	0.053	-0.048	-0.235

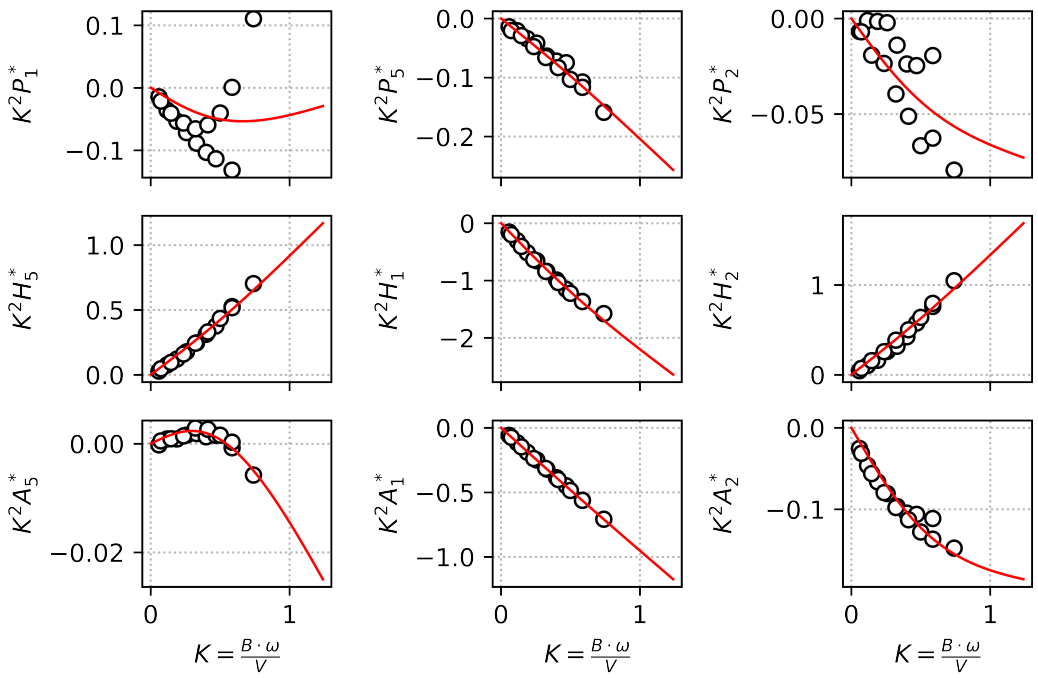


Figure 12: Curve fits of the rational functions defined in Eq. (2.35) to the experimentally determined dimensionless aerodynamic derivatives, as function of the reduced circular frequency. These aerodynamic derivatives are associated with the aerodynamic damping.

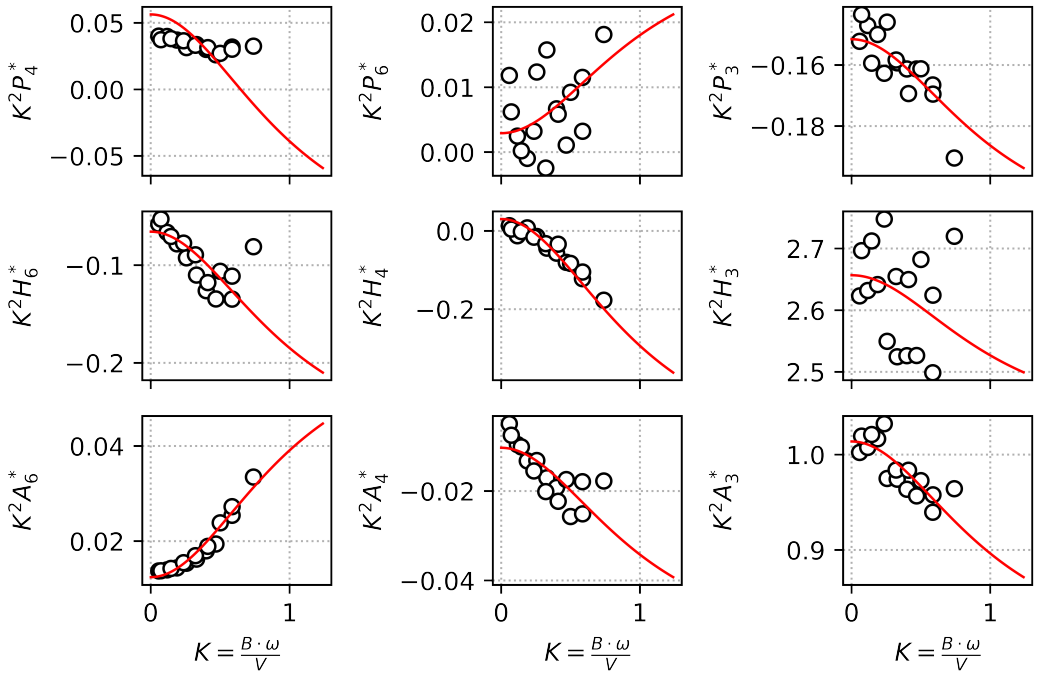


Figure 13: Curve fits of the rational functions defined in Eq. (2.35) to the experimentally determined dimensionless aerodynamic derivatives, as function of the reduced circular frequency. These aerodynamic derivatives are associated with the aerodynamic stiffness

4.1.2 Load coefficients

Figure 14 show the experimentally obtained load coefficient data of the drag, lift and pitching moment for a range of angles of attack. The estimated derivatives at zero angle of attack are also presented in the figures as the dashed orange lines. The slope of these dashed lines gives the load coefficients C'_D, C'_L and C'_M. These values, together with the average value of the load coefficients at $\alpha = 0$ are presented in Table 4.

Table 4: Load coefficients

$\overline{C}_D$	$\overline{C}_L$	$\overline{C}_M$	C'_D	C'_L	C'_M
0.81	-0.31	-0.01	-0.62	2.73	1.05

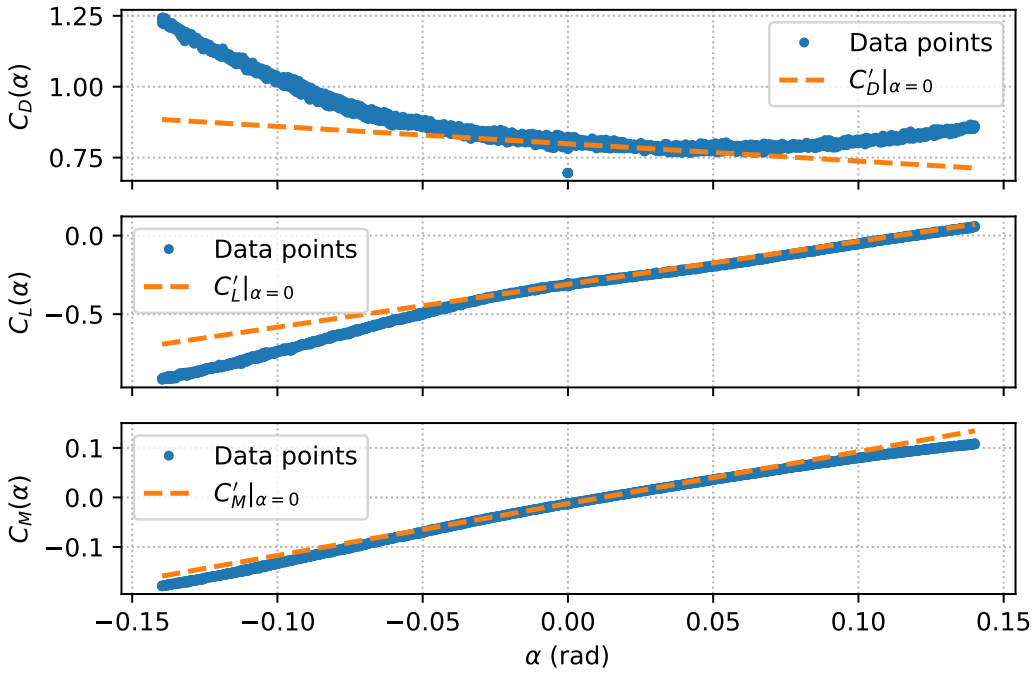


Figure 14: Approximation of the derivative of the load coefficients at angle $\alpha = 0$ with regard to the wind tunnel data.

4.1.3 Verification of the rational functions

Figure 15 display the motion induced forces caused by an arbitrary motion. The motion history for each DOF is shown in the left figure while the corresponding forces are shown in the right figure. The solid blue lines shows the motion induced forces obtained through the state space model given in Section 3.4. The motion is chosen as a single harmonic motion with a reduced circular frequency $K = 0.586$ such that the self-excited forces can additionally be calculated through Eq. (2.12) with the experimental wind tunnel data of the aerodynamic derivatives. The dashed orange lines in Figure 15 represent the self-excited forces obtained through Eq. (2.12).

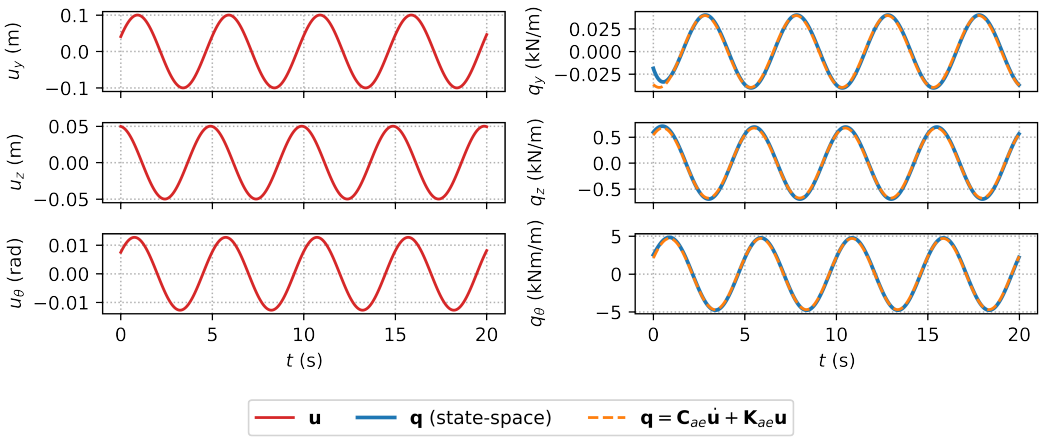


Figure 15: Simulation of motion induced forces of the Hålogaland Bridge.

4.2 Buffeting response

In this section, the results from the buffeting response calculations are presented. Both the time and frequency domain responses are presented and compared.

Figure 16 shows the comparison of the buffeting response, at the midspan, obtained through both frequency and time domain analysis. The figure is showing the response in a statistical sense, comparing the standard deviations and correlation factors of the frequency and time domain response as functions of the mean wind velocity. The figures from the upper left and diagonally down shows the standard deviations of the horizontal, vertical and torsional response, respectively. While the off-diagonal figures shows the correlation between the responses. Table 5 and Table 6 presents the values obtained from Figure 16 at the various mean wind velocities for the frequency and time domain solution, respectively.

Table 5: Standard deviations and correlation coefficients from frequency domain analysis.

V (m/s)	σ_{yy} (m)	σ_{zz} (m)	$\sigma_{\theta\theta}$ (rad)	ρ_{yz} (-)	$\rho_{y\theta}$ (-)	$\rho_{z\theta}$ (-)
10	0.034	0.014	0.0002	-0.257	0.482	0.129
30	0.536	0.151	0.0027	-0.137	0.586	0.084
50	1.674	0.410	0.0096	-0.062	0.615	0.092
60	2.543	0.591	0.0163	0.014	0.629	0.118
70	3.802	0.858	0.0281	0.088	0.603	0.117
78	4.495	1.119	0.0581	0.261	0.445	-0.011

Table 6: Standard deviations and correlation coefficients (mean value of 20 realisations) from time domain analysis.

V (m/s)	σ_{yy} (m)	σ_{zz} (m)	$\sigma_{\theta\theta}$ (rad)	ρ_{yz} (-)	$\rho_{y\theta}$ (-)	$\rho_{z\theta}$ (-)
10	0.042	0.014	0.0002	-0.326	0.541	0.073
30	0.474	0.140	0.0023	-0.122	0.577	0.084
50	1.511	0.389	0.0088	-0.054	0.599	0.118
60	2.309	0.566	0.0150	-0.004	0.611	0.107
70	3.383	0.783	0.0269	0.164	0.610	0.164
78	4.364	1.080	0.0519	0.267	0.041	0.041

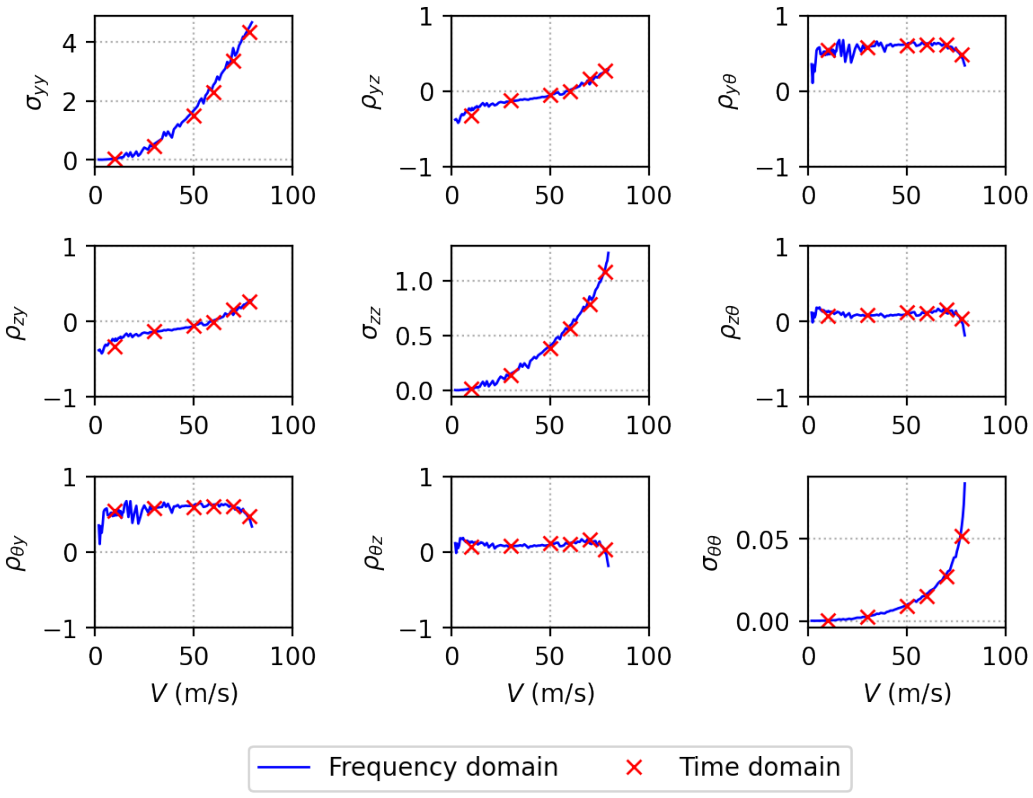


Figure 16: Predicted standard deviation and correlation coefficients of the buffeting response at the midspan. Red crosses are the results obtained from the average of 20 time domain simulations of 10 minutes, while the solid blue lines are the frequency domain response.

4.3 Comparing power spectra

By employing Welch's method [56] to the time domain response, it is possible to estimate the power spectra of the response from the buffeting response in the time domain. Welch's method is applied to all 20 time simulations and the 20 spectra is subsequently averaged out into one graph. Figure 17 presents the comparison of the estimated power spectra from the time domain response and the power spectra from the frequency domain response (Eq. (2.30)) for three different mean wind velocities; 30, 60 and 80 m/s. The figure consist of three sub-figures where the top figure show the auto-spectral density of the horizontal response, the middle figure show the auto-spectral density of the vertical response and the

bottom figure show the auto-spectral density of the torsional response.

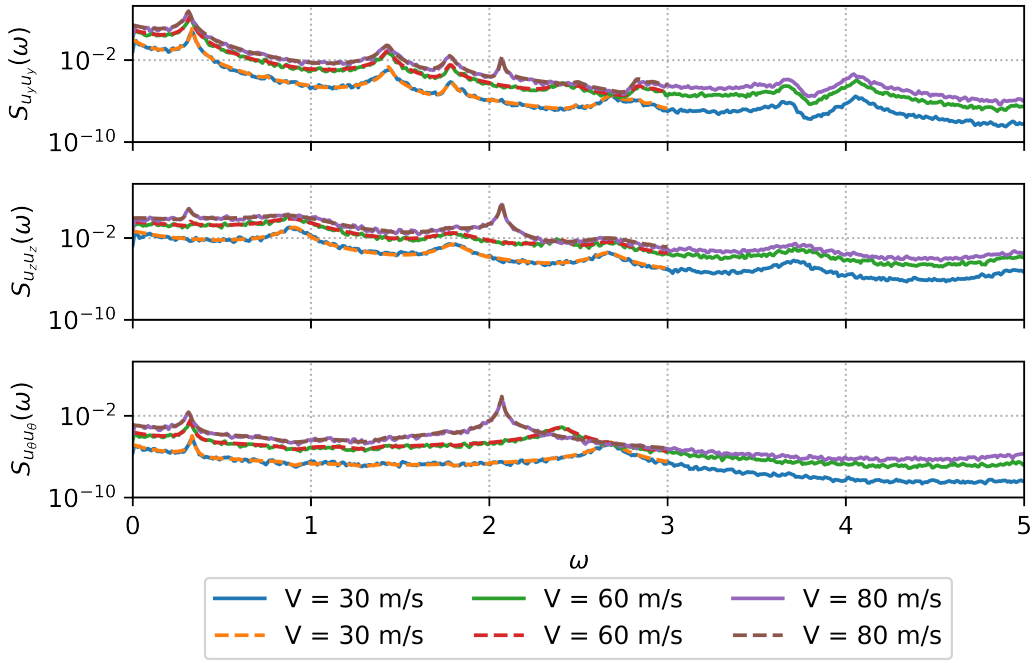


Figure 17: Comparison of the frequency response spectra and the Welch spectra estimated from the time domain response. The dashed lines indicate the frequency response spectra while the solid lines indicate the estimated response spectra.

4.4 Stability limit

In this section, the results regarding the flutter stability of the aeroelastic system are presented. As previously mentioned, the stability limit is found through the eigenvalue analysis of the state space matrix $\mathbf{A}_c$.

In Figure 18, the stability of the system is illustrated using only two mode shapes to calculate the flutter stability. The figure is just for illustration, as showing the same figure with all 100 mode shapes would be near indecipherable. This pair gives the lowest critical mean wind velocity for any given pair of mode shapes. The mode shapes are $\boldsymbol{\phi}_5$ and $\boldsymbol{\phi}_{16}$ which correspond to the third vertical and first torsional mode, respectively. This pair yields a critical mean wind velocity of 90.2 m/s. However, the stability limit is reduced to 80.69 m/s when all 100 mode shapes are taken into consideration.

Figure 19 shows the free vibration response at the midspan for two different mean wind velocities. One velocity is equal to the flutter stability, while the other is slightly higher than the stability limit. The response is calculated through the same state space model from the time domain analysis, i.e. Eq. (2.53), but with the generalised buffeting load set equal to zero, $\widetilde{\mathbf{q}}_{buff} = \mathbf{0}$, and an arbitrary initial condition, e.g. $\mathbf{x}_{k=0} = (1,\ 1,\ ...,\ 1)^T$.

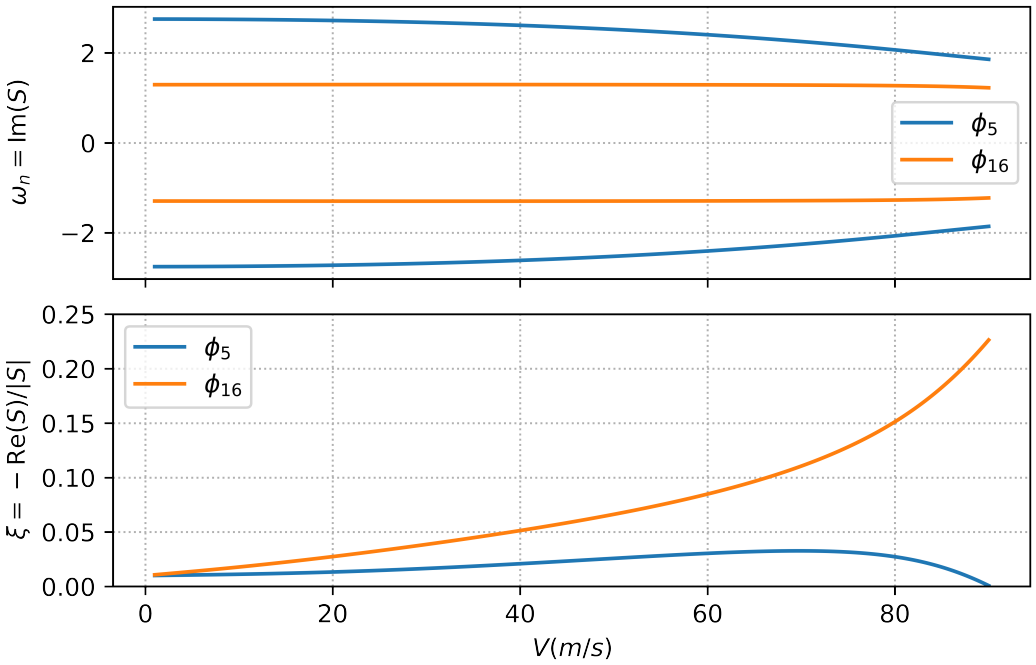


Figure 18: Eigenvalues of the system as a function of the mean wind velocity for one pair of mode shapes, namely $\boldsymbol{\phi}_5$ and $\boldsymbol{\phi}_{16}$. Top figure represent the imaginary part, e.g. the natural frequency, while the bottom figure represent the damping ratio. This pair of mode shapes yields a critical velocity $V_{cr} = 90.2$ m/s.

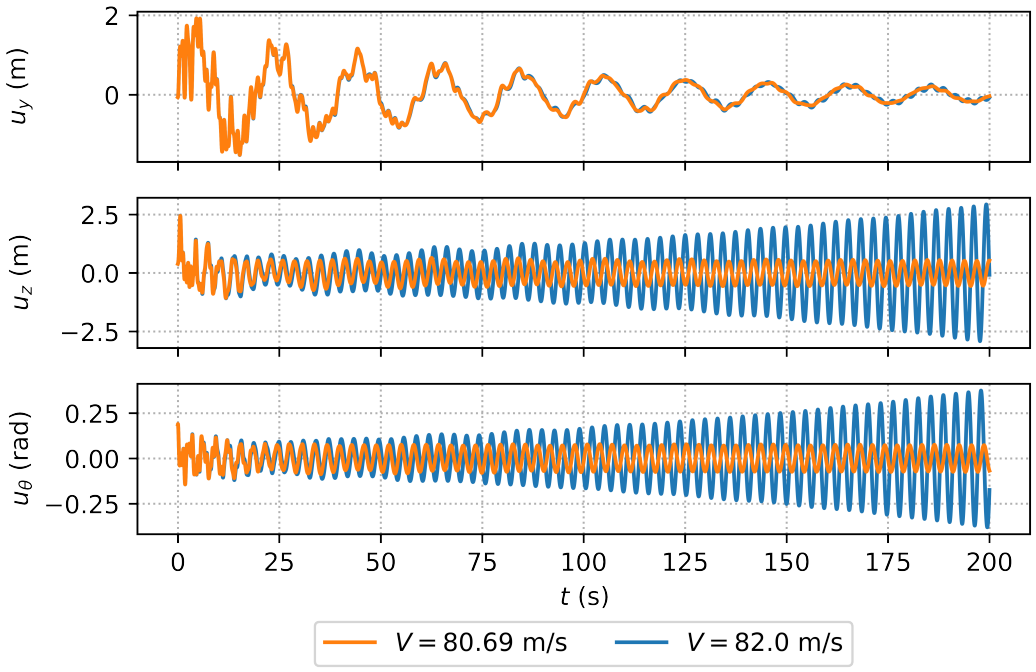


Figure 19: Free vibration response at the midspan of the bridge for two mean wind velocities; the stability limit and a slightly higher velocity.

4.5 Extreme response time domain

In this section, the results from extreme value analysis of the time domain response are presented. The following sections shows the results from the time domain analysis for three different mean wind velocities, namely, 30, 60, and 80 m/s. In each section, there is one figure showing the midspan response for one of the 20 time simulations, in addition to three figures showing the maximum response for each DOF from the simulations. These figures also show the estimated response at the 95th percentile and the confidence interval obtained through the bootstrap method.

A summary of the results is presented in Table 7. Here, $\hat{R}_j$ $j \in \{y,\ z,\ \theta\}$ are the estimated response corresponding to the 95th percentile and $\hat{R}_{j,conf}$ denote the 95 % confidence interval of the estimated peak response $\hat{R}_j$.

Table 7: Estimators of the 95th percentile response and the corresponding 95 % confidence interval.

V (m/s)	$\hat{R}_y$ (m)	$\hat{R}_{y,conf}$ (m)	$\hat{R}_z$ (m)	$\hat{R}_{z,conf}$ (m)	$\hat{R}_\theta$ (rad)	$\hat{R}_{\theta,conf}$ (rad)
30	1.88	(1.42, 2.23)	0.62	(0.49, 0.71)	0.010	(0.009, 0.011)
60	9.51	(7.13, 11.36)	2.26	(1.96, 2.50)	0.065	(0.055, 0.072)
80	17.40	(13.34, 20.60)	5.75	(4.59, 6.66)	0.395	(0.297, 0.473)

4.5.1 30 m/s

Figure 20 show the horizontal, vertical and torsional response at 30 m/s mean wind velocity for one of the 20 simulated time series. The (absolute) maximum response from the figure is extracted, along with the other 19 simulated time series, and used to make the left-side figure of Figure 21, Figure 22 and Figure 23.

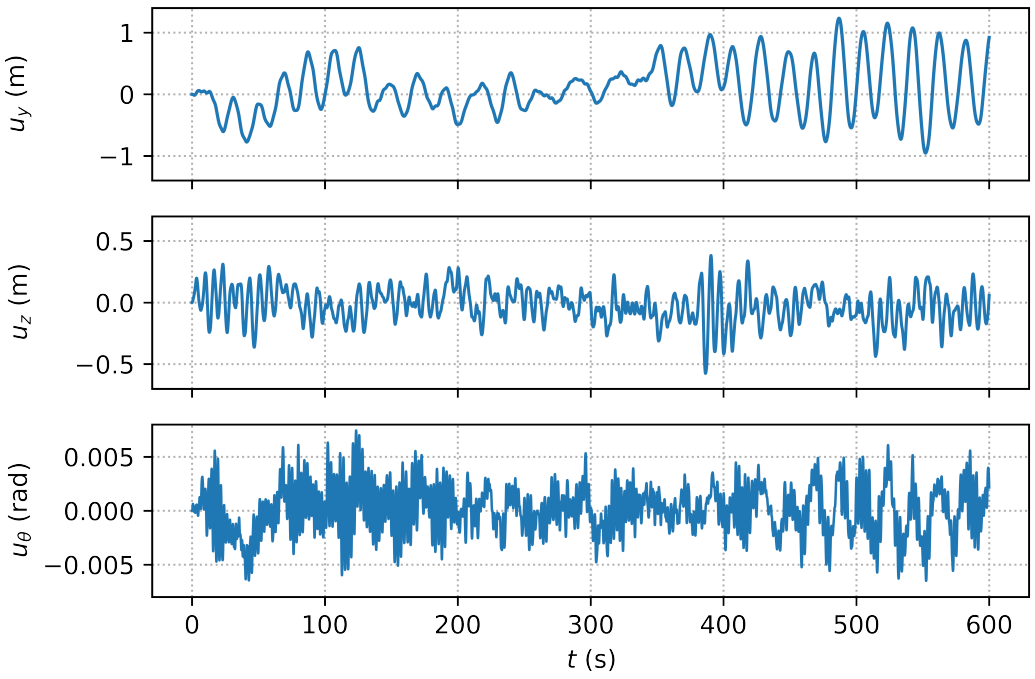


Figure 20: Midspan response from one time series simulation at 30 m/s mean wind velocity

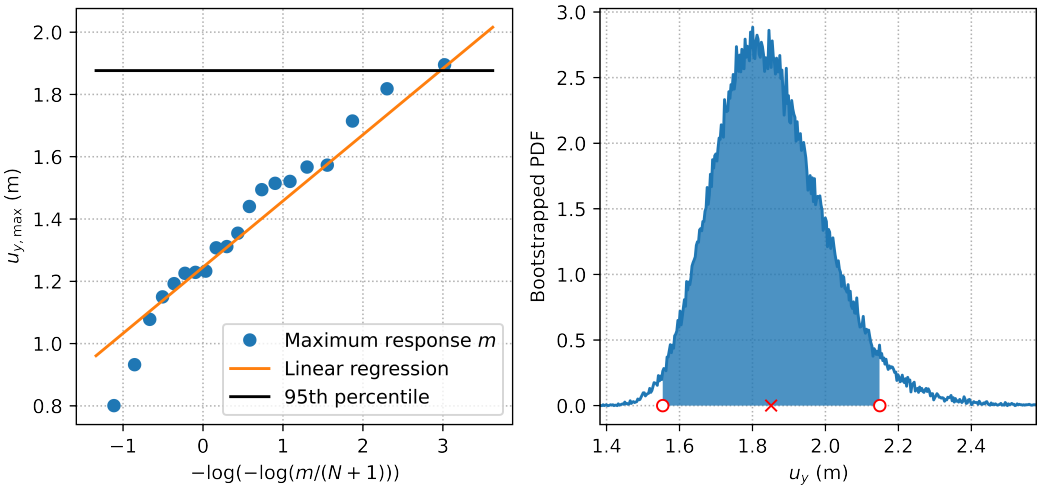


Figure 21: Extreme value analysis of the horizontal midspan response at 30 m/s mean wind velocity. Left figure: the blue points are the maximum response from the simulations, the orange line describes the Gumbel parameters (see Eq. (2.71)) and the black line denote the 95th percentile. Right figure: the solid blue line denote the bootstrapped PDF while the red cross denote the 95th percentile and the red circles and the shaded area marks the 95 % confidence interval.

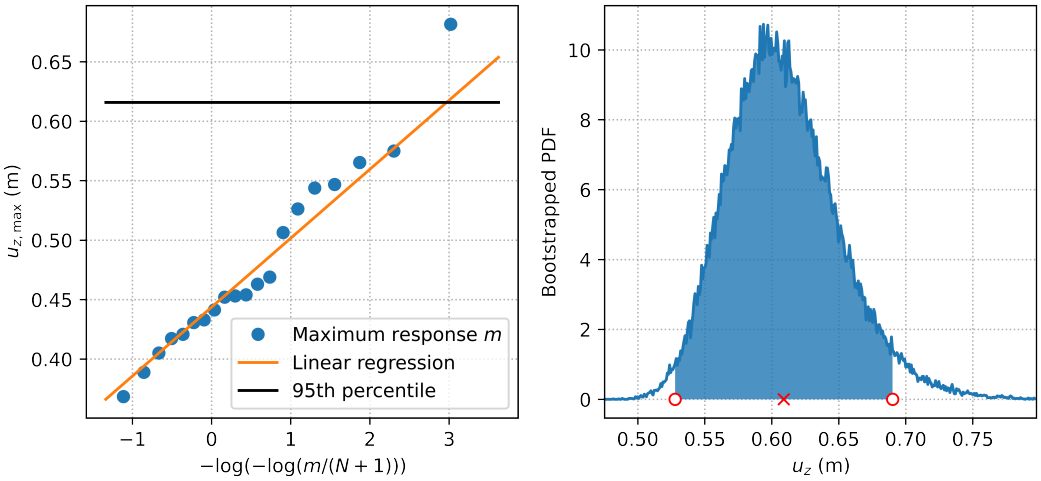


Figure 22: Extreme value analysis of the vertical midspan response at 30 m/s mean wind velocity. Left figure: the blue points are the maximum response from the simulations, the orange line describes the Gumbel parameters (see Eq. (2.71)) and the black line denote the 95th percentile. Right figure: the solid blue line denote the bootstrapped PDF while the red cross denote the 95th percentile and the red circles and the shaded area marks the 95 % confidence interval.

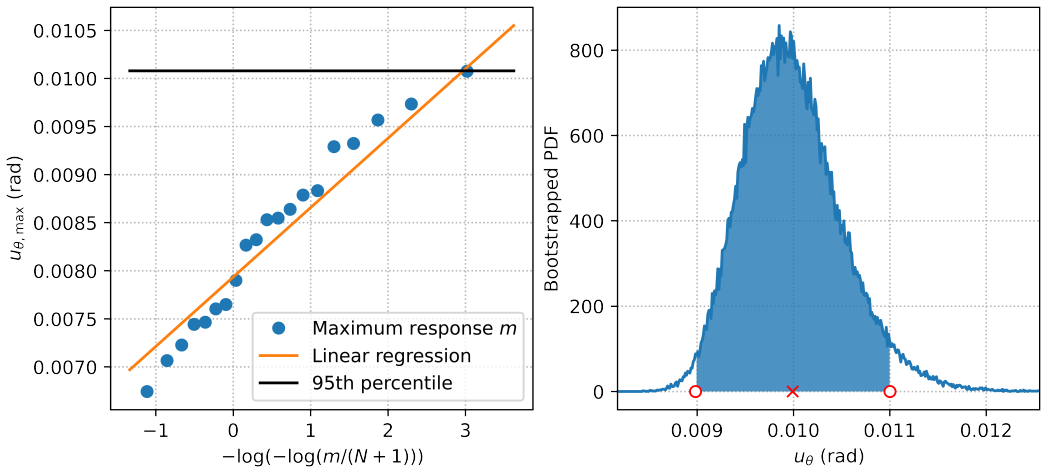


Figure 23: Extreme value analysis of the torsional midspan response at 30 m/s mean wind velocity. Left figure: the blue points are the maximum response from the simulations, the orange line describes the Gumbel parameters (see Eq. (2.71)) and the black line denote the 95th percentile. Right figure: the solid blue line denote the bootstrapped PDF while the red cross denote the 95th percentile and the red circles and the shaded area marks the 95 % confidence interval.

4.5.2 60 m/s

Figure 24 shows the horizontal, vertical and torsional response at 60 m/s mean wind velocity for one of the 20 simulated time series. The (absolute) maximum response from the figure is extracted, along with the other 19 simulated time series, and used to make the left-side figure in Figure 25, Figure 26 and Figure 27.

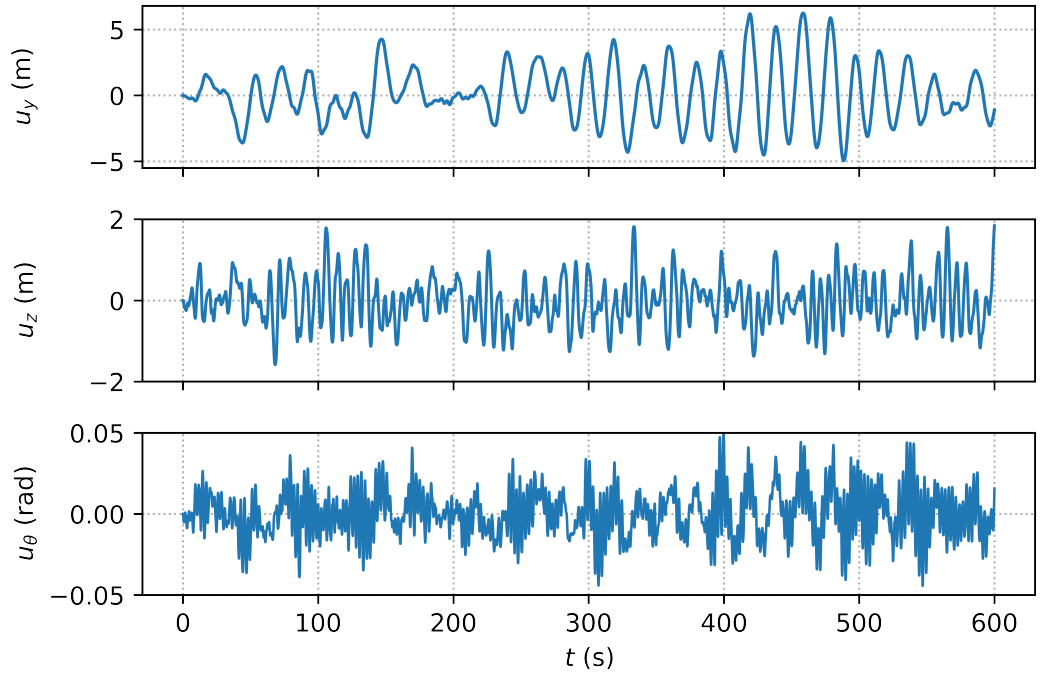


Figure 24: Midspan response from one time series simulation at 60 m/s mean wind velocity

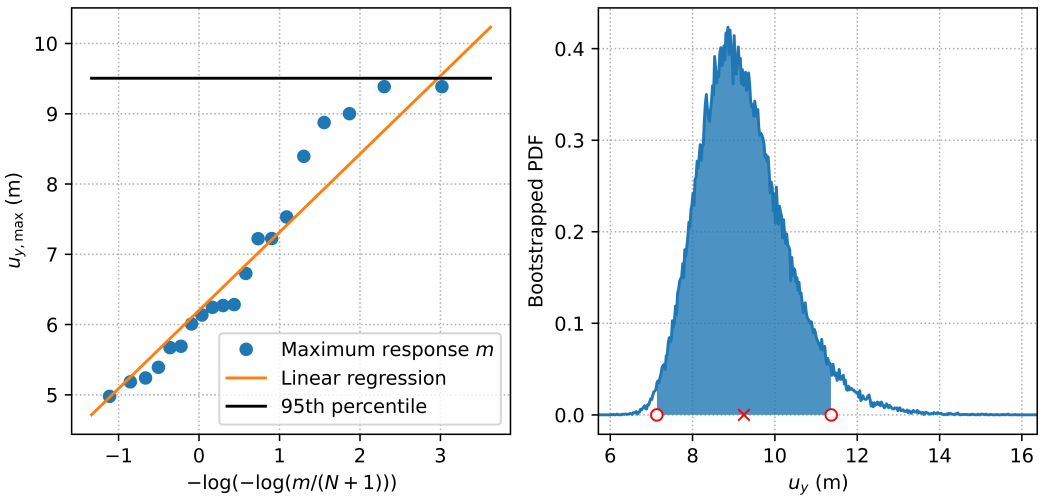


Figure 25: Extreme value analysis of the horizontal midspan response at 60 m/s mean wind velocity. Left figure: the blue points are the maximum response from the simulations, the orange line describes the Gumbel parameters (see Eq. (2.71)) and the black line denote the 95th percentile. Right figure: the solid blue line denote the bootstrapped PDF while the red cross denote the 95th percentile and the red circles and the shaded area marks the 95 % confidence interval.

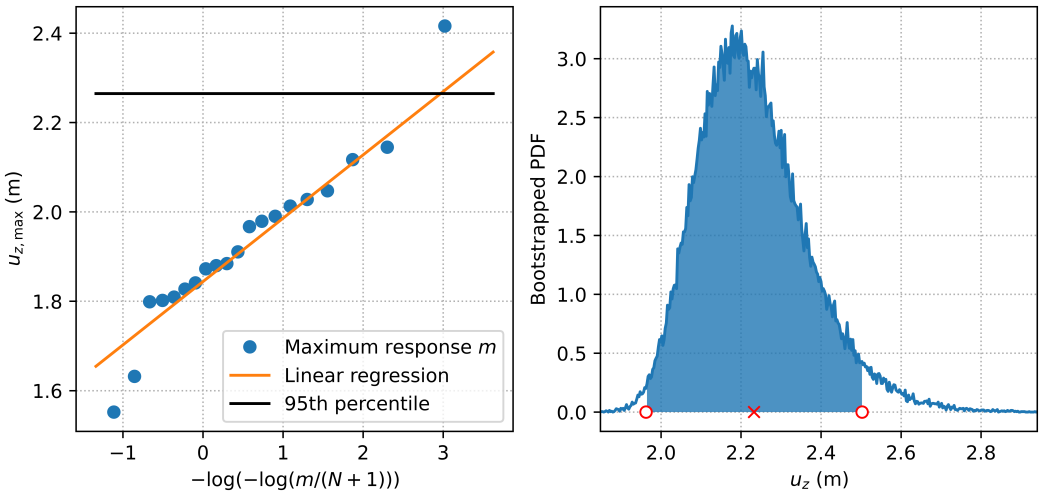


Figure 26: Extreme value analysis of the vertical midspan response at 60 m/s mean wind velocity. Left figure: the blue points are the maximum response from the simulations, the orange line describes the Gumbel parameters (see Eq. (2.71)) and the black line denote the 95th percentile. Right figure: the solid blue line denote the bootstrapped PDF while the red cross denote the 95th percentile and the red circles and the shaded area marks the 95 % confidence interval.

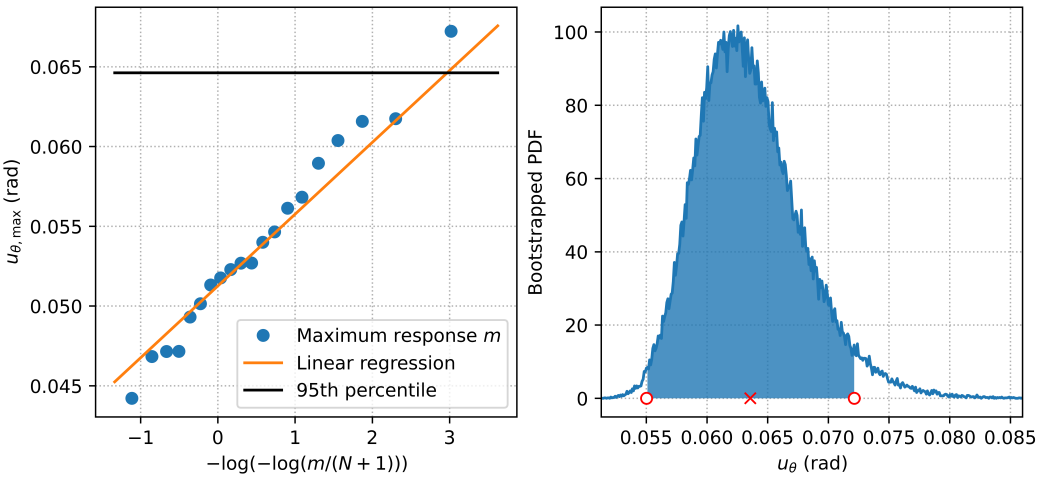


Figure 27: Extreme value analysis of the torsional midspan response at 60 m/s mean wind velocity. Left figure: the blue points are the maximum response from the simulations, the orange line describes the Gumbel parameters (see Eq. (2.71)) and the black line denote the 95th percentile. Right figure: the solid blue line denote the bootstrapped PDF while the red cross denote the 95th percentile and the red circles and the shaded area marks the 95 % confidence interval.

4.5.3 80 m/s

Figure 28 shows the horizontal, vertical and torsional response at 80 m/s mean wind velocity for one of the 20 simulated time series. The (absolute) maximum response from the figure is extracted, along with the other 19 simulated time series, and used to make the left-side figure in Figure 29, Figure 30 and Figure 31.

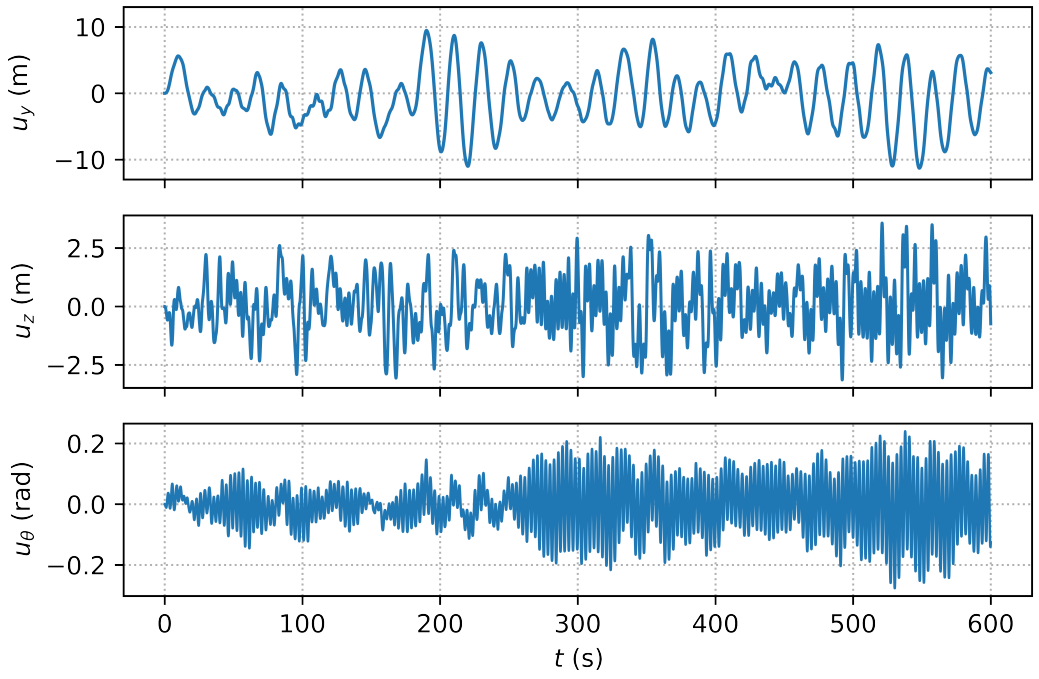


Figure 28: Midspan response from one time series simulation at 80 m/s mean wind velocity

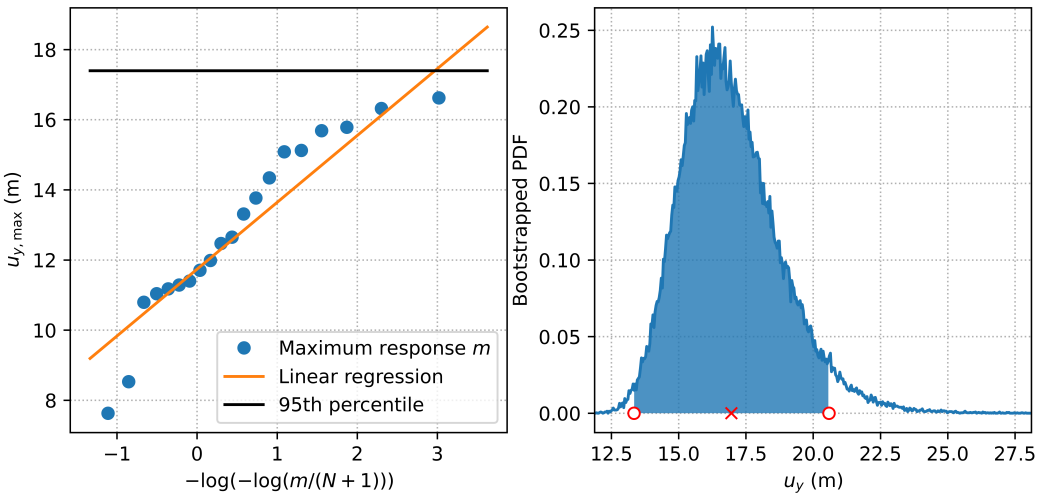


Figure 29: Extreme value analysis of the horizontal midspan response at 80 m/s mean wind velocity. Left figure: the blue points are the maximum response from the simulations, the orange line describes the Gumbel parameters (see Eq. (2.71)) and the black line denote the 95th percentile. Right figure: the solid blue line denote the bootstrapped PDF while the red cross denote the 95th percentile and the red circles and the shaded area marks the 95 % confidence interval.

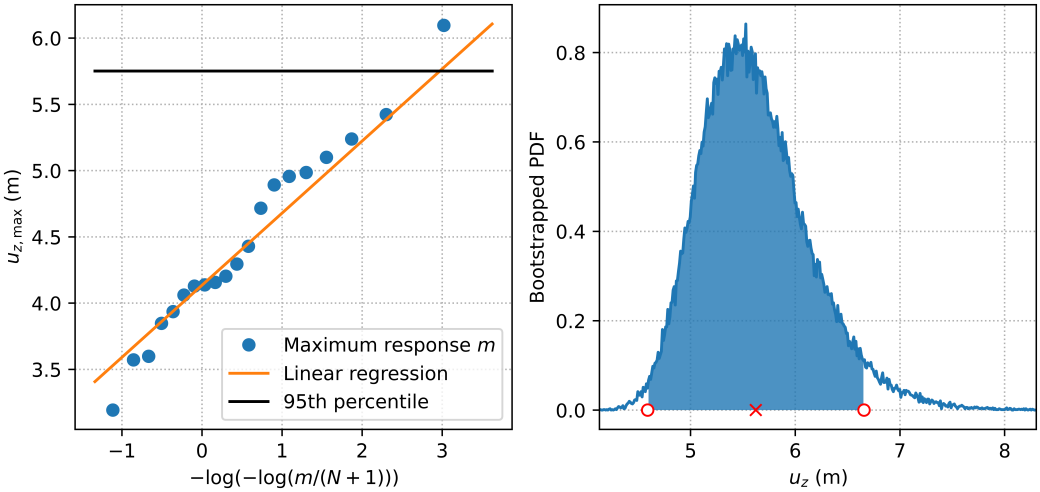


Figure 30: Extreme value analysis of the vertical midspan response at 80 m/s mean wind velocity. Left figure: the blue points are the maximum response from the simulations, the orange line describes the Gumbel parameters (see Eq. (2.71)) and the black line denote the 95th percentile. Right figure: the solid blue line denote the bootstrapped PDF while the red cross denote the 95th percentile and the red circles and the shaded area marks the 95 % confidence interval.

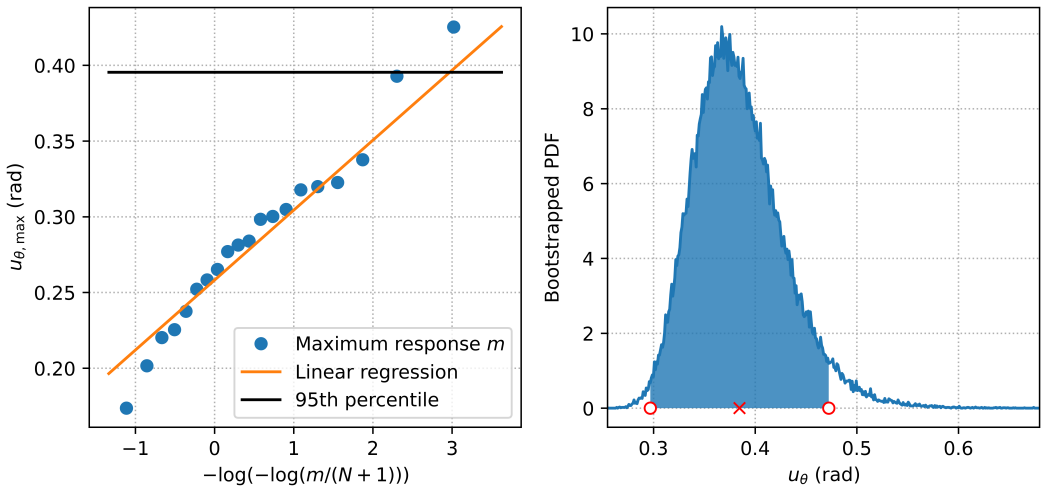


Figure 31: Extreme value analysis of the torsional midspan response at 80 m/s mean wind velocity. Left figure: the blue points are the maximum response from the simulations, the orange line describes the Gumbel parameters (see Eq. (2.71)) and the black line denote the 95th percentile. Right figure: the solid blue line denote the bootstrapped PDF while the red cross denote the 95th percentile and the red circles and the shaded area marks the 95 % confidence interval.

4.6 Extreme response frequency domain

This section presents the results from extreme value analysis of the frequency domain response.

Figure 32 show the CDF of the largest peak for a time interval of 10 minutes, the same length as the time series simulations, for the horizontal, vertical and torsional response. The red crosses denotes the 95th percentile peak response, which are presented in Table 8. Table 9 show the ratio of the 95th percentile peak response of the time and frequency domain results.

Table 8: Largest peak corresponding to the 95th percentile.

V (m/s)	R_y (m)	R_z (m)	R_θ (rad)
30	1.91	0.57	0.011
60	9.05	2.26	0.065
80	16.87	5.39	0.455

Table 9: Comparison of the time and frequency 95th percentile peak response.

V (m/s)	$\hat{R}_y/R_y$	$\hat{R}_z/R_z$	$\hat{R}_\theta/R_\theta$
30	0.98	1.09	0.91
60	1.05	1	1
80	1.03	1.07	0.87

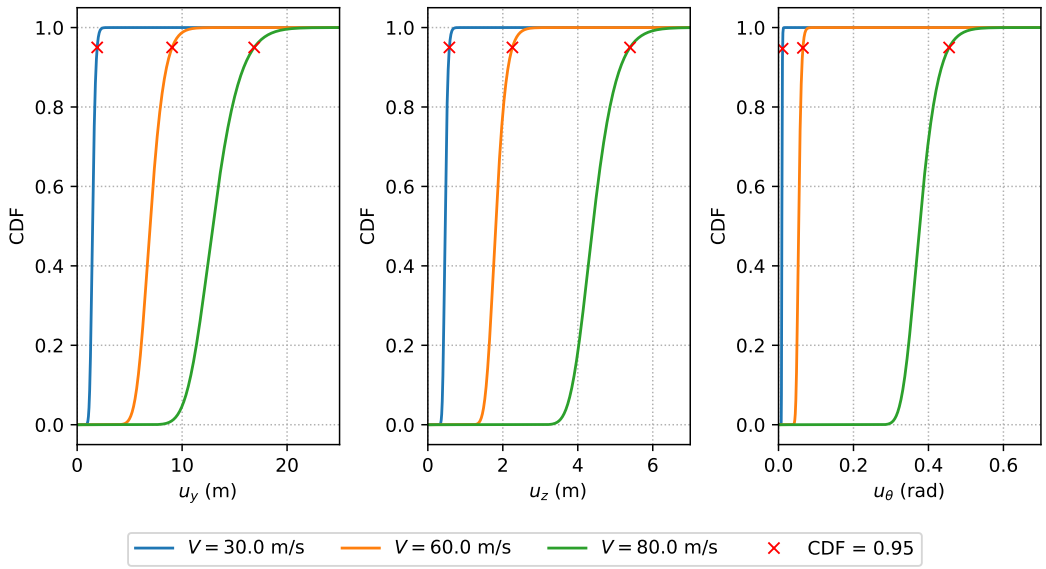


Figure 32: The CDF of the largest peak on an interval $T = 600$ s at three different mean wind velocities; 30, 60 and 80 m/s. Left figure: horizontal response. Middle figure: vertical response. Right figure: torsional response. The red crosses denote the 95th percentile.

5 Discussion

5.1 Wind tunnel data

Wind tunnel testing is necessary in order to obtain the aerodynamic derivatives and the load coefficients. Experimental testing always introduces some degree of uncertainty to the results. This section will discuss the results obtained from the wind tunnel testing.

5.1.1 Aerodynamic derivatives

Figure 12 and Figure 13 shows the aerodynamic derivatives associated with the aerodynamic damping and stiffness, respectively. The figures show the data obtained from the wind tunnel testing and the curve fits of the aerodynamic derivatives. The data is shown as black circles and are obtained from two tests with different mean wind velocities, namely 8 and 10 m/s, and a range of reduced frequencies. In the figures showing the aerodynamic derivatives P_1^* and A_4^*, the data points seem to diverge onto two different paths as a result of the two tests. For the derivatives P_4^*, P_5^*, H_1^*, H_2^*, H_4^*, H_5^*, A_1^*, A_2^*, A_5^* and A_6^* the data coincide, while for the remaining derivatives the data is more scattered.

The red curves in the figures show the curve fitting of the aerodynamic derivatives. For the most part, the curve fits are able to represent the data accurately, especially for the derivatives where the data from the two tests coincide. However, this is not the case for P_4^* where the data points are almost kept at a constant value while the curve fit does not seem to fit the data accurately. This is due to the fact that the coefficients describing the rational functions describes a pair of aerodynamic derivatives, as state in Section 3.3.1. This means that P_4^* and P_1^* share the same coefficients, namely a_1^{11}, a_2^{11} and a_4^{11} (values given in Table 3.1), and since the data points of P_1^* diverges onto two paths it affects the result of the curve fit of P_4^*.

As mentioned earlier it may be convenient to force the curve fits to appropriate quasi-static asymptotes in order to avoid the problems of the inverse Fourier transform. Since the Fourier transform uses the entire frequency range, the curve fits need to be extrapolated outside the reduced frequency range. The extrapolation may yield unrealistic values, especially considering the fact that the data from the wind tunnel testing only ranges from a reduced circular frequency of 0.06 to 0.74. It might therefore be more appropriate to apply quasi-static asymptotes, especially for to case of P_4^*. However, quasi-static was not implemented. The limited amount of available data introduces uncertainty into curve fitting and the buffeting response calculations. The degree of uncertainty may have been reduced by obtaining more results from the wind tunnel testing of the aerodynamic derivatives.

5.1.2 Load coefficients

The load coefficients shown in Figure 14 shows a clear pattern. However, there is some scatter in regard to the force coefficients for each angle of attack. This scatter is most prominent in the figure showing the drag coefficient, especially around the zero angle of attack which have a single data point a relative large distance away from the rest of the points. This outlier might stem from an error in the measurements during the testing and was considered negligible when attaining the load coefficients used in calculating the

buffeting response.

5.1.3 Verification of the rational functions

In addition to the visual confirmation of the accuracy of the curve fitted functions of the aerodynamic derivatives, the motion induced forces can be simulated by the use of the state space model presented in Section 3.4 and compared to the self-excited forces calculated through Eq. (2.12). The comparison of the self-excited forces are shown in Figure 15. In the top right figure, there is a clear discrepancy between the two force histories. There are also discrepancies in the two other sub figures containing the force histories, but they are more subtle. These discrepancies come from the fact that the state space model assume that the motion begin at $t = 0$ with the initial conditions shown in the figure while the motion does not start at $t = 0$ when a single harmonic motion is assumed in Eq. (2.12). The beginning of the force histories are therefore not interesting when looking at how well the rational functions are able to describe the motion induced forces. It is therefore concluded that the rational functions are able to describe the load history with high accuracy, at least within the frequency range the wind tunnel data inhabits.

5.2 Buffeting response

The comparison of the statistical properties of the buffeting response in time and frequency domain shown in Figure 16 (and quantified in Table 5 and Table 6) show that the frequency and time response coincide fairly well. However, the frequency response (solid lines) display fluctuations, which was not expected. This is most likely an error in the implementation of the frequency response calculations. This is supported by the fact that the frequency domain analysis was a lot more computationally expensive than the time domain analysis. In theory, the time domain analysis is more computationally expensive than the frequency domain analysis. However, this was not the case here, as the time domain analysis required much less time than the frequency domain analysis to complete the calculations. the buffeting response was significantly faster to obtain in the time domain even considering the fact that multiprocessing was applied to the frequency response calculations to try to speed up the calculations.

It was expected similar results as the time and frequency domain calculations are effectively just different paths to the same destination. However, the time domain analysis is vexed by the need of performing time series simulations. This raises the question; how many simulations are are sufficient enough? Figure 16 obviously show that 20 response series yields a satisfactory result. Further investigations could have been made to reduced the number of simulations while still gaining a satisfactory result in order to save computational time. However, 20 simulations were used to reduce the uncertainty of the time domain response and the extreme value analysis.

5.3 Comparing power spectra

Figure 17 shows the comparison of the auto-spectral densities of the buffeting response obtained from frequency domain analysis and the auto-spectral density of the buffeting response obtained from employing Welch's method to the buffeting response in time domain. The figure shows the frequency ranging from 0 to 5 rad/s. However, the frequency range from the frequency domain analysis only ranges from 0 to 3 rad/s. This was due to

the troubles with the frequency response implementation mentioned in the previous section. This resulted in the Python script not being able to complete the task in a feasible amount of time.

Despite the difficulties in calculating the buffeting response, the most interesting rang of frequencies is covered and the frequency and time domain solution coincide in this range. The bottom figure in Figure 17 shows the graph of the auto-spectral density of the rotational response for the three different mean wind velocities V; 30, 60 and 80 m/s. There are two noticeable peaks for all three velocities. The first one lies around the frequency of the first horizontal mode shape (see Table 1) and the peaks are noticeable around the same frequency in the figures showing the auto-spectral density of the horizontal response. The peaks in the spectra of the rotational response stem from the coupling of the horizontal and torsional modes. The second peak along the graph, where $V = 30$ m/s, correspond to the first torsional mode. This peak is not as distinctive as the first peak, but it can be seen in all three figures. As the mean wind velocity increases, the peaks shift towards the left and are getting higher and narrower. At $V = 80$ m/s, just below the stability limit, the peaks are very distinctive in all three auto spectra due to mode coupling. This result indicate that the unsteady dynamic response of the bridge is coupled horizontally, vertically and torsionally. This was not expected based on the theory presented in Section 2.4 which stated that the flutter stability is due to the vertical or torsional response, or a combination of the two. Figure 18 was based on this theory and tried to find an approximation of the stability limit based on only one vertical and one torsional mode. However, from what is seen in Figure 17 a better approximation could have been found by including one horizontal mode.

5.4 Stability

Figure 18 shows the results obtained from the eigenvalue analysis, as described in Section 3.6, for the first torsional mode ($\boldsymbol{\phi}_{16}$) and the third vertical mode ($\boldsymbol{\phi}_5$, see Figure 7). In the bottom figure, the damping ratio is shown as a function of the mean wind velocity and the top figure shows the imaginary part of the systems eigenvalues, or the natural frequency, as function of the mean wind velocity. The damping ratio seem to increase for both modes as the mean velocity increases. However, around $V = 70$ m/s the damping ratio of $\boldsymbol{\phi}_{16}$ begins to decrease. It continues to decrease until the damping ratio reaches zero and the stability limit is reached. The natural frequencies of these two modes are also found from the eigenvalue analysis and is shown in the top figure. In the top figure, the natural frequency of the third vertical mode is observed to be approximately constant for all mean wind velocities. The natural frequency of the first torsional mode, however, is seen to decrease as the velocity increases. This coincide with the results observed in Figure 17. This shows that the natural frequency and damping ratio of a system is dependent on the mean wind velocity.

The results shown in Figure 18 are only calculates with the two modes that gives the lowest critical velocity for any pair of modes. The reason for only calculating the stability limit with only two modes was to get an approximation of the critical velocity, based on the flutter theory, and present a readable figure. However, as discussed in the previous section, a better approximation of the stability limit would have been obtained if it had included a horizontal mode because of the unstable dynamic response being a coupling of horizontal, vertical and torsional response. Never the less, the figure shows the essence of the stability calculations. The same figure could be made with the calculation of all 100 mode shapes, but there would be a struggle to extract any information from that figure.

A figure with all 100 mode shapes would also show the increases and decreases in both the natural frequencies and the damping ratios similar to what is seen in Figure 18. The real stability limit was calculated, as described in Section 3.6, and the critical velocity was determined to be 80.69 m/s.

Figure 19 show the consequence of the bridge being exposed to a mean wind velocity above the critical velocity. At the critical velocity, the midspan response is shown to be near constant over time, while for a velocity greater than the critical velocity, the free vibration response becomes unstable and keeps increasing over time. This is clearly visible in the rotational and vertical response, while the horizontal is beginning to show signs of instability towards the end. The instability is due to the fact that the system exhibits negative damping when exposed to mean wind velocities greater than the critical velocity. The negative damping can be illustrated in Figure 18 by imagining the continuation of the curves. After the stability limit is reached, $\boldsymbol{\phi}_{16}$ will continue into negative values and thus exhibit negative damping ratios.

The stability limit of the Hålogaland Bridge have previously been examined by Kvamstad [57] in 2011. Kvamstad found the critical mean wind velocity to be 68.1 m/s, which is a massive difference compared to the 80.69 m/s obtained in this thesis. Noticeable differences are that Kvamstad assumed the damping ratios to be 0.5 %, while in this thesis the damping ratio was assumed to equal to 1 % in all modes. A greater damping ratio will yield a greater critical velocity, but the difference in damping ratios does not justify the 12.59 m/s difference. The difference is most likely due to different aerodynamic derivatives. Kvamstad's measurements came from FORCE Technology's wind tunnel in Lyngby Denmark in 2010 while the wind tunnel data used in this thesis came from wind tunnel testing at NTNU over a decade later, performed by Solstad and Onstad. Kvamstad was unable to obtain the aerodynamic derivatives associated with horizontal motion as the section model in Denmark was restrained in the horizontal direction.

5.5 Extreme values

Given the results shown in Table 7 and Table 8 the extreme value analysis of the frequency and time domain buffeting response seem to coincide. The ratio of the 95th percentile extreme response of the time and frequency response is shown in Table 9. It is seen that the horizontal and vertical response at 60 m/s yields the exact same value in both domains, while the other responses vary slightly. The biggest discrepancy is found in torsional response. At 80 m/s the time domain result is 13 % less than the frequency domain result and at 30 m/s it is 9 % less. The horizontal response seem to be the most accurate with a maximum deviation of 5 % at 60 m/s.

In the figures showing the extreme responses obtained from the time domain simulations, the linear regression of the points does not coincide perfectly with the points, which would be expected if the extreme values followed a Gumbel distribution and sufficiently many realisations of the response were made. In Figure 22 and Figure 26 the maximum of all the extreme responses seem to be much greater than the rest of the extreme responses. In Figure 25 the points towards the end form a curved shape, and in Figure 26 and Figure 29 the minimum of the extreme responses seem to yield much lower values than the other points and the regression line. These deviations from a perfect line may be caused by too few simulations, or it could be that the assumed Gumbel distribution is not the true probability distribution of the extreme response. This could be investigated further by performing more and longer time domain simulations and by using the GEV distribution

instead.

6 Conclusion

In this thesis the wind induced dynamic response of the Hålogaland Bridge have been calculated both in time and frequency domain. This was done by using experimental data from wind tunnel testing of a down-scale section model of the Hålogaland Bridge to determine the aerodynamic properties of the Hålogaland Bridge. The experimental aerodynamic derivative data was used to curve fit rational functions to the transfer functions. The curve fits were then verified by using a state space model to simulating the motion induced forces and comparing it to the self-excited forces given in Eq. (2.12), at least within the range of the experimental data. The load coefficient were also successfully obtained from the wind tunnel data which showed clear and distinctive trends. Correlated time series simulations of the wind field were simulated using the Monte Carlo method and the Cholesky decomposition of the cross-spectral density of the turbulence components in order to decrease the computational costs. In theory, the time and frequency domain buffeting response should give the exact same answer if the time domain response have converged, i.e. sufficiently many simulations are performed. There were a total of 20 time series simulations with lengths of 600 s performed. While this did not yield the exact same answer in both domains it turned out to be approximately the same and it is concluded that 20 time series simulations with lengths of 600 s are sufficient enough to describe the response.

Extreme response analysis and flutter stability was also investigated as they are important aspects of bridge design. The extreme response analysis was conducted for the frequency and time response which also had a fairly high level of accuracy. The flutter stability yielded a somewhat unexpected result as it was expected that a good approximation of the stability limit could be obtained considering one vertical mode and one torsional mode. It was discovered from the results that a better approximation might have been made by adding a horizontal mode.

References

[1] O. Øiseth, A. Rönnquist and R. Sigbjörnsson, 'Finite element formulation of the self-excited forces for time-domain assessment of wind-induced dynamic response and flutter stability limit of cable-supported bridges', *Finite elements in analysis and design*, vol. 50, pp. 173–183, 2012.

[2] *Announcement of the Alan G. Davenport Wind Load Chain*, Accessed: 2023-05-18. [Online]. Available: http://www.iawe.org/about/Wind_Loading_Chain.pdf.

[3] N. Isyumov, 'Alan G. Davenport's mark on wind engineering', *Journal of Wind Engineering and Industrial Aerodynamics*, vol. 104, pp. 12–24, 2012.

[4] *The Alan G. Davenport Wind Loading Chain*, Accessed: 2023-05-18. [Online]. Available: https://www.eng.uwo.ca/media/news/2013/wind-loading-chain.html.

[5] E. Strømmen, *Theory of Bridge Aerodynamics*, 1st ed. Springer, 2006.

[6] W. R. Sears, 'Some aspects of non-stationary airfoil theory and its practical application', *Journal of the Aeronautical Sciences*, vol. 8, no. 3, pp. 104–108, 1941.

[7] H. W. Liepmann, 'On the application of statistical concepts to the buffeting problem', *Journal of the Aeronautical Sciences*, vol. 19, no. 12, pp. 793–800, 1952.

[8] A. G. Davenport, 'The spectrum of horizontal gustiness near the ground in high winds', *Quarterly Journal of the Royal Meteorological Society*, vol. 87, no. 372, pp. 194–211, 1961.

[9] J. C. Kaimal, J. Wyngaard, Y. Izumi and O. Coté, 'Spectral characteristics of surface-layer turbulence', *Quarterly Journal of the Royal Meteorological Society*, vol. 98, no. 417, pp. 563–589, 1972.

[10] G. Solari, 'Gust buffeting. i: Peak wind velocity and equivalent pressure', *Journal of Structural Engineering*, vol. 119, no. 2, pp. 365–382, 1993.

[11] T. Von Karman, 'Progress in the statistical theory of turbulence', *Proceedings of the National Academy of Sciences*, vol. 34, no. 11, pp. 530–539, 1948.

[12] Statens Vegvesen, *N400 bruprosjektering*, 2015. [Online]. Available: https://www.vegvesen.no/fag/publikasjoner/handboker/vegnormalene/n400/.

[13] R. H. Scanlan and J. J. Tomko, 'Airfoil and bridge deck flutter derivatives', *Journal of the Engineering Mechanics Division*, vol. 97, pp. 1717–1737, 1971.

[14] O. Øiseth, A. Rönnquist and R. Sigbjörnsson, 'Simplified prediction of wind-induced response and stability limit of slender long-span suspension bridges, based on modified quasi-steady theory: A case study', *Journal of wind engineering and industrial aerodynamics*, vol. 98, no. 12, pp. 730–741, 2010.

[15] D. E. Newland, *An introduction to random vibrations, spectral & wavelet analysis*. Courier Corporation, 2012.

[16] X. Chen, M. Matsumoto and A. Kareem, 'Aerodynamic coupling effects on flutter and buffeting of bridges', *Journal of Engineering Mechanics*, vol. 126, no. 1, pp. 17–26, 2000.

[17] E. Simiu and R. H. Scanlan, *Wind effects on structures: fundamentals and applications to design*. John Wiley New York, 1996, vol. 688.

[18] F. Tubino and G. Solari, 'Gust buffeting of long span bridges: Double modal transformation and effective turbulence', *Engineering structures*, vol. 29, no. 8, pp. 1698–1707, 2007.

[19] O. Øiseth, A. Rönnquist and R. Sigbjörnsson, 'Effects of co-spectral densities of atmospheric turbulence on the dynamic response of cable-supported bridges: A case study', *Journal of Wind Engineering and Industrial Aerodynamics*, vol. 116, pp. 83–93, 2013.

[20] X. Chen and A. Kareem, 'Advanced analysis of coupled buffeting response of bridges: A complex modal decomposition approach', *Probabilistic engineering mechanics*, vol. 17, no. 2, pp. 201–213, 2002.

[21] Y. Xu, O. Øiseth and T. Moan, 'Time domain simulations of wind-and wave-induced load effects on a three-span suspension bridge with two floating pylons', *Marine Structures*, vol. 58, pp. 434–452, 2018.

[22] E. Kreyszig, *Advanced Engineering Mathematics*, 10th ed. Wiley, 2011.

[23] C. G. Bucher and Y. K. Lin, 'Stochastic stability of bridges considering coupled modes', *Journal of Engineering Mechanics*, vol. 114, no. 12, pp. 2055–2071, 1988.

[24] O. Øiseth, A. Rönnquist and R. Sigbjörnsson, 'Time domain modeling of self-excited aerodynamic forces for cable-supported bridges: A comparative study', *Computers & structures*, vol. 89, no. 13-14, pp. 1306–1322, 2011.

[25] O. Øiseth, A. Rönnquist, A. Naess and R. Sigbjörnsson, 'Estimation of extreme response of floating bridges by monte carlo simulation', in *Proceedings of the 9th International Conference on Structural Dynamics, EURODYN*, 2014, pp. 2905–2912.

[26] C. Borri, C. Costa and W. Zahlten, 'Non-stationary flow forces for the numerical simulation of aeroelastic instability of bridge decks', *Computers & structures*, vol. 80, no. 12, pp. 1071–1079, 2002.

[27] X. Chen, M. Matsumoto and A. Kareem, 'Time domain flutter and buffeting response analysis of bridges', *Journal of Engineering Mechanics*, vol. 126, no. 1, pp. 7–16, 2000.

[28] O. Øiseth, 'Contribution to IABSE WG 10 Super-long Span Bridge Aerodynamics Step 1-1a', Norwegian University of Science and Technology, 2017.

[29] W. L. Brogan, *Modern Control Theory*, 3rd ed. Prentice Hall, 1991.

[30] M. Shinozuka, 'Monte carlo solution of structural dynamics', *Computers & Structures*, vol. 2, no. 5-6, pp. 855–874, 1972.

[31] M. Shinozuka and C.-M. Jan, 'Digital simulation of random processes and its applications', *Journal of sound and vibration*, vol. 25, no. 1, pp. 111–128, 1972.

[32] K. Aas-Jakobsen and E. Strømmen, 'Time domain buffeting response calculations of slender structures', *Journal of Wind Engineering and Industrial Aerodynamics*, vol. 89, no. 5, pp. 341–364, 2001.

[33] C. Benoit, 'Note sur une méthode de résolution des équations normales provenant de l'application de la méthode des moindres carrésa un systeme d'équations linéaires en nombre inférieura celui des inconnues. application de la méthodea la résolution d'un systeme défini d'équations linéaires (procédé du commandant cholesky)', *Bulletin géodésique*, vol. 2, no. 1, pp. 67–77, 1924.

[34] G. Deodatis, 'Simulation of ergodic multivariate stochastic processes', *Journal of engineering mechanics*, vol. 122, no. 8, pp. 778–787, 1996.

[35] M. Di Paola, 'Digital simulation of wind field velocity', *Journal of Wind Engineering and Industrial Aerodynamics*, vol. 74, pp. 91–109, 1998.

[36] Q. Ding, L. Zhu and H. Xiang, 'Simulation of stationary gaussian stochastic wind velocity field', *Wind & structures*, vol. 9, no. 3, pp. 231–243, 2006.

[37] L. Carassale and G. Solari, 'Monte carlo simulation of wind velocity fields on complex structures', *Journal of Wind Engineering and Industrial Aerodynamics*, vol. 94, no. 5, pp. 323–339, 2006.

[38] H. Katsuchi, N. Jones, R. Scanlan and H. Akiyama, 'Multi-mode flutter and buffeting analysis of the akashi-kaikyo bridge', *Journal of Wind Engineering and Industrial Aerodynamics*, vol. 77, pp. 431–441, 1998.

[39] T. Agar, 'Aerodynamic flutter analysis of suspension bridges by a modal technique', *Engineering structures*, vol. 11, no. 2, pp. 75–82, 1989.

[40] J. Jakobsen and E. Hjorth-Hansen, 'Arne selberg's formula for flutter speed in light of multimodal flutter analysis', in *Twelfth International Conference on Wind Engineering, Cairns, Australia*, 2007.

[41] A. Naess, *Applied Extreme Value Statistics Including the ACER method*, 2022.

[42] E. J. Gumbel, 'The return period of flood flows', *The annals of mathematical statistics*, vol. 12, no. 2, pp. 163–190, 1941.

[43] M. Fréchet, 'Sur la loi de probabilité de l'écart maximum', *Ann. Soc. Math. Polon.*, vol. 6, pp. 93–116, 1927.

[44] W. Weibull, 'A statistical distribution function of wide applicability', *Journal of applied mechanics*, 1951.

[45] K. V. Bury *et al.*, *Statistical models in applied science*. Wiley, 1975.

[46] A. Naess and T. Moan, *Stochastic dynamics of marine structures*. Cambridge University Press, 2013.

[47] H. O. Madsen, S. Krenk and N. C. Lind, *Methods of structural safety*. Courier Corporation, 2006.

[48] B. Efron and R. J. Tibshirani, *An introduction to the bootstrap*. CRC press, 1994.

[49] A. C. Davison and D. V. Hinkley, *Bootstrap methods and their application*. Cambridge university press, 1997.

[50] Statens Vegvesen, accessed: 2023-05-22. [Online]. Available: https://www.vegvesen.no/vegprosjekter/europaveg/e6halogalandsbrua/nyhetsarkiv/halogalandsbrua-far-internasjonal-anerkjennelse/.

[51] Statens Vegvesen. 'Techincal data'. (), [Online]. Available: https://www.vegvesen.no/vegprosjekter/europaveg/e6halogalandsbrua/english/technical-data/ (visited on 5th May 2023).

[52] M. Smith, *ABAQUS/Standard User's Manual, Version 6.9*, English. United States: Dassault Systèmes Simulia Corp, 2009.

[53] 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', M.S. thesis, NTNU, 2022.

[54] C. R. Harris, K. J. Millman, S. J. van der Walt *et al.*, 'Array programming with NumPy', *Nature*, vol. 585, no. 7825, pp. 357–362, Sep. 2020. DOI: 10.1038/s41586-020-2649-2. [Online]. Available: https://doi.org/10.1038/s41586-020-2649-2.

[55] P. Virtanen, R. Gommers, T. E. Oliphant *et al.*, 'SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python', *Nature Methods*, vol. 17, pp. 261–272, 2020. DOI: 10.1038/s41592-019-0686-2.

[56] P. Welch, 'The use of fast fourier transform for the estimation of power spectra: A method based on time averaging over short, modified periodograms', *IEEE Transactions on audio and electroacoustics*, vol. 15, no. 2, pp. 70–73, 1967.

[57] H. Kvamstad, 'Assessment of the flutter stability limit of the hålogaland bridge using a probabilistic approach [master's thesis]', *Trondheim: Norwegian University of Science and Technology*, 2011.

Appendix

A Python sripts

The following Python scripts are written to carry out the calculations of this thesis. The scripts are (tried to be) written with readability in mind. Therefore, it is possible to improve the code in order to optimise the computational cost. Generally, the scripts uses the output from other files as input and some scripts have so-called 'magic numbers' which should be changed, e.g. depending on which response is wanted. Therefore, any use of the scripts should be done with great care.

A.1 functions.py

```
import numpy as np
import scipy.linalg as spla
from scipy import interpolate

def distance_matrix(l, npoints=2):
    """Creates the distance matrix dydy needed for obtaining cross
    ↪ spectral density
    Parameters:
        l - (float) distance between the first and last point
        npoints - (int) number of points on the distance l

    returns:
        dydy (2D-array) a matrix with the absolute distance to each point
    """
    y = np.linspace(0,l,npoints)
    return np.abs(np.array([y]) - np.array([y]).T)

def MCCholesky(frequencyaxis, SS, Nsim=1, dwsim=.001):
    """ Creates random realisation of the turbulence components by Monte
    ↪ Carlo simulation,
        Cholesky decomposition and interpolation of decomposed matrices

    Args:
        frequencyaxis (array): frequency axis
        SS (ndarray): cross spectral matrix
        Nsim (int, optional): Number of simulations. Defaults to 1.
        dwsim (float, optional): smaller frequency step for simulation.
↪ Defaults to .001.

    Returns:
        array, ndarray: returns the time array and all Nsim simulations
    """
    omegaaxis = frequencyaxis
    GG = np.zeros_like(SS)

```

```
    for k in range(len(omegaaxis)):
        if np.max(np.abs(SS[:, :, k]))<1e-10:
            continue
        else:
            GG[:, :, k] = spla.cholesky(SS[:, :, k], lower=True)

    wsim = np.arange(dwsim, omegaaxis[-1], dwsim) # omegaaxissim
    NFFT = int(2**np.ceil(np.log2(len(wsim))))
    t = np.linspace(0, 2*np.pi/dwsim, NFFT)
    if Nsim==1:
        x = np.zeros((GG.shape[0], len(t)))
    else:
        x = np.zeros((Nsim, GG.shape[0], len(t)))

    for z in range(Nsim):
        phi = 2*np.pi* np.random.rand(GG.shape[0], len(wsim)) #Random
        ↪  phase angle between 0 and 2pi
        for m in range(GG.shape[0]):
            for n in range(0, m+1):
                c = interpolate.interp1d(omegaaxis, np.abs(GG[m, n,
                ↪  :]))(wsim) * np.exp(1j *phi[n, :]) * np.sqrt(2*dwsim)
                x[z, m, :] += np.real(np.fft.ifft(c, n=NFFT)* NFFT)
        # print("{} simulation(s) complete".format(z+1))
    return t, x

def gumbel_beta(std):
    """Find the beta parameter in the gumbel distribtion

    Args:
        std (float): standard deviation

    Returns:
        float: beta
    """
    return np.sqrt(6/np.pi**2 * std**2)

def gumbel_mu(mean, std):
    """Find the mu parameter in the gumbel distribution

    Args:
        mean (float): mean of the distribution
        std (float): standard deviation

    Returns:
        float: mu
    """
    return mean - gumbel_beta(std)*np.euler_gamma

def gumbel_CDF(x, mean, std):
    """gumbel cumulative distribution function

```

```
82      Args:
83          x (array): axis
84          mean (float): mean of the distribution
85          std (float): standard deviation
86
87      Returns:
88          array: gumbel CDF
89      """
90      beta = gumbel_beta(std)
91      mu = gumbel_mu(mean, std)
92
93      return np.exp(-np.exp(-(x-mu)/beta))
94
95  def gumbel_PDF(x, mean, std):
96      """gumbel probability distribution function
97
98      Args:
99          x (array): axis
100         mean (float): mean of the distrivution
101         std (float): standard deviation
102
103     Returns:
104         array: gumbel PDF
105     """
106     beta = gumbel_beta(std)
107     mu = gumbel_mu(mean, std)
108
109     return np.exp(-np.exp(-(x-mu)/beta)) * np.exp(-(x-mu)/beta) / beta
110
111 def gumbel_PDF2(x, mu, beta):
112     return np.exp(-np.exp(-(x-mu)/beta)) * np.exp(-(x-mu)/beta) / beta
113
114 def aerodynamic_derivatives(vred, RFa, d):
115     """ Calculates the ADs from the rational functions and state
116         parameters for one spesific reduced velocity
117
118     Args:
119         vred (float): reduced velocity
120         RFa (ndarray): rational functions
121         d (array): state parameters
122
123     Returns:
124         (2, 3, 3) array: first (3, 3) array contains the stiffness related
    ↪ ADs, second the damping
125     """
126
127     AD = np.zeros((2, 3, 3))
128
129     sumRFa = np.zeros((3, 3), dtype=complex)
130
131     for i in range(RFa.shape[0]):
```

```
132         if i == 0:
133             sumRFa += RFa[i]
134         elif i == 1:
135             sumRFa += RFa[i]* 1j/vred
136         else:
137             sumRFa += RFa[i]* (1j/vred)/((1j/vred) + d[i-2])
138 
139     sumRFa *= vred**2
140 
141     AD[0] = np.real(sumRFa)
142     AD[1] = np.imag(sumRFa)
143 
144     return AD
145 
146 def wind_tunnel_measurments(omegaaxis, rho, B, V, RFa, d):
147     """ Creates the aerodynamic stiffness and damping matrices for
148         an array of frequencies and one wind velocity V by using the
149         rational funcition that is found through the experimental wind
150         tunnel testing
151 
152     Args:
153         omega (array): frequency axis
154         rho (float): air density
155         B (float): section width
156         V (float): mean wind velocity
157         RFa (ndarray): rational functions
158         d (array): state parameters
159 
160     Returns:
161         ndarray, ndarray: aerodynamic stiffness and damping matrices
162     """
163     cae = np.zeros((3, 3, len(omegaaxis)))
164     kae = np.zeros((3, 3, len(omegaaxis)))
165 
166     vred = V/(B*omegaaxis)
167     for k in range(len(omegaaxis)):
168         AD = aerodynamic_derivatives(vred[k], RFa, d)
169         for i in range(AD.shape[0]):
170             AD[i, -1, :] *= B
171             AD[i, :, -1] *= B
172         kae[:, :, k] = 1/2*rho*B**2*omegaaxis[k]**2 *AD[0] # Eq.(4) Øiseth
            ↪ et al.
173         cae[:, :, k] = 1/2*rho*B**2*omegaaxis[k] *AD[1] # Eq.(4) Øiseth et
            ↪ al.
174 
175     return kae, cae
176 
177 def modal_aerodynamic_damping_and_stiffness(V, omegaaxis, phiphi, rho, B,
    ↪ RFa, d, beam):
178     """ Calculates the modal aerodynamic damping and stiffness matrix
179         for a given mean wind velocity V
```

```
180
181     Args:
182         V (float): mean wind velocity
183         omegaaxis (array): frequency axis
184         phiphi (ndarray): mode shapes
185         rho (float): air density
186         B (float): section width
187         RFa (ndarray): rational functions
188         d (array): state parameters
189
190     Returns:
191         ndarray, ndarray: aerodynamic stiffness and damping matrices
192     """
193     kae, cae = wind_tunnel_measurments(omegaaxis, rho, B, V, RFa, d)
194
195     KKae = np.zeros((phiphi.shape[0], phiphi.shape[0], len(omegaaxis)))
196     CCae = np.zeros((phiphi.shape[0], phiphi.shape[0], len(omegaaxis)))
197
198     phiphi_p = np.transpose(phiphi, [1, 0, 2])
199
200     for k in range(len(omegaaxis)):
201         integrandKK = np.zeros((phiphi.shape[0], phiphi.shape[0],
        ↪  len(beam)))
202         integrandCC = np.zeros((phiphi.shape[0], phiphi.shape[0],
        ↪  len(beam)))
203         for m in range(len(beam)):
204             integrandKK[:, :, m] = phiphi_p[:, :, m].T @ kae[:, :, k] @
            ↪  phiphi_p[:, :, m]
205             integrandCC[:, :, m] = phiphi_p[:, :, m].T @ cae[:, :, k] @
            ↪  phiphi_p[:, :, m]
206
207         KKae[:, :, k] = np.trapz(integrandKK, beam, axis=-1)
208         CCae[:, :, k] = np.trapz(integrandCC, beam, axis=-1)
209
210     return KKae, CCae
211
212 def frequency_response_matrix(MM, CC, KK, V, omegaaxis, phiphi, rho, B,
↪  RFa, d, beam):
213     """ Calculate the frequency response matrix for a system
214         (with self-excited forces) for one mean wind velcoity V
215
216     Args:
217         MM (ndarray): mass matrix
218         CC (ndarray): damping matrix
219         KK (ndarray): stiffness matrix
220         V (float): mean wind velocity
221         omegaaxis (array): frequency axis
222         phiphi (ndarray): mode shapes
223         rho (float): air density
224         B (float): section width
225         RFa (ndarray): rational functions
```

```
226         d (array): state parameters
227 
228     Returns:
229         HH: Frequency response matrix for one given wind velocity V
230     """
231     KKae, CCae = modal_aerodynamic_damping_and_stiffness(V, omegaaxis,
        ↪  phiphi, rho, B, RFa, d, beam) # Eq.(6) Øiseth et al.
232     HH = np.zeros((MM.shape[0], MM.shape[0], len(omegaaxis)),
        ↪  dtype=complex)
233 
234     for k in range(len(omegaaxis)):
235         HH[:, :, k] = np.linalg.inv(-omegaaxis[k]**2*MM +
            ↪  1j*omegaaxis[k]*(CC-CCae[:, :, k]) + (KK-KKae[:, :, k])) #
            ↪  Eq.()9 Øiseth et al.
236     return HH
237 
238 def cross_spectral_wind_field(V, omegaaxis, dxdx):
239     """ Calculate the cross-spectral densities of the turbulence
240         for one mean wind velocity V and one frequency omega
241     Args:
242         V (float): mean wind velocity
243         omega (float): frequency
244         dxdx (ndarray): distance matrix
245 
246     Returns:
247         SvSv: CSD turbulence
248     """
249     L1 = 100.0 # Reference integral length scale
250     z1 = 10.0 # Reference height
251     z = 50.0 # Height above ground
252     xLu = L1*(z/z1)**0.3 # Integral length scale
253     xLw = 1/12*xLu # Integral length scale
254     Au, Aw = 6.8/2/np.pi, 9.4/2/np.pi # Constant in the auto-spectral
        ↪  density
255     Iu = 0.15 # Turbulence intensity
256     Iw = 1/4*Iu# Turbulence intensity
257     Cuy, Cwy = 10.0, 6.5 # Decay coeff
258 
259     SvSv = np.zeros((2, 2, dxdx.shape[0], dxdx.shape[0], len(omegaaxis)))
260     Su = (Iu*V)**2*Au*xLu/V/((1+1.5*Au*omegaaxis*xLu/V)**(5.0/3.0))
261     Sw = (Iw*V)**2*Aw*xLw/V/((1+1.5*Aw*omegaaxis*xLw/V)**(5.0/3.0))
262     for ii in range(len(omegaaxis)):
263         Cou = np.exp(-Cuy/2/np.pi*dxdx*omegaaxis[ii]/V)
264         Cow = np.exp(-Cwy/2/np.pi*dxdx*omegaaxis[ii]/V)
265         SvSv[0, 0, :, :, ii] = Su[ii]*Cou
266         SvSv[1, 1, :, :, ii] = Sw[ii]*Cow
267     SvSv[0, 1, :, :, :] = 0
268     SvSv[1, 0, :, :, :] = 0
269     return SvSv
270 
271 def cross_spectral_load_matrix(omegaaxis, V, BqBq, dxdx, phiphi):
```

```
    """ Calculate the cross spectral density load matrix for a
        given mean wind velocity V

    Args:
        omegaaxis (array): frequency axis
        V (float): mean wind velocity
        BqBq (ndarray): buffeting load matrix
        dxdx (ndarray): distance matrix
        phiphi (ndarray): mode shapes

    Returns:
        SQSQ: CSD load
    """
    phiphi_p = np.transpose(phiphi, [1, 0, 2])
    omegaaxis_load = np.logspace(np.log10(np.min(omegaaxis)),
    ↪  np.log10(np.max(omegaaxis)), round(len(omegaaxis)/4))
    SQSQ_coarse = np.zeros((phiphi.shape[0], phiphi.shape[0],
    ↪  len(omegaaxis_load)))

    SvSv = cross_spectral_wind_field(V, omegaaxis, dxdx)
    for k in range(len(omegaaxis_load)):
        integrand = np.zeros((phiphi.shape[0], phiphi.shape[0],
        ↪  dxdx.shape[0], dxdx.shape[0]))
        for m in range(dxdx.shape[0]):
            for n in range(dxdx.shape[0]):
                integrand[:, :, m, n] = phiphi_p[:, :, m].T @ BqBq @
                ↪  SvSv[:, :, m, n, k] @ np.conj(BqBq).T @ phiphi_p[:, :,
                ↪  n]
        SQSQ_coarse[:, :, k] = np.trapz(np.trapz(integrand, dxdx[0],
        ↪  axis=-1), dxdx[0], axis=-1)

    SQSQ = np.zeros((phiphi.shape[0], phiphi.shape[0], len(omegaaxis)))
    for m in range(SQSQ.shape[0]):
        for n in range(SQSQ.shape[1]):
            SQSQ[m, n, :] = interpolate.interp1d(omegaaxis_load,
            ↪  SQSQ_coarse[m, n, :])(omegaaxis)
    return SQSQ
```

A.2 curve_fit_ads.py

```
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np

# This file uses a least square method (pseudoinverse) to obtain the
↪  a-matrices describing ADs
# they are curve fitted to the transfer functions and plots all the fits
# [[ K^2(P1 i + P4)     K^2(P5 i + P6)     K^2(P2 i + P3)]
```

```
#  [ K^2(H5 i + H6)     K^2(H1 i + H4)     K^2(H2 i + H3)]
#  [ K^2(A5 i + A6)     K^2(A1 i + A4)     K^2(A2 i + A3)]]

df_dim = pd.read_excel("ADs_HALOGALAND_PUS.xlsx", sheet_name="Dim section model")
df_ADs = pd.read_excel("ADs_HALOGALAND_PUS.xlsx", sheet_name="Aerodynamic derivatives")
df_red_vel = pd.read_excel("ADs_HALOGALAND_PUS.xlsx", sheet_name="Reduced velocities")
df_mean_vel = pd.read_excel("ADs_HALOGALAND_PUS.xlsx", sheet_name="Mean wind velocity")

stiffness_ADs = ['P_4', 'P_6', 'P_3', 'H_6', 'H_4', 'H_3', 'A_6', 'A_4', 'A_3']
damping_ADs = ['P_1', 'P_5', 'P_2', 'H_5', 'H_1', 'H_2', 'A_5', 'A_1', 'A_2']

B = df_dim['B'][0]
d = np.array([1])
aa = np.zeros((3, 3, 3)) # curvefitting with a1, a2 and a4 (d=1)

n_dp = len(df_ADs[stiffness_ADs[0]].to_numpy()) # number of data points
xx = np.zeros((len(stiffness_ADs), 3, 2*n_dp))

V_hat = df_red_vel['P_4']
K = 1/V_hat # Every AD is measured with approx the same reduced velocity

for i in range(len(damping_ADs)):
    xx[i, 1, :n_dp] = K
    xx[i, 2, :n_dp] = np.imag(1j*K/(1j*K + d))

    xx[i, 0, n_dp:] = np.ones(len(K))
    xx[i, 2, n_dp:] = np.real(1j*K/(1j*K + d))

    AD_stiff = df_ADs[stiffness_ADs[i]].to_numpy()
    AD_damp = df_ADs[damping_ADs[i]].to_numpy()

    ADs = np.hstack([AD_damp*K**2, AD_stiff*K**2])

    a_s = np.linalg.pinv(xx[i]).T @ ADs
    aa[0][i//3,  i%3] = a_s[0]
    aa[1][i//3,  i%3] = a_s[1]
    aa[2][i//3,  i%3] = a_s[2]
np.savez("RFa_d.npz", aa=aa, d=d)
```

A.3 static_force_coefficients.py

```
"""
Finner statiske last koeffisenter fra Static_coeff_HALOGALAND_PUS.xlsx
"""
#%%
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

df_dim = pd.read_excel("Static_coeff_HALOGALAND_PUS.xlsx", sheet_name="Dim section model")
df_stat_coeff_0 = pd.read_excel("Static_coeff_HALOGALAND_PUS.xlsx", sheet_name="Mean wind 0.0")
df_stat_coeff_1 = pd.read_excel("Static_coeff_HALOGALAND_PUS.xlsx", sheet_name="Mean wind 1.0")
df_stat_coeff_6 = pd.read_excel("Static_coeff_HALOGALAND_PUS.xlsx", sheet_name="Mean wind 6.0")
df_stat_coeff_8 = pd.read_excel("Static_coeff_HALOGALAND_PUS.xlsx", sheet_name="Mean wind 8.0")
df_stat_coeff_10 = pd.read_excel("Static_coeff_HALOGALAND_PUS.xlsx", sheet_name="Mean wind 10.0")
stat_list = [df_stat_coeff_0 ,df_stat_coeff_1 ,df_stat_coeff_6 ,df_stat_coeff_8, df_stat_coeff_10]
coeff_list = ["C_D, C_L, C_m"]

alpha = df_stat_coeff_10["pitch motion"]
Cd = df_stat_coeff_10["C_D"].to_numpy()
Cl = df_stat_coeff_10["C_L"].to_numpy()
Cm = df_stat_coeff_10["C_m"].to_numpy()

## Zero angle of attack
alpha0_indx = np.argmin(np.abs(alpha))
C_D = Cd[alpha0_indx]
C_L = Cl[alpha0_indx]
C_m = Cm[alpha0_indx]

#%% Derivatives of coefficents
# Approximating the gradient of C_D
b_cd = np.argmin(np.abs(alpha+.04))
o_cd = np.argmin(np.abs(alpha-.02))
dCd = (Cd[o_cd]-Cd[b_cd])/(alpha[o_cd]-alpha[b_cd])

# Approximating the gradient of C_L
b_cl = np.argmin(np.abs(alpha+.025))
o_cl = np.argmin(np.abs(alpha-.025))
dCl = (Cl[o_cl]-Cl[b_cl])/(alpha[o_cl]-alpha[b_cl])

# Approximating the gradient of C_m
b_cm = np.argmin(np.abs(alpha+.025))
```

```
o_cm = np.argmin(np.abs(alpha-.025))
dCm = (Cm[o_cm]-Cm[b_cm])/(alpha[o_cm]-alpha[b_cm])

load_coeff = np.array([C_D, dCd, C_L, dCl, C_m, dCm])
#np.save("load_coefficients.npy", load_coeff)

fig, axs = plt.subplots(nrows=3, ncols=1, dpi=1000, sharex=True)

approx_x = np.array([np.min(alpha), np.max(alpha)])
approx_y1 = np.array([np.max(alpha*dCd+Cd[alpha0_indx]),
↪  np.min(alpha*dCd+Cd[alpha0_indx])])
approx_y2 = np.array([np.min(alpha*dCl+Cl[alpha0_indx]),
↪  np.max(alpha*dCl+Cl[alpha0_indx])])
approx_y3 = np.array([np.min(alpha*dCm+Cm[alpha0_indx]),
↪  np.max(alpha*dCm+Cm[alpha0_indx])])

axs[0].plot(alpha, Cd, ".", label=r"Data points", lw=.1)
axs[0].plot(approx_x, approx_y1*.99, "--",label=r"$C_D'|_{\alpha=0}$",
↪  lw=2)
axs[0].legend()
axs[0].grid(":")
axs[0].set_ylabel(r"$C_D(\alpha)$")

axs[1].plot(alpha, Cl, ".", label=r"Data points", lw=.7)
axs[1].plot(approx_x, approx_y2, "--",label=r"$C_L'|_{\alpha=0}$", lw=2)
axs[1].legend()
axs[1].grid(":")
axs[1].set_ylabel(r"$C_L(\alpha)$")

axs[2].plot(alpha, Cm, '.', label=r"Data points", lw=1)
axs[2].plot(approx_x, approx_y3, "--",label=r"$C_M'|_{\alpha=0}$", lw=2)
axs[2].legend()
axs[2].grid(":")
axs[2].set_ylabel(r"$C_M(\alpha)$")

axs[2].set_xlabel(r"$\alpha$")
plt.tight_layout()
fig.align_ylabels(axs)
#plt.savefig("force_coeff.png", bbox_inches='tight')
```

A.4 ImportModalProperties.py

```
## Script made by Øyvind Wiig Petersen
import h5py
import numpy as np
import matplotlib.pyplot as plt

hf = h5py.File('HalogalandModel_exportmodal.h5', 'r')
```

```
7
8  # Frequencies
9  f = np.array(hf.get('f'))
10
11 # Generalized mass
12 gm = np.array(hf.get('gm'))
13
14 # Node coordinates all nodes (size [N_nodes,4])
15 nodecoord = np.array(hf.get('nodecoord'))
16
17 # Mode shape matrix for all DOFs (size [6*N_nodes,N_modes])
18 phi = np.array(hf.get('phi'))
19
20 # Labels corresponding to each row of phi (each DOF), as a list of
   ↪  strings
21 phi_label_temp = np.array(hf.get('phi_label'))
22 phi_label=phi_label_temp[:].astype('U10').ravel().tolist()
23
24 # The DOFs are as follows:
25 # U1=x=longitudinal translation (along bridge)
26 # U2=y=horizontal translation
27 # U3=z=vertical translation
28 # UR1=rotation about x aka. torsion
29 # UR2=rotation about y aka. vertical bending
30 # UR3=rotation about z aka. horizontal bending
31
32 #%% Get node coordinates and mode shape for bridge deck only
33 # Nodes in bridge deck (total 573)
34 node_deck=np.arange(1004,1576+1,1)
35
36 # Create list index of nodes in bridge deck
37 index_node_deck=[]
38
39 for k in np.arange(len(node_deck)):
40     index_node_deck.append(np.argwhere(node_deck[k]==nodecoord[:,0])[0,0])
41
42 nodecoord_deck=nodecoord[index_node_deck,:]
43
44 # Create list of index of y-DOFs,z-DOFs, and t-DOFs in bridge deck
45 index_y=[]
46 index_z=[]
47 index_t=[]
48
49 for k in np.arange(len(node_deck)):
50     str_y=str(node_deck[k]) + '_U2'
51     index_y.append(phi_label.index(str_y))
52
53     str_z=str(node_deck[k]) + '_U3'
54     index_z.append(phi_label.index(str_z))
55
56     str_t=str(node_deck[k]) + '_UR1'
```

```
    index_t.append(phi_label.index(str_t))

phi_y=phi[index_y,:]
phi_z=phi[index_z,:]
phi_t=phi[index_t,:]

#%% Plot single mode
mode_plot=12 # Mode number to plot
scale_factor=1e4 # Scale factor

plt.figure()

x=nodecoord_deck[:,1] #x-coordinate of deck nodes

h1=plt.plot(x,phi_y[:,mode_plot]*scale_factor, label='Horizontal')
h2=plt.plot(x,phi_z[:,mode_plot]*scale_factor, label='Vertical')
h3=plt.plot(x,phi_t[:,mode_plot]*scale_factor, label='Torsion')

plt.xlabel('x [m]')
plt.ylabel('Modal deflection [m or rad]')

plt.legend()

plt.show()
```

A.5 stability.py

```
import numpy as np
from curve_fit_ads import aa, d
import ImportModalProperties as modal_properties
import matplotlib.pyplot as plt

#modechosen1 = 3
modechosen2 = 4
modechosen3 = 15

modes = np.array([modechosen2, modechosen3])
modes = np.array(range(0,100))

wn = modal_properties.f[modes]*2*np.pi

xin = np.ones_like(wn)*1/100
MM = np.diag(modal_properties.gm[modes])
KK = np.diag(wn**2*modal_properties.gm[modes])
CC = np.diag(modal_properties.gm[modes]*2*wn*xin) # Assumed diagonal
↪  generalised damping matrix

B = 18.6 # Width of Hålogaland
```

```
rho = 1.25 # Air density

length = modal_properties.x[-1] - modal_properties.x[0] # Length of
↪  bridge
nlength = len(modal_properties.x) # Number of increments
beam = np.linspace(0, length, nlength) # Discretisation of bridge

phiphi = np.zeros((len(modes), 3, nlength))
for m in range(phiphi.shape[0]):
    phiphi[m, 0, :] = modal_properties.phi_y[:, modes[m]]
    phiphi[m, 1, :] = modal_properties.phi_z[:, modes[m]]
    phiphi[m, 2, :] = modal_properties.phi_t[:, modes[m]]

AA = np.copy(aa)
for i in range(len(aa)):
    AA[i, :, -1] *= B
    AA[i, -1, :] *= B

AA_tilde = np.zeros((aa.shape[0], MM.shape[0], MM.shape[0]))
phiphi_p = np.transpose(phiphi, [1, 0, 2])

for k in range(aa.shape[0]):
    integrand = np.zeros((len(beam), MM.shape[0], MM.shape[0]))
    for m in range(len(beam)):
       integrand[m, :, :] = (phiphi_p[:, :, m].T @ AA[k, : , :] @
       ↪  phiphi_p[:, :, m])

    for i in range(MM.shape[0]):
        for j in range(MM.shape[0]):
            AA_tilde[k, i, j] = np.trapz(integrand[:, i, j], beam)

# Stability assesment
MMinv = np.linalg.inv(MM)

A1 = AA_tilde[0]
A2 = AA_tilde[1]

QQ_c = np.concatenate(AA_tilde[2:], axis=-1)
DD_c = np.diag(np.repeat(d, AA_tilde.shape[1])) # Eq. 16 Øiseth (2012)
EE_c = np.zeros((len(d), AA_tilde.shape[1], AA_tilde.shape[1]))
EE_c[:] = np.eye(AA_tilde.shape[1]) # Eq. 16 Øiseth (2012)
EE_c = np.concatenate(EE_c)

Vs = 1
dV = 1
it = 0
#%%
while (it<1000):
    it += 1
    Vs += dV

```

```
70      AA_c = -np.vstack((
71      np.hstack((
72          np.zeros_like(MM), -np.eye(MM.shape[0]), np.zeros((MM.shape[0],
            ↪  QQ_c.shape[1]))
73      )),
74      np.hstack((
75          MMinv@(KK-1/2*rho*Vs**2*A1), MMinv@(CC-1/2*rho*Vs**2*B/Vs*A2),
            ↪  -1/2*rho*Vs**2*MMinv@QQ_c
76      )),
77      np.hstack((
78          np.zeros((EE_c.shape[0], MM.shape[0])), -EE_c, Vs/B*DD_c
79      ))
80      ))
81      eig, _ = np.linalg.eig(AA_c)
82      if np.max(np.real(eig))>0:
83          Vs -= dV
84          dV *= 0.5
85
86  print(Vs)
87  x = Vs
88
89  #%%
90  Vs = np.linspace(1, 90, 1000)
91  eigS = np.zeros((len(Vs), AA_c.shape[0]), dtype=complex)
92  MMinv = np.linalg.inv(MM)
93
94  for i in range(len(Vs)):
95      AA_c = -np.vstack((
96      np.hstack((
97          np.zeros_like(MM), -np.eye(MM.shape[0]), np.zeros((MM.shape[0],
            ↪  QQ_c.shape[1]))
98      )),
99      np.hstack((
100         MMinv@(KK-1/2*rho*Vs[i]**2*A1),
            ↪  MMinv@(CC-1/2*rho*Vs[i]**2*B/Vs[i]*A2),
            ↪  -1/2*rho*Vs[i]**2*MMinv@QQ_c
101     )),
102     np.hstack((
103         np.zeros((EE_c.shape[0], MM.shape[0])), -EE_c, Vs[i]/B*DD_c
104     ))
105     ))
106
107     eig, _ = np.linalg.eig(AA_c)
108     indx = np.argsort(np.imag(eig))
109     eigS[i, :] = eig[indx]
110
111
112 fig, axs = plt.subplots(nrows=2, ncols=1, sharex=True, dpi=1000)
113
114 for i in range(len(modes)):
```

```
    axs[0].plot(Vs, freq[:, i],
    ↪  color="C{}".format(str(i)),label=(r"$\phi_{" +
    ↪  str(modes[-(i+1)]+1) + "}$") )
    axs[0].plot(Vs, freq[:, -(i+1)],color="C{}".format(str(i)))

for i in range(len(modes)):
    axs[1].plot(Vs, damp[:, i], label=(r"$\phi_{" + str(modes[-(i+1)]+1) +
    ↪  "}$") )

axs[0].set_ylabel(r'$\omega_n = $Im$(S)$')
axs[0].grid(ls=":")
axs[0].legend()

axs[1].set_ylabel(r'$\xi = -$Re$(S)/|S|$')
axs[1].set_xlabel(r"$V$ (m/s)")
axs[1].set_ylim(0, .25)
axs[1].grid(ls=":")
axs[1].legend()
plt.tight_layout()
fig.align_ylabels(axs)
#fig.savefig("critical velocity", bbox_inches='tight')0)
plt.show()
```

A.6 buffeting_response_time_domain.py

```
import numpy as np
import scipy.linalg as spla
import scipy.interpolate as spip
import ImportModalProperties as modal_properties # Import modal properties
↪  (from abaqus model)
from functions import distance_matrix, MCCholesky

#%% Hålogaland parameters/ constants
B = 18.6 # Section width
D = 3 # Section height
rho = 1.25 # Air denisty
L = (modal_properties.x[-1]-modal_properties.x[0]) # Length of bridge
nlength = len(modal_properties.x) # number of nodes along the length
beam = np.linspace(0, L, nlength) # discretising the bridge

#%% Wind field parameters Er disse verdiene riktig for Hålogaland`?`
L1 = 100.0 # Reference integral length scale
z1 = 10.0 # Reference height
z = 50.0 # Height above ground
xLu = L1*(z/z1)**0.3 # Integral length scale
xLw = 1/12*xLu # Integral length scale
Au, Aw = 6.8/2/np.pi, 9.4/2/np.pi # Constant in the auto-spectral density
Iu = 0.15 # Turbulence intensity
```

```
Iw = 1/4*Iu# Turbulence intensity
Cuy, Cwy = 10.0, 6.5 # Decay coeff

#%% MonteCarlos Simulations Wind Field
Nsim = 20
omegaaxis = np.linspace(0.00000001,10,100)
dxdx = distance_matrix(L, nlength)

#%% Modal properties
modes = np.arange(0, 100, 1) # all modes
wn = modal_properties.f[modes]*2*np.pi
xin = np.ones_like(wn)* 1/100

MM = np.diag(modal_properties.gm[modes])
KK = MM*np.diag(wn)**2
CC = 2*MM*np.diag(wn*xin)

phiphi = np.zeros((MM.shape[0], 3, len(beam)))
for m in range(MM.shape[0]):
    phiphi[m, 0, :] = modal_properties.phi_y[:, modes[m]]
    phiphi[m, 1, :] = modal_properties.phi_z[:, modes[m]]
    phiphi[m, 2, :] = modal_properties.phi_t[:, modes[m]]

#%% Creating a coarser discretisation, reduced to about 10 m segments
n = 120 # creater coarser discretisation
beam_red = np.linspace(0, L, n)
phiphi_red = np.zeros((phiphi.shape[0], phiphi.shape[1], len(beam_red)))
for i in range(phiphi.shape[0]):
    for j in range(phiphi.shape[1]):
        phiphi_red[i, j, :] = spip.interp1d(beam, phiphi[i, j,
        ↪  :])(beam_red)

phiphi = phiphi_red
beam = beam_red
dxdx = distance_matrix(L, n)

#%% Generalised rational functions
npzfile = np.load('results\\RFa_d.npz')
RFa = np.copy(npzfile['aa'])
d = npzfile['d']
for i in range(RFa.shape[0]):
    RFa[i, :, -1] *= B
    RFa[i, -1, :] *= B

RFa_gen = np.zeros((RFa.shape[0], MM.shape[0], MM.shape[0]))
phiphi_p = np.transpose(phiphi, [1, 0, 2])

for k in range(RFa.shape[0]):
    integrand = np.zeros((len(beam), MM.shape[0], MM.shape[0]))
    for m in range(len(beam)):
```

```
72          integrand[m, :, :] = phiphi_p[:, : , m].T @ RFa[k, :, :] @
            ↪  phiphi_p[:, :, m]
73      RFa_gen[k, :, :] = np.trapz(integrand, beam, axis=0)
74
75  #%% Static load coefficients
76  load_coeff = np.load('results\\load_coefficients.npy')
77  Cd, dCd = load_coeff[0], load_coeff[1]
78  Cl, dCl = load_coeff[2], load_coeff[3]
79  Cm, dCm = load_coeff[4], load_coeff[5]
80
81  #%% Stability assessment
82  MMinv = np.linalg.inv(MM)
83  QQ_c = np.concatenate(RFa_gen[2:], axis=-1)
84  DD_c = np.diag(np.repeat(d, RFa_gen.shape[1])) # Eq. 16 Øiseth (2012)
85  EE_c = np.zeros((len(d), RFa_gen.shape[1], RFa_gen.shape[1]))
86  EE_c[:] = np.eye(RFa_gen.shape[1]) # Eq. 16 Øiseth (2012)
87  EE_c = np.concatenate(EE_c)
88
89  #%% Calculating the response
90  mean_wind_velocities = np.array([10, 30, 50, 60, 70, 78, 80, 81])
91  covariance_matrix = np.zeros((phiphi.shape[1], phiphi.shape[1],
    ↪  len(mean_wind_velocities)))
92  rr = []
93  for v in range(len(mean_wind_velocities)):
94      V = mean_wind_velocities[v]
95      ## State matrix for selected mean wind velocity
96      AA_c = -np.vstack((
97              np.hstack((
98                  np.zeros_like(MM), -np.eye(MM.shape[0]),
                    ↪  np.zeros((MM.shape[0], QQ_c.shape[1]))
99                  )),
100             np.hstack((
101                 MMinv@(KK-1/2*rho*V**2*RFa_gen[0]),
                    ↪  MMinv@(CC-1/2*rho*V**2*B/V*RFa_gen[1]),
                    ↪  -MMinv@QQ_c*1/2*rho*V**2
102                 )),
103             np.hstack((
104                 np.zeros((EE_c.shape[0], MM.shape[0])), -EE_c, V/B*DD_c
105                 ))
106             ))
107     BB_c = np.vstack((
108         np.zeros_like(MM),
109         MMinv,
110         np.zeros((EE_c.shape[0], MM.shape[0]))
111     ))
112
113     ## Discrete time matrices
114     dt = .01
115     AA = spla.expm(AA_c*dt)
116     BB = np.linalg.inv(AA_c)@(AA - np.eye(AA.shape[0])) @ BB_c
117     CCss = np.hstack(([
```

```
            np.eye(MM.shape[0]), np.zeros_like(MM), np.zeros((MM.shape[0],
            ↪  EE_c.shape[0]))
            ]))

    ## Monte Carlos Simulations Wind Field
    SuSu = np.zeros((dxdx.shape[0], dxdx.shape[0], len(omegaaxis)))
    SwSw = np.zeros((dxdx.shape[0], dxdx.shape[0], len(omegaaxis)))
    Su = (Iu*V)**2*Au*xLu/V/((1+1.5*Au*omegaaxis*xLu/V)**(5.0/3.0))
    Sw = (Iw*V)**2*Aw*xLw/V/((1+1.5*Aw*omegaaxis*xLw/V)**(5.0/3.0))
    for k in range(len(omegaaxis)):
        Co_hat_u = np.exp(-Cuy/2/np.pi*dxdx*omegaaxis[k]/V)
        Co_hat_w = np.exp(-Cwy/2/np.pi*dxdx*omegaaxis[k]/V)
        SuSu[:, :, k] = Su[k]*Co_hat_u
        SwSw[:, :, k] = Sw[k]*Co_hat_w

    tsim, Xu = MCCholesky(omegaaxis, SuSu, Nsim)
    _, Xw = MCCholesky(omegaaxis, SwSw, Nsim)
    t = tsim[tsim<600]

    for uw in range(Nsim):
        ## Generalised buffeting load
        u = Xu[uw][:, tsim<600]
        w = Xw[uw][:, tsim<600]

        BqBq = 1/2*rho*V*B*np.array([
            [2*D/B*Cd, (D/B*dCd-Cl)],
            [2*Cl,     (dCl+D/B*Cd)],
            [2*B*Cm,          B*dCm]
            ])

        qq = np.zeros((3, len(t), len(beam)))
        for n in range(len(beam)):
            qq[:, :, n] = BqBq@np.real(np.array([u[n, :], w[n, :]]))

        QQ = np.zeros((MM.shape[0], len(t)))
        for k in range(len(t)):
            integrand = np.zeros((MM.shape[0], len(beam)))
            for n in range(len(beam)):
                integrand[:, n] = phiphi_p[:, :, n].T @ qq[:, k, n]
            QQ[:, k] = np.trapz(integrand, beam, axis=1)

        ## Convert the state space model to discrete time
        tint = np.arange(0, np.max(t), dt)
        QQint = spip.interp1d(t, QQ.T, axis=0)(tint)

        x = np.zeros((AA.shape[0], len(tint)))
        y = np.zeros((MM.shape[0], len(tint)))

        for k in range(len(tint)-1):
            x[:, k+1] = AA@x[:, k] + BB@QQint[k, :]
            y[:, k+1] = CCss@x[:, k]
```

```
168
169         resp = phiphi_p[:, :, round(len(beam)/2)] @y # midspan response
170         rr.append(resp)
171     print("{} of {} response calculations completed".format(v+1,
    ↪  len(mean_wind_velocities)))
172
173 np.savez("results\\time_domain_20nsim_600tsim.npz", rr=rr,
    ↪  mean_wind_velocities=mean_wind_velocities, Nsim=Nsim)
```

A.7 buffeting_response_frequency_domain.py

```
1  import numpy as np
2  import scipy.interpolate as spip
3  import halogaland_model.ImportModalProperties as modal_properties # Import
   ↪  modal properties (from abaqus model)
4  from functions import distance_matrix, frequency_response_matrix,
   ↪  cross_spectral_load_matrix
5
6  #%% Hålogaland parameters/ constants
7  B = 18.6 # Section width
8  D = 3 # Section height
9  rho = 1.25 # Air denisty
10 L = (modal_properties.x[-1]-modal_properties.x[0]) # Length of bridge
11 nlength = len(modal_properties.x) # number of nodes along the length
12 beam = np.linspace(0, L, nlength) # discretising the bridge
13 dxdx = distance_matrix(L, nlength)
14
15 omegaaxis = np.linspace(0.00000001, 10, 1000)
16 mean_wind_velocities = np.hstack((
17     np.arange(2, 75, 4),
18     np.arange(79, 82, .5)
19 ))
20
21 #%% Buffeting load matrix
22 load_coeff = np.load("results\\load_coefficients.npy")
23 Cd, dCd = load_coeff[0], load_coeff[1]
24 Cl, dCl = load_coeff[2], load_coeff[3]
25 Cm, dCm = load_coeff[4], load_coeff[5]
26 Axx = np.array([1, 1, 1, 1, 1, 1]) # Axx values set to 1 for simplicity
27
28 BqBq = 1/2*rho*B*np.array([
29     [2*D/B*Cd*Axx[0], ((D/B)*dCd-Cl)*Axx[1]],
30     [2*Cl*Axx[2],     (dCl+(D/B)*Cd)*Axx[3]],
31     [2*B*Cm*Axx[4],             B*dCm*Axx[5]]
32     ])
33
34 #%% Modal properties
35 modes = np.arange(0, 100, 1) # all modes
```

```
wn = modal_properties.f[modes]*2*np.pi
xin = np.ones_like(wn)* 1/100

MM = np.diag(modal_properties.gm[modes])
KK = MM*np.diag(wn)**2
CC = 2*MM*np.diag(wn*xin)

phiphi = np.zeros((MM.shape[0], 3, len(beam)))
for m in range(MM.shape[0]):
    phiphi[m, 0, :] = modal_properties.phi_y[:, modes[m]]
    phiphi[m, 1, :] = modal_properties.phi_z[:, modes[m]]
    phiphi[m, 2, :] = modal_properties.phi_t[:, modes[m]]

#%% Creating a coarser discretisation
n = 120 # creater coarser discretisation
beam_red = np.linspace(0, L, n)
phiphi_red = np.zeros((phiphi.shape[0], phiphi.shape[1], len(beam_red)))
for i in range(phiphi.shape[0]):
    for j in range(phiphi.shape[1]):
        phiphi_red[i, j, :] = spip.interp1d(beam, phiphi[i, j,
        ↪  :])(beam_red)

phiphi = phiphi_red
beam = beam_red
dxdx = distance_matrix(L, n)

phiphi_p = np.transpose(phiphi, [1, 0, 2])
xr = round(len(beam)/2)
np.savez('results\\modal_properties.npz', phiphi_p=phiphi_p, xr=xr)
#%% Loading curvefitted rational functions
npzfile = np.load('results\\RFa_d.npz')
RFa = np.copy(npzfile['aa'])
d = npzfile['d']

covariance_matrix = np.zeros((phiphi.shape[1], phiphi.shape[1],
↪  len(mean_wind_velocities)))
SRSR = np.zeros((phiphi.shape[1], phiphi.shape[1],
↪  len(mean_wind_velocities), len(omegaaxis)), dtype=complex)
import time

# Hva er det som tar så lang tid?
for i, V in enumerate(mean_wind_velocities):
    tic = time.time()
    HH = frequency_response_matrix(MM, CC, KK, V, omegaaxis, phiphi, rho,
    ↪  B, RFa, d, beam)
    toc = time.time()
    SQSQ = cross_spectral_load_matrix(omegaaxis, V, V*BqBq, dxdx, phiphi)
    for k, w in enumerate(omegaaxis):
        SRSR[:, :, i, k] = phiphi_p[:, :, xr] @ (HH[:, :, k] @ SQSQ[:, :,
        ↪  k] @ np.conj(HH[:, :, k]).T) @ phiphi_p[:, :, xr].T # Eq. (9)
        ↪  IABSE
```

```
    covariance_matrix[:, :, i] = np.real(np.trapz(SRSR[:, :, i, :],
    ↪  omegaaxis, axis=-1))


#%% Saving results
#np.savez('results\\covariance_matrix_frequency_domain.npz',
↪  covariance_matrix=covariance_matrix,
↪  mean_wind_velocities=mean_wind_velocities)
#np.savez('results\\SRSR.npz', SRSR=SRSR)
```

A.8 plot_ads.py

```
import pandas as pd
import matplotlib. pyplot as plt
import numpy as np

df_dim = pd.read_excel("ADs_HALOGALAND_PUS.xlsx", sheet_name="Dim section
↪  model")
df_ads = pd.read_excel("ADs_HALOGALAND_PUS.xlsx", sheet_name="Aerodynamic
↪  derivatives")
df_red_vel = pd.read_excel("ADs_HALOGALAND_PUS.xlsx", sheet_name="Reduced
↪  velocities")
df_mean_vel = pd.read_excel("ADs_HALOGALAND_PUS.xlsx", sheet_name="Mean
↪  wind velocity")
B = df_dim['B']
# Want to add all the different velocities into its own array
n_mean_vel = np.unique(np.round(df_mean_vel['P_1']).to_numpy(dtype=int))

#%%
stiffness_ads = ['P_4', 'P_6', 'P_3', 'H_6', 'H_4', 'H_3', 'A_6', 'A_4',
↪  'A_3']
damping_ads = ['P_1', 'P_5', 'P_2', 'H_5', 'H_1', 'H_2', 'A_5', 'A_1',
↪  'A_2']

fig_s, axs_s = plt.subplots(nrows=3, ncols=3, num="ADs related to
↪  stiffness")
fig_d, axs_d = plt.subplots(nrows=3, ncols=3, num="ADs related to
↪  damping")

for i in range(3):
    for j in range(3):
        mean_velocities_s =
        ↪  np.round(df_mean_vel[stiffness_ads[i*3+j]]).to_numpy(dtype=int)

        red_vel_stiff_1 =
        ↪  df_red_vel[stiffness_ads[i*3+j]][mean_velocities_s==n_mean_vel[0]]
        red_vel_stiff_2 =
        ↪  df_red_vel[stiffness_ads[i*3+j]][mean_velocities_s==n_mean_vel[1]]
```

```
26
27          ads_stiff_1 =
            ↪  df_ads[stiffness_ads[i*3+j]][mean_velocities_s==n_mean_vel[0]]
28          ads_stiff_2 =
            ↪  df_ads[stiffness_ads[i*3+j]][mean_velocities_s==n_mean_vel[1]]
29
30          red_vel_damp_1 =
            ↪  df_red_vel[damping_ads[i*3+j]][mean_velocities_s==n_mean_vel[0]]
            ↪  # redusert V ikke K!
31          red_vel_damp_2 =
            ↪  df_red_vel[damping_ads[i*3+j]][mean_velocities_s==n_mean_vel[1]]
32
33          ads_damp_1 =
            ↪  df_ads[damping_ads[i*3+j]][mean_velocities_s==n_mean_vel[0]]
34          ads_damp_2 =
            ↪  df_ads[damping_ads[i*3+j]][mean_velocities_s==n_mean_vel[1]]
35
36          axs_s[i][j].plot(red_vel_stiff_1, ads_stiff_1, '.', c="C0")
37          axs_s[i][j].plot(red_vel_stiff_2, ads_stiff_2, '.', c="C1")
38
39          axs_d[i][j].plot(red_vel_damp_1, ads_damp_1, '.', c="C0")
40          axs_d[i][j].plot(red_vel_damp_2, ads_damp_2, '.', c="C1")
41
42          axs_s[i][j].set_ylabel('$'+stiffness_ads[i*3+j]+'^*$')
43          axs_d[i][j].set_ylabel('$'+damping_ads[i*3+j]+'^*$')
44
45          axs_s[i][j].grid()
46          axs_d[i][j].grid()
47
48      axs_s[j][i].set_xlabel('$K=V/(B\cdot\omega)$')
49      axs_d[j][i].set_xlabel('$K=V/(B\cdot\omega)$')
50
51  fig_s.axes[-1].lines[0].set_label(r"$V\approx" +str(n_mean_vel[0])  +"$")
52  fig_s.axes[-1].lines[1].set_label(r"$V\approx" +str(n_mean_vel[1])  +"$")
53  fig_s.axes[-1].legend()
54
55  fig_d.axes[-1].lines[0].set_label(r"$V\approx" +str(n_mean_vel[0])  +"$")
56  fig_d.axes[-1].lines[1].set_label(r"$V\approx" +str(n_mean_vel[1])  +"$")
57  fig_d.axes[-1].legend()
58
59  fig_d.tight_layout()
60  fig_s.tight_layout()
61
62  plt.show()
63  \subsection{plot\_ads.py}
64
```

A.9 plot_mode_shapes.py

```
import h5py
import numpy as np
import matplotlib.pyplot as plt

hf = h5py.File('HalogalandModel_exportmodal.h5', 'r')

# Node coordinates all nodes (size [N_nodes,4])
nodecoord = np.array(hf.get('nodecoord'))

# Mode shape matrix for all DOFs (size [6*N_nodes,N_modes])
phi = np.array(hf.get('phi'))

# Labels corresponding to each row of phi (each DOF), as a list of strings
phi_label_temp = np.array(hf.get('phi_label'))
phi_label=phi_label_temp[:].astype('U10').ravel().tolist()

#%% Get node coordinates and mode shape for bridge deck only
# Nodes in bridge deck (total 573)
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))

phi_y=phi[index_y,:]
phi_z=phi[index_z,:]
phi_t=phi[index_t,:]

```

```
x=nodecoord_deck[:,1] #x-coordinate of deck nodes

# Plotting first 8 modes horisontal, vertical and torsional modes
horizontal_modes = [0, 2, 6, 9, 10, 13, 17, 18]
vertical_modes = [1, 3, 4, 5, 7, 8 , 11, 12]
torsional_modes = [15, 28, 31, 47, 52, 59, 64, 79]

fig, axs = plt.subplots(nrows=len(horizontal_modes), ncols=3, dpi=1000, sharex=True)
scale_factor = 10**4
for i in range(len(horizontal_modes)):

    axs[i, 0].plot(x/(x[0]-x[-1])+.5, phi_y[:, horizontal_modes[i]])
    axs[i, 0].plot(x/(x[0]-x[-1])+.5, phi_z[:, horizontal_modes[i]])
    axs[i, 0].plot(x/(x[0]-x[-1])+.5, phi_t[:, horizontal_modes[i]])

    axs[i, 1].plot(x/(x[0]-x[-1])+.5, phi_y[:, vertical_modes[i]], label=r"$\phi_y$")
    axs[i, 1].plot(x/(x[0]-x[-1])+.5, phi_z[:, vertical_modes[i]], label=r"$\phi_z$")
    axs[i, 1].plot(x/(x[0]-x[-1])+.5, phi_t[:, vertical_modes[i]], label=r"$\phi_t$")

    axs[i, 2].plot(x/(x[0]-x[-1])+.5, phi_y[:, torsional_modes[i]])
    axs[i, 2].plot(x/(x[0]-x[-1])+.5, phi_z[:, torsional_modes[i]])
    axs[i, 2].plot(x/(x[0]-x[-1])+.5, phi_t[:, torsional_modes[i]])

    axs[i, 0].set_ylabel(r"$\phi_{" + str(horizontal_modes[i]+1) + "}$")
    axs[i, 1].set_ylabel(r"$\phi_{" + str(vertical_modes[i]+1) + "}$")
    axs[i, 2].set_ylabel(r"$\phi_{" + str(torsional_modes[i]+1) + "}$")

    axs[i, 0].set_ylim(-np.max(np.abs(phi_y[:, horizontal_modes[i]]))*1.2, np.max(np.abs(phi_y[:, horizontal_modes[i]]))*1.2)
    axs[i, 1].set_ylim(-np.max(np.abs(phi_z[:, vertical_modes[i]]))*1.2, np.max(np.abs(phi_z[:, vertical_modes[i]]))*1.2)
    axs[i, 2].set_ylim(-np.max(np.abs(phi_t[:, torsional_modes[i]]))*1.2, np.max(np.abs(phi_t[:, torsional_modes[i]]))*1.2)

    axs[i, 0].grid(":")
    axs[i, 1].grid(":")
    axs[i, 2].grid(":")

plt.setp(plt.gcf().get_axes(), yticks=[])
plt.tight_layout()
fig.align_ylabels(axs[:, 0])
fig.align_ylabels(axs[:, 1])
fig.align_ylabels(axs[:, 2])

axs[0, 0].set_title(r"Horizontal mode shapes $\phi_y$", fontsize=10)
axs[0, 1].set_title(r"Vertical mode shapes $\phi_z$", fontsize=10)
axs[0, 2].set_title(r"Torsional mode shapes $\phi_t$", fontsize=10)
```

```
axs[-1, 0].set_xlabel(r"$x/L$")
axs[-1, 1].set_xlabel(r"$x/L$")
axs[-1, 2].set_xlabel(r"$x/L$")

axs[-1, 1].legend(loc='upper center', bbox_to_anchor=(0.5, -1.4),
          fancybox=True, shadow=False, ncol=3)
plt.savefig("Figures\\modeshapes.png", bbox_inches='tight')
```

A.10 plot_time_series.py

```
import numpy as np
import matplotlib.pyplot as plt
import ImportModalProperties2 as modal_properties # Import modal
↪  properties (from abaqus model)
from functions import distance_matrix, MCCholesky

#%% Hålogaland parameters/ constants
B = 18.6 # Section width
D = 3 # Section height
rho = 1.25 # Air denisty
L = (modal_properties.x[-1]-modal_properties.x[0]) # Length of bridge
nlength = len(modal_properties.x) # number of nodes along the length
beam = np.linspace(0, L, nlength) # discretising the bridge

#%% Wind field parameters Er disse verdiene riktig for Hålogaland`?`
L1 = 100.0 # Reference integral length scale
z1 = 10.0 # Reference height
z = 50.0 # Height above ground
xLu = L1*(z/z1)**0.3 # Integral length scale
xLw = 1/12*xLu # Integral length scale
Au, Aw = 6.8/2/np.pi, 9.4/2/np.pi # Constant in the auto-spectral density
Iu = 0.15 # Turbulence intensity
Iw = 1/4*Iu# Turbulence intensity
Cuy, Cwy = 10.0, 6.5 # Decay coeff

#%% MonteCarlos Simulations Wind Field
Nsim = 2
omegaaxis = np.linspace(0.00000001,10,100)
dxdx = distance_matrix(L, nlength)

#%% Creating a coarser discretisation, reduced to about 10 m segments
n = 120 # creater coarser discretisation
beam_red = np.linspace(0, L, n)
beam = beam_red
dxdx = distance_matrix(L, n)

#%% Plotting time series
V = 60
```

```
38
39 ## MonteCarlos Simulations Wind Field
40 SuSu = np.zeros((dxdx.shape[0], dxdx.shape[0], len(omegaaxis)))
41 SwSw = np.zeros((dxdx.shape[0], dxdx.shape[0], len(omegaaxis)))
42 Su = (Iu*V)**2*Au*xLu/V/((1+1.5*Au*omegaaxis*xLu/V)**(5.0/3.0))
43 Sw = (Iw*V)**2*Aw*xLw/V/((1+1.5*Aw*omegaaxis*xLw/V)**(5.0/3.0))
44 for k in range(len(omegaaxis)):
45     Co_hat_u = np.exp(-Cuy/2/np.pi*dxdx*omegaaxis[k]/V)
46     Co_hat_w = np.exp(-Cwy/2/np.pi*dxdx*omegaaxis[k]/V)
47     SuSu[:, :, k] = Su[k]*Co_hat_u
48     SwSw[:, :, k] = Sw[k]*Co_hat_w
49
50 tsim, Xu = MCCholesky(omegaaxis, SuSu, Nsim)
51 _, Xw = MCCholesky(omegaaxis, SwSw, Nsim)
52 t = tsim[tsim<600]
53
54 us = []
55 ws = []
56 for uw in range(Nsim):
57     u = Xu[uw][:, tsim<600]
58     w = Xw[uw][:, tsim<600]
59     us.append(u)
60     ws.append(w)
61
62 #%% Plotting
63
64 fig, axs = plt.subplots(nrows=4, ncols=1, sharex=True, dpi=1000)
65 quarterspan = len(beam)//4
66 midspan = len(beam)//2
67 plot_points = [quarterspan, midspan]
68 for i in range(2):
69     axs[2*i].plot(t, us[i][plot_points[i]], lw=1.1)
70     axs[2*i+1].plot(t, ws[i][plot_points[i]], lw=1.1)
71
72
73     axs[2*i].grid(ls=":")
74     axs[2*i+1].grid(ls=":")
75
76     axs[2*i].set_ylabel(r"$u_"+"{}".format(i+1)+" (t)$")
77     axs[2*i+1].set_ylabel(r"$w_"+"{}".format(i+1)+" (t)$")
78
79 axs[-1].set_xlabel(r"Time $t (s)$")
80 fig.align_ylabels()
81 fig.savefig("Figures\\time_series_simulation.png", bbox_inches="tight")
```

A.11 plot_self_excited_forces.py

```
1
2  #%%
3  import numpy as np
4  import matplotlib.pyplot as plt
5  from curve_fit_ads import aa, damping_ad, stiffness_ad, df_ADs, K
6  from scipy.linalg import expm
7
8  def wind_induced_self_excited_forces(t, rr, d, aa, rho, V, B):
9      """Calculates the wind induced self excited forces through
10     a state space model
11
12     Args:
13         t (array): time array
14         rr (array_like): contains the response (3xlen(t) ry, rz, rrot)
15         d (array): contains all d values obtain from the curve fitting of
   ↪  ADs(all values set to 1 for simplicity)
16         aa (array_like): all a matrices from the transfer function (3x3
   ↪  matrices * len(d)+2)
17         rho (double): air density
18         V (double): wind speed
19         B (double): width
20
21     Returns:
22         array_like: return the self excited forces
23     """
24     # Finding the selfexcited forces through state space model
25     dt = t[1] - t[0]
26     rr_dot = np.gradient(rr, axis=-1)/dt # velocity
27
28     I = np.eye(3)
29
30     D_c = -V/B * np.diag(np.repeat(d, len(I)))
31     E_c = np.zeros((len(d), len(I), len(I)))
32     E_c[:] = I
33     RFa = aa.copy()
34
35     for i in range(len(RFa)):
36         RFa[i, :, -1] *= B
37         RFa[i, -1, :] *= B
38
39     Q_c = 1/2 *rho*V**2 * RFa[2:]
40
41     x = np.zeros((len(d)*3, len(t)))
42
43     D = expm(D_c*dt)
44     E = np.linalg.inv(D_c)@ (D - np.eye(len(D)))@  E_c
45
46     Q = Q_c
```

```
    z = np.zeros_like(rr)
    q = np.zeros_like(rr)

    for k in range(len(t)-1):
        x[:, k+1] = D @ x[:, k] + E[0] @ rr_dot[:, k]
        z[:,k] = Q @ x[:, k]
        q[:,k] = 1/2 *rho*V**2*(RFa[0] @ rr[:, k]  + RFa[1] @ rr_dot[:,
        ↪  k]*B/V) + z[:, k]

    return q


# Testing the wind_induced_self_excited_forces with an arbitrary motion
# and one value for K
#%%
reduced_frequency_index = 7
K_red = K[reduced_frequency_index]


V = 40
B = 18.6
rho = 1.25
motion_freq = K_red*V/B


t = np.linspace(0, 20, 1000)
rr = np.array([.1*np.sin(motion_freq*(t+1/3)),
↪  .05*np.sin(motion_freq*(t+4/3)),
↪  .08*np.sin(motion_freq*(t+1/2))/2/np.pi])
d = np.array([1])

q = wind_induced_self_excited_forces(t, rr, d, aa, rho, V, B)

fig, axs = plt.subplots(nrows=3, ncols=2, sharex=True, dpi=1000,
↪  figsize=(8, 3))

axs[0][0].plot(t, rr[0],  c="C3", label=r"$\mathbf{u}$")
axs[1][0].plot(t, rr[1],  c="C3")
axs[2][0].plot(t, rr[2],  c="C3")

axs[0][1].plot(t[:-1], q[0][:-1]/1000, lw=2)
axs[1][1].plot(t[:-1], q[1][:-1]/1000, lw=2)
axs[2][1].plot(t[:-1], q[2][:-1]/1000, lw=2, label=r'$\mathbf{q}$
↪  (state-space)')

axs[0][0].set_ylabel(r"$u_y$ (m)")
axs[1][0].set_ylabel(r"$u_z$ (m)")
axs[2][0].set_ylabel(r"$u_\theta$ (rad)")

axs[0][1].set_ylabel(r"$q_y$ (kN/m)")
axs[1][1].set_ylabel(r"$q_z$ (kN/m)")
```

```
axs[2][1].set_ylabel(r"$q_\theta$ (kNm/m)")

fig.align_ylabels()
# Control against q = Cae* rr_dot + Kae *rr




K_ae = 1/2*rho*V**2*K_red**2 * np.array([[ df_ADs['P_4'][reduced_frequency_index], df_ADs['P_6'][reduced_frequency_index], B*df_ADs['P_3'][reduced_frequency_index] ],
                                         [ df_ADs['H_6'][reduced_frequency_index], df_ADs['H_4'][reduced_frequency_index], B*df_ADs['H_3'][reduced_frequency_index] ],
                                         [ B*df_ADs['A_6'][reduced_frequency_index], B*df_ADs['A_4'][reduced_frequency_index], B**2*df_ADs['A_3'][reduced_frequency_index] ]])

C_ae = 1/2*rho*V*K_red*B * np.array([[ df_ADs['P_1'][reduced_frequency_index], df_ADs['P_5'][reduced_frequency_index], B*df_ADs['P_2'][reduced_frequency_index] ],
                                         [ df_ADs['H_5'][reduced_frequency_index], df_ADs['H_1'][reduced_frequency_index], B*df_ADs['H_2'][reduced_frequency_index] ],
                                         [ B*df_ADs['A_5'][reduced_frequency_index], B*df_ADs['A_1'][reduced_frequency_index], B**2*df_ADs['A_2'][reduced_frequency_index] ]])



dt = t[1]-t[0]
rr_dot = np.gradient(rr, axis=-1)/dt
q_se = K_ae @ rr + C_ae @ rr_dot


axs[0][1].plot(t, q_se[0]/1000,"--")
axs[1][1].plot(t, q_se[1]/1000,"--")
axs[2][1].plot(t, q_se[2]/1000,"--", label=r'$\mathbf{q}=\mathbf{C}_{ae}\dot{\mathbf{u}}+\mathbf{K}_{ae}\mathbf{u}$')

axs[-1][0].set_xlabel(r"$t$ (s)")
```

```
axs[-1][1].set_xlabel(r"$t$ (s)")

handles, labels = axs[2][1].get_legend_handles_labels()
fig.legend(loc='upper center', bbox_to_anchor=(.5, 0),
           fancybox=True, shadow=False, ncol=3)

axs[0][0].grid(ls=":")
axs[1][0].grid(ls=":")
axs[2][0].grid(ls=":")

axs[0][1].grid(ls=":")
axs[1][1].grid(ls=":")
axs[2][1].grid(ls=":")

plt.tight_layout()
fig.savefig("Figures\\self excited forces.png", bbox_inches="tight")
plt.show()
#%%

```

A.12 plot_extreme_value.py

```
import matplotlib.pyplot as plt
import numpy as np
from gumbel import method_of_moments_gumbel
from gumbel_bootstrap import gumbel_parametric_bootstrap
from functions import *

#%% Frequency domian
npzfile_freq_dom = np.load('results\\freq_resp_NEW.npz')
covariance_matrix = npzfile_freq_dom['covariance_matrix']
mean_wind_velocities_freq = npzfile_freq_dom['mean_wind_vel']

#%% Time domain
npzfile_time_dom = np.load('results\\time_domain_20nsim_600tsim.npz')
rr = npzfile_time_dom['rr']
mean_wind_velocities_time = npzfile_time_dom['mean_wind_velocities']
Nsim = npzfile_time_dom['Nsim']

#%% Plotting extreme value distriubtion
for V in mean_wind_velocities_time[mean_wind_velocities_time==60]:
    #V = mean_wind_velocities_time[v]
    rr_mid = np.zeros((Nsim, rr.shape[-1]))

    for n in range(Nsim):
        rr_mid[n, :] = rr[1*Nsim + n, 1, :] # Choose which response to
        ↪  look at

```

```
    rr_mid_max = np.sort(np.amax(np.abs(rr_mid), axis=-1))
    p = np.zeros_like(rr_mid_max)
    for m in range(len(rr_mid_max)):
        p[m] = (m+1)/(len(rr_mid_max)+1)

    pt = .95

    mean = np.mean(rr_mid_max)
    std = np.std(rr_mid_max)

    beta = gumbel_beta(std)
    mu = gumbel_mu(mean, std)

    fig, axs = plt.subplots(nrows=1, ncols=2, dpi=1000, figsize=(9,4))

    rr_arr = np.linspace(0, 2*np.max(rr_mid_max), 10000)
    gumbel_cdf = gumbel_CDF(rr_arr, mean, std)
    rr_p = rr_arr[gumbel_cdf>pt][0]

    ll_p = -np.log(-np.log(p))

    axs[0].plot(ll_p, rr_mid_max, "o", label=r"Maximum response $m$")
    ## Linear regression
    reg_x = np.array([1.2*np.min(ll_p), 1.2*np.max(ll_p)])
    reg_y = reg_x*beta+mu
    axs[0].plot(reg_x, reg_y, label = "Linear regression")
    axs[0].plot(reg_x, [rr_p, rr_p], "k", label="{:.0f}th
    ↪  percentile".format(pt*100))

    axs[0].set_xlabel("$-\log(-\log(m/(N+1)))$")
    axs[0].set_ylabel("$u_{\max}$")
    axs[0].legend()
    axs[0].grid(ls=":")

    ## Bootstrapping

    Nsim_gum1, Nsim_gum2 = 100000, 20
    N = Nsim_gum1 * Nsim_gum2

    gumbel_mc_sim = mu - np.log(-np.log(np.random.rand(N)))*beta

    mat = np.reshape(gumbel_mc_sim, (Nsim_gum1 , Nsim_gum2))
    R = np.zeros(mat.shape[0])

    for i in range(len(R)):
        mean_i = np.mean(mat[i, :])
        std_i  = np.std(mat[i, :])

        betai = gumbel_beta(std_i)
        mui = gumbel_mu(mean_i, std_i)

```

```
76          R[i] = mui - np.log(-np.log(pt))*betai
77
78      R_hat = np.mean(R)
79      std_R = np.std(R)
80      w_q2 = 1.96 #95 % confidence interval
81      confidence_interval = np.array([R_hat - w_q2*std_R, R_hat +
        ↪  w_q2*std_R])
82      p_conf = .95
83      q = 1-p_conf
84      confidence_interval = np.array([np.sort(R)[int(q*Nsim_gum1/2)],
        ↪  np.sort(R)[int((1-q/2)*Nsim_gum1)]])
85      ## plot_bootstrap
86      Nbin = int(np.min([len(R)/10, 500]))
87      xout = np.linspace(np.min(R), np.max(R), Nbin)
88      hist = np.histogram(R, Nbin)
89      prob = hist[0]/len(R)/(xout[1] - xout[0])
90
91      axs[1].plot(xout, prob)
92      axs[1].plot(R_hat,0, "x", mfc='w', c="red", label=r"95th percentile")
93      axs[1].fill_between(
94          x=xout,
95          y1=prob,
96          where= (xout >= confidence_interval[0]) & (xout <=
            ↪  confidence_interval[1]),
97          alpha=1,
98      )
99
100     axs[1].plot(confidence_interval,[0, 0], "o", mfc='w', c="red",
        ↪  label=r"95\% confidence")
101     axs[1].set_xlabel("$u (m)$")
102     axs[1].set_ylabel("$PDF$")
103
104     axs[1].plot(xout, gumbel_PDF(xout, R_hat, std_R))
105     #axs[1].legend()
106     axs[1].grid(ls=":")
107     axs[1].set_xlim(xout[0], xout[-1]*.9)
108
109     fig.savefig("Figures\\gumbel_V=60_vertical.png", bbox_inches="tight")
110
111
112     #mu, beta = method_of_moments_gumbel(rr_mid, pt)
113     #gumbel_parametric_bootstrap(beta, mu, Nsim_gum1, Nsim_gum2, pt)
114 plt.show()
115
116 npzfile = np.load("results\\time_domain_20nsim_600tsim.npz")
117 rr = npzfile['rr']
118 mean_wind_velocities_time = npzfile['mean_wind_velocities']
119 Nsim = npzfile['Nsim']
120
121 #%%
122 fig, axs = plt.subplots(nrows=3, ncols=1, sharex=True, dpi=1000)
```

```
t = np.linspace(0, 600, rr.shape[-1])
axs[0].plot(t, rr[-(Nsim+1), 0, :], lw=1.3)
axs[1].plot(t, rr[-(Nsim+1), 1, :], lw=1.2)
axs[2].plot(t, rr[-(Nsim+1), 2, :], lw=1)

axs[0].set_ylabel(r"$u_y$ (m)")
axs[1].set_ylabel(r"$u_z$ (m)")
axs[2].set_ylabel(r"$u_\theta$ (rad)")

fig.align_ylabels()

axs[-1].set_xlabel(r"$t$ (s)")

axs[0].grid(ls=":")
axs[1].grid(ls=":")
axs[2].grid(ls=":")

axs[0].set_ylim(-13,13)

plt.tight_layout()

fig.savefig("Figures\\midspan_response_V=80.png", bbox_inches="tight")
```

A.13 plot_cdf_peak_frequency.py

```
import numpy as np
import matplotlib.pyplot as plt

npzfile_freq_dom = np.load('results\\freq_resp_NEW.npz')
mean_wind_velocities_freq = npzfile_freq_dom['mean_wind_vel']

SRSR = npzfile_freq_dom['SRSR']
omegaaxis = npzfile_freq_dom['omegaaxis']
# %% Mean wind velocities of interest
indx30 = np.where(mean_wind_velocities_freq==30.)
indx60 = np.where(mean_wind_velocities_freq==60.)
indx80 = np.where(mean_wind_velocities_freq==80.)
indices = np.array([indx30, indx60, indx80])

# %% probability distribution of the largest peak
ndofs = 3
nwinds = 3

sigma = np.zeros((ndofs, nwinds))
sigma_dot = np.zeros_like(sigma)


for i in range(ndofs):
```

```
    for j in range(nwinds):
        sigma[i, j] = np.sqrt(np.trapz(np.real(SRSR[i, i, indices[j]]),
        ↪  omegaaxis))
        sigma_dot[i, j] = np.sqrt(np.trapz(np.real(omegaaxis**2*SRSR[i, i,
        ↪  indices[j]]), omegaaxis))

a = np.linspace(0, 30, 100000)
vy = np.zeros((len(a), sigma.shape[1], sigma.shape[0]))
for k in range(len(a)):
    vy[k] = 1/2/np.pi * sigma_dot/sigma *np.exp(-1/2*(a[k]/sigma)**2)

T = 600
P = np.exp(-vy*T)

#%% Plotting results
fig, axs = plt.subplots(nrows=1, ncols=3, dpi=1000, figsize=(8,4))
for i in range(nwinds):
    axs[0].plot(a, P[:, 0, i], label=r"$V =
    ↪  "+str(mean_wind_velocities_freq[indices[i]][0][0])+ "$ m/s")
    axs[1].plot(a, P[:, 1, i])
    axs[2].plot(a, P[:, 2, i])

    pt = .95
    axs[0].plot(a[(np.abs(P[:, 0, i] - pt)).argmin()], P[ (np.abs(P[:, 0,
    ↪  i] - pt)).argmin(), 0, i], 'rx')
    axs[1].plot(a[(np.abs(P[:, 1, i] - pt)).argmin()], P[ (np.abs(P[:, 1,
    ↪  i] - pt)).argmin(), 1, i], 'rx')
    axs[2].plot(a[(np.abs(P[:, 2, i] - pt)).argmin()], P[ (np.abs(P[:, 2,
    ↪  i] - pt)).argmin(), 2, i], 'rx')

    print(a[(np.abs(P[:, 0, i] - pt)).argmin()], # pt response
          a[(np.abs(P[:, 1, i] - pt)).argmin()],
          a[(np.abs(P[:, 2, i] - pt)).argmin()])

    axs[0].set_xlim(0, 25)
    axs[1].set_xlim(0, 7)
    axs[2].set_xlim(0, .7)

    axs[0].set_xlabel(r"$u_y$ (m)")
    axs[1].set_xlabel(r"$u_z$ (m)")
    axs[2].set_xlabel(r"$u_\theta$ (rad)")

    axs[0].grid(ls=":")
    axs[1].grid(ls=":")
    axs[2].grid(ls=":")


axs[0].plot(-1,-1, "rx", label=r"CDF = 0.95")
axs[0].set_ylim(-.05, 1.05)
fig.legend(loc='upper center', bbox_to_anchor=(.5, 0),
          fancybox=True, shadow=False, ncol=4)
```

```
69
70 axs[0].set_ylabel(r"CDF")
71 axs[1].set_ylabel(r"CDF")
72 axs[2].set_ylabel(r"CDF")
73 plt.tight_layout()
74 fig.savefig('Figures\\probability_largest_peak.pdf', bbox_inches="tight")
```

A.14 plot_mid_span_response.py

```
1  import matplotlib.pyplot as plt
2  import numpy as np
3
4  #%% Frequency domian
5  npzfile_freq_dom = np.load('results\\freq_resp_NEW.npz')
6  covariance_matrix = npzfile_freq_dom['covariance_matrix']
7  mean_wind_velocities_freq = npzfile_freq_dom['mean_wind_vel']
8
9  #%% Time domain
10 npzfile_time_dom = np.load('results\\time_domain_20nsim_600tsim.npz')
11 rr = npzfile_time_dom['rr']
12 mean_wind_velocities_time = npzfile_time_dom['mean_wind_velocities']
13 Nsim = npzfile_time_dom['Nsim']
14
15 ## Statistical properties | standard deviation and correlation
   ↪  coefficients
16 cov_mat_time = np.zeros((rr.shape[1], rr.shape[1],
   ↪  len(mean_wind_velocities_time)))
17 for i in range(len(mean_wind_velocities_time)):
18     cov = np.zeros((3, 3, Nsim))
19     for n in range(Nsim):
20         cov[:, :, n] = np.cov(rr[i*Nsim + n])
21     cov_mat_time[:, :, i] = np.mean(cov, axis=-1) # Average all the
       ↪  covarianses
22
23 #%% Plotting the standard deviation and correlation coefficients
24 fig, axs = plt.subplots(nrows=3, ncols=3, dpi=200)
25 fig.suptitle(r"$\sigma$ and $\rho$ for midspan response")
26 labels = ["$\sigma_{yy}$", "$\\rho_{yz}$", "$\\rho_{y\\theta}$",
27           "$\\rho_{z y}$", "$\sigma_{zz}$", "$\\rho_{z\\theta}$",
28           "$\\rho_{\\theta y}$", "$\\rho_{\\theta z}$",
             ↪  "$\sigma_{\\theta\\theta}$" ]
29
30 from scipy.signal import savgol_filter
31 covariance_matrix = covariance_matrix[:, :, mean_wind_velocities_freq<80]
32 mean_wind_velocities_freq =
   ↪  mean_wind_velocities_freq[mean_wind_velocities_freq<80]
33
34 cov_mat_time = cov_mat_time[:, :, mean_wind_velocities_time<80]
```

```
mean_wind_velocities_time = mean_wind_velocities_time[mean_wind_velocities_time<80]
for i in range(3):
    for j in range(3):
        ax = axs[i][j]
        if i==j:
            ax.plot(mean_wind_velocities_freq, covariance_matrix[i, j]**.5, 'b', lw=1)
            #ax.plot(mean_wind_velocities_freq, savgol_filter(covariance_matrix[i, j]**.5, 51, 6), 'black', lw=1)
            ax.plot(mean_wind_velocities_time, cov_mat_time[i, j]**.5, 'rx', lw=.5)
            ax.set_ylabel(labels[3*i+j])
            ax.set_xlim(10, 100)
        else:
            ax.plot(mean_wind_velocities_freq, covariance_matrix[i, j]/(covariance_matrix[i, i]**.5)/(covariance_matrix[j, j]**.5), "b", lw=1)
            #ax.plot(mean_wind_velocities_freq, savgol_filter(covariance_matrix[i, j]/(covariance_matrix[i, i]**.5)/(covariance_matrix[j, j]**.5), 51, 4), 'black', lw=1)
            ax.plot(mean_wind_velocities_time, cov_mat_time[i, j]/(cov_mat_time[i, i]**.5)/(cov_mat_time[j, j]**.5), 'rx', lw=.5)
            ax.set_ylabel(labels[3*i+j])
            ax.set_xlim(15, 100)
            ax.set_ylim(-1, 1)
plt.tight_layout()
```

A.15 plot_free_vibration.py

```
import numpy as np
import matplotlib.pyplot as plt
import scipy.interpolate as spip
import scipy.linalg as spla
# Import modal properties (from abaqus model)
import ImportModalProperties2 as modal_properties
from functions import distance_matrix

# %% Hålogaland parameters/ constants
B = 18.6  # Section width
D = 3  # Section height
rho = 1.25  # Air denisty
L = (modal_properties.x[-1]-modal_properties.x[0])  # Length of bridge
nlength = len(modal_properties.x)  # number of nodes along the length
beam = np.linspace(0, L, nlength)  # discretising the bridge
```

```
16
17
18 # %% MonteCarlos Simulations Wind Field
19 Nsim = 20
20 omegaaxis = np.linspace(0.00000001, 10, 110)
21 dxdx = distance_matrix(L, nlength)
22
23 # %% Modal properties
24 modes = np.arange(0, 100, 1)  # all modes
25 wn = modal_properties.f[modes]*2*np.pi
26 xin = np.ones_like(wn) * 1/100
27
28 MM = np.diag(modal_properties.gm[modes])
29 KK = MM*np.diag(wn)**2
30 CC = 2*MM*np.diag(wn*xin)
31
32 phiphi = np.zeros((MM.shape[0], 3, len(beam)))
33 for m in range(MM.shape[0]):
34     phiphi[m, 0, :] = modal_properties.phi_y[:, modes[m]]
35     phiphi[m, 1, :] = modal_properties.phi_z[:, modes[m]]
36     phiphi[m, 2, :] = modal_properties.phi_t[:, modes[m]]
37
38 # %% Creating a coarser discretisation, reduced to about 10 m segments
39 n = 120  # creater coarser discretisation
40 beam_red = np.linspace(0, L, n)
41 phiphi_red = np.zeros((phiphi.shape[0], phiphi.shape[1], len(beam_red)))
42
43 for i in range(phiphi.shape[0]):
44     for j in range(phiphi.shape[1]):
45         phiphi_red[i, j, :] = spip.interp1d(beam, phiphi[i, j,
          ↪  :])(beam_red)
46
47 phiphi = phiphi_red
48 beam = beam_red
49 dxdx = distance_matrix(L, n)
50
51 # %% Generalised rational functions
52 npzfile = np.load('results\\RFa_d.npz')
53 RFa = np.copy(npzfile['aa'])
54 d = npzfile['d']
55 for i in range(RFa.shape[0]):
56     RFa[i, :, -1] *= B
57     RFa[i, -1, :] *= B
58
59 RFa_gen = np.zeros((RFa.shape[0], MM.shape[0], MM.shape[0]))
60 phiphi_p = np.transpose(phiphi, [1, 0, 2])
61
62 for k in range(RFa.shape[0]):
63     integrand = np.zeros((len(beam), MM.shape[0], MM.shape[0]))
64     for m in range(len(beam)):
65         integrand[m, :, :] = phiphi_p[:, :,
```

```
66                                          m].T @ RFa[k, :, :] @ phiphi_p[:, :,
                                             ↪  m]
67      RFa_gen[k, :, :] = np.trapz(integrand, beam, axis=0)
68
69
70  # %% Calculating the free vibration response for the critical velocity and
    ↪  a higher velocity
71  Vcr = 80.69
72  Vocr = 82
73  Vs = np.array([Vocr, Vcr])
74  fig, axs = plt.subplots(nrows=3, ncols=1, sharex=True, dpi=1000)
75  for i, V in enumerate(Vs):
76
77      t = np.linspace(0, 200, 10000)
78      dt = t[1] - t[0]
79
80      AA = RFa_gen.copy()
81
82      A1 = AA[0] * 1/2*rho*V**2
83      A2 = AA[1] * 1/2*rho*V**2 *B/V
84      QQ = AA[2:]
85      d = np.atleast_1d(d)
86
87      QQ_c = np.concatenate(AA[2:], axis=-1) * 1/2*rho*V**2
88      DD_c = V/B*np.diag(np.repeat(d, AA.shape[1])) # Eq. 16 Øiseth (2012)
89      EE_c = np.zeros((len(d), AA.shape[1], AA.shape[1]))
90      EE_c[:] = np.eye(AA.shape[1]) # Eq. 16 Øiseth (2012)
91      EE_c = np.concatenate(EE_c)
92
93      MMinv = np.linalg.inv(MM)
94
95      AA_c = -np.vstack((
96              np.hstack((np.zeros_like(MM),
97                         -np.eye(MM.shape[0]),
98                         np.zeros((MM.shape[0], QQ.shape[1])))),
99              np.hstack((MMinv@(KK-A1),
100                        MMinv@(CC-A2),
101                        -MMinv@QQ_c)),
102             np.hstack((np.zeros((EE_c.shape[0], MM.shape[0])),
103                        -EE_c,
104                        DD_c))
105             ))
106     P = np.zeros((len(wn), len(t)))
107     PP = np.vstack((
108             np.zeros_like(P),
109             P,
110             np.zeros((MM.shape[0]*len(d), P.shape[1]))))
111
112     BB_c = np.zeros((PP.shape[0], PP.shape[0]))
113     BB_c[MM.shape[0]:MM.shape[0]*2, MM.shape[0]:MM.shape[0]*2] =
        ↪  np.linalg.inv(MM)
```

```
114
115     AA_d = spla.expm(AA_c* dt)
116     BB_d = np.linalg.inv(AA_c) @ (AA_d - np.eye(len(AA_d))) @ BB_c
117
118     sol = np.zeros_like(PP)
119     sol[:, 0] = np.ones(len(sol[0:, 0]))*1000
120
121     for k in range(len(t)-1):
122         sol[:, k+1] = AA_d @ sol[:, k] + BB_d @ PP[:, k]
123
124     rr = sol[:MM.shape[0]]
125     rr_dot = sol[MM.shape[0]:2*MM.shape[0]]
126     xx = sol[2*MM.shape[0]:]
127
128     # midspan repsonse
129     rr_mid = phiphi_p[:, :, round(len(beam)/2)] @ rr
130
131
132     axs[0].plot(t, rr_mid[0, :], label= r"$V="+str(V)+"$ m/s")
133     axs[1].plot(t, rr_mid[1, :])
134     axs[2].plot(t, rr_mid[2, :], label= r"$V="+str(V)+"$ m/s")
135
136 axs[0].set_ylabel(r"$u_y$ (m)")
137 axs[1].set_ylabel(r"$u_z$ (m)")
138 axs[2].set_ylabel(r"$u_\theta$ (rad)")
139
140 axs[0].grid(ls=":")
141 axs[1].grid(ls=":")
142 axs[2].grid(ls=":")
143
144 fig.align_ylabels()
145 axs[2].set_xlabel(r"$t$ (s)")
146 handles, labels = axs[2].get_legend_handles_labels()
147 axs[2].legend(handles[::-1], labels[::-1], loc='upper center',
    ↪  bbox_to_anchor=(0.5, -.5),
148           fancybox=True, shadow=False, ncol=2)
149
150 plt.tight_layout()
151 fig.savefig("Figures\\Free vibration response.png", bbox_inches="tight")
152 # %%
153
154
```

A.16 plot_frequency_and_welch.py

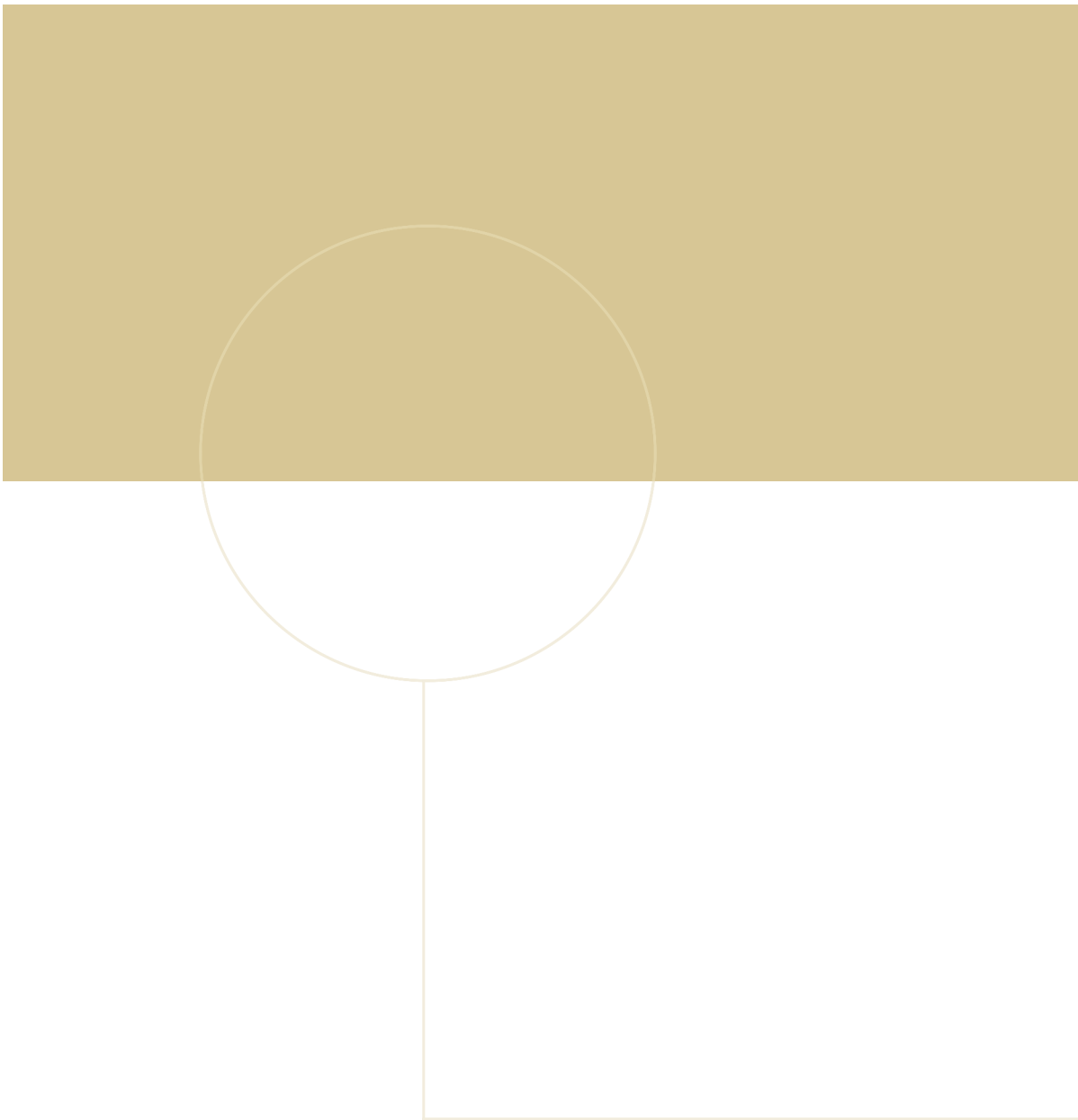
```
1 import matplotlib.pyplot as plt
2 import numpy as np
3 from scipy.signal import csd
```

```
4
5  #%% Time domain
6  npzfile_time_dom = np.load('results\\time_domain_20nsim_600tsim.npz')
7  rr = npzfile_time_dom['rr'] # midspan response
8  mean_wind_velocities_time = npzfile_time_dom['mean_wind_velocities']
9  Nsim = npzfile_time_dom['Nsim']
10
11 #%% Frequency domain
12 npzfile_freq = np.load('results\\freq_resp_NEW.npz')
13 SRSR = npzfile_freq['SRSR']
14 omegaaxis = npzfile_freq['omegaaxis']
15 mean_wind_vel = npzfile_freq['mean_wind_vel']
16
17 #%% Comparing the response of time and frequency domain with welch's
   ↪ method
18 Ndiv = 1
19 Nwindow = np.ceil(rr.shape[-1]/Ndiv) # Length of window/segment
20 Nfft_pow2 = 2**(np.ceil(np.log2(Nwindow))) # Next power of 2 for zero
   ↪ padding
21 t = np.linspace(0, 600, rr.shape[-1])
22 dt = t[1] - t[0]
23
24
25 # Spectral matrix
26 V_plot = np.array([30, 60, 80])
27 S_welch=np.zeros((len(V_plot), 3, 3, np.int32(Nfft_pow2/2+1)),
   ↪ dtype=np.complex_)
28 # Fill spectral matrix by taking the average of the cross spectral density
   ↪ between
29 for v in range(len(V_plot)):
30     for k1 in range(3):
31         for k2 in range(3):
32                 indx =
                    ↪ np.where(mean_wind_velocities_time==V_plot[v])[0][0]
33                 f, S_Hz = csd(rr[indx*Nsim:(indx+1)*Nsim, k1, :],
                    ↪ rr[indx*Nsim:(indx+1)*Nsim, k2, :], fs=1/dt,
                    ↪ window='hann', nperseg=Nwindow, noverlap=None,
                    ↪ nfft=Nfft_pow2, detrend='constant',
                    ↪ return_onesided=True, scaling='density', axis=-1,
                    ↪ average='mean')
34
35                 w_welch = f*2*np.pi # Frequency axis in rad/s
36                 S_welch[v, k1, k2, :]= np.average(S_Hz, axis=0)/(2*np.pi)
                    ↪ # Spectrum in rad/s
37
38 #%% Plotting the cross spectra
39 fig, axs = plt.subplots(nrows=3, ncols=1, dpi=1000, sharex=True)
40 labels = ["u_y", "u_z", "u_\\theta"]
41 for v in range(len(V_plot)):
42     V = V_plot[v]
43     for k1 in range(3):
```

```
            ax = axs[k1]
            ax.plot(w_welch,np.real(S_welch[v, k1, k1, :]), label="V = {}
            ↪  m/s".format(V))

            if V_plot[v] in mean_wind_vel:
                indx2 = np.where(mean_wind_vel==V_plot[v])[0][0]
                ax.plot(omegaaxis, np.real(SRSR[k1, k1, indx2, :]), '--',
                ↪  label="V = {} m/s".format(V))
            ax.set_ylabel(('$S_{' + labels[k1] + " " + labels[k1] +
            ↪  '}(\omega)$'))
            ax.grid(ls=":")
            ax.set_yscale('log')
            ax.set_xlim(0,5)
            ax.set_ylim(10**(-10), 2*10**3)

ax.set_xlabel('$\omega$')
ax.legend(loc='upper center', bbox_to_anchor=(0.5, -.5),
          fancybox=True, shadow=False, ncol=3)
plt.tight_layout()
plt.savefig("Figures\\welch.png", bbox_inches='tight')
```



NTNU

Norwegian University of
Science and Technology