Mari Voll Dombu
Markus Gjelstad

Parametric modelling of a suspension bridge with an aluminium girder

Buffeting response and flutter stability

Master's thesis in Civil and Environmental Engineering
Supervisor: Ole Øiseth
June 2019

Master's thesis

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

NTNU
Norwegian University of
Science and Technology

Mari Voll Dombu
Markus Gjelstad

Parametric modelling of a suspension bridge with an aluminium girder

Buffeting response and flutter stability

Master's thesis in Civil and Environmental Engineering
Supervisor: Ole Øiseth
June 2019

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 2019

SUBJECT AREA:	DATE:	NO. OF PAGES:
Structural Dynamics	06.06.2019	18+62+30=110

TITLE:

Parametric modelling of a suspension bridge with an aluminium girder - Buffeting response and flutter stability

Parametrisk modellering av ei hengebru med brukasse i aluminium - Buffetingrespons og flutterstabilitet

BY:

Mari Voll Dombu and Markus Gjelstad

SUMMARY:
This thesis investigates the dynamic behaviour of a suspension bridge with an aluminium girder over Langenuen in Norway. The main focus has been the buffeting response and flutter stability. A preliminary design has been used as a basis for constructing a parametric finite element model using Abaqus Scripting. Further on, modal data such as frequencies, vibration modes and generalized masses have been extracted from Abaqus. The buffeting response and the flutter instability limits are found by using aerodynamic derivatives from wind tunnel testing of the Great Belt East Bridge, since no such data exists for the Langenuen bridge. Supplementary matlab scripts have been used to calculate the results. The results from the dynamic analyses have been compared by changing the girder material from aluminium to steel.
The standard deviations of the buffeting response are generally higher for aluminium than for steel. In the horizontal direction, the aluminium option had a maximum standard deviation of about 750 mm at midspan for a wind velocity of 40 m/s, while the steel option had 320 mm. In the vertical direction, the maximum for aluminium was about 500 mm in the quarter span, and 380 mm for steel.
Both the aluminium and steel options have acceptable stability against flutter. The design wind velocity was calculated to be 63.6 m/s, while the critical wind velocity for aluminium and steel was 74.2 m/s and 106.2 m/s, respectively.
There are historically no known suspension bridges that has aluminium as the only structural material in the girder, so for that reason some changes to the design are expected to occur during the time of the project. This was also the case for this thesis. The cross sectional properties have been updated during the work with this thesis, and the cable and hanger diameters are optimized with respect to a utilization of 30% under static self weight. With a parametric model, it was easy to do changes to the input in the script and then generate a new finite element model. It turned out to be a good investment to spend extra time on making the model parametric as creating a model with new cross sectional parameters and a new material definition was carried out in a short amount of time.
Based on the founds in this thesis, the dynamic behaviour due to wind is acceptable for the suspension bridge with an aluminium girder. However, the cost will be a crucial factor when choosing a material, and this has only been discussed briefly. There are some aspects beyond the scope of this thesis that would also be important to look further into, such as fatigue.

RESPONSIBLE TEACHER: Ole Øiseth

SUPERVISOR(S): Ole Øiseth, Øyvind Wiig Petersen

CARRIED OUT AT: Department of Structural Engineering, NTNU

Institutt for konstruksjonsteknikk

FAKULTET FOR INGENIØRVITENSKAP
NTNU – Norges teknisk-naturvitenskapelige universitet

MASTEROPPGAVE 2019

for

Mari Voll Dombu og Markus Gjelstad

Parametric modelling of a suspension bridge with an aluminium girder - Buffeting response and flutter stability

Parametrisk modellering av ei hengebru med brukasse i aluminium - Buffetingrespons og flutterstabilitet

Statens vegvesen planlegger en ombygning av E39 slik at det blir mulig å reise fra Trondheim til Kristiansand ferjefritt. Dette innebærer at det skal bygges en rekke brukonstruksjoner. En av disse bruene vil krysse Langenuen. Denne oppgaven dreier seg om å undersøke om det er mulig å krysse fjorden med ei hengebru med brukasse i aluminium.

Oppgavens formål er å:
• Utvikle et brukonsept for kryssing av Langenuen.
• Bestemme den vindinduserte dynamiske responsen for den valgte utformingen
• Bestemme den aeroelastiske stabilitetsgrensen.

Løsningen av oppgaven bør inneholde følgende:
• Søk i litteraturen etter lastkoeffisienter og aerodynamiske deriverte.
• Etablering av en parametrisk abaqusmodell ved hjelp av et pythonskript.
• Beregning av egenfrekvenser og svingeformer
• Beregning av aeroelastisk stabilitetsgrense
• Beregning av vindindusert dynamisk respons i frekvens- eller tidsplanet.

Besvarelsen organiseres i henhold til gjeldende retningslinjer.

Veileder(e): Ole Øiseth, Øyvind Wiig Petersen

Besvarelsen skal leveres til Institutt for konstruksjonsteknikk innen 11. juni 2019.

Ole Øiseth
faglærer

Abstract

This thesis investigates the dynamic behaviour of a suspension bridge with an aluminium girder over Langenuen in Norway. The main focus has been the buffeting response and flutter stability. A preliminary design has been used as a basis for constructing a parametric finite element model using Abaqus Scripting. Further on, modal data such as frequencies, vibration modes and generalized masses have been extracted from Abaqus. The buffeting response and the flutter instability limits are found by using aerodynamic derivatives from wind tunnel testing of the Great Belt East Bridge, since no such data exists for the Langenuen bridge. Supplementary matlab scripts have been used to calculate the results. The results from the dynamic analyses have been compared by changing the girder material from aluminium to steel.

The standard deviations of the buffeting response are generally higher for aluminium than for steel. In the horizontal direction, the aluminium option had a maximum $\sigma_{r_y r_y}$ of about 750 mm at midspan for a wind velocity of 40 m/s, while the steel option had 320 mm. In the vertical direction, the aluminium option had a maximum $\sigma_{r_z r_z}$ of about 500 mm in the quarter span, in oppose to 380 mm for the steel option.

Both the aluminium and steel options have acceptable stability against flutter. The design wind velocity was calculated to be 63.6 m/s, while the critical wind velocity for aluminium and steel was 74.2 m/s and 106.2 m/s, respectively.

There are historically no known suspension bridges that has aluminium as the only structural material in the girder, so for that reason some changes to the design are expected to occur during the time of the project. This was also the case for this thesis. The cross sectional properties have been updated during the work with this thesis, and the cable and hanger diameters are optimized with respect to a utilization of 30 $\%$ under static self weight. With a parametric model, it was easy to do changes to the input in the script and then generate a new finite element model. It turned out to be a good investment to spend extra time on making the model parametric, as creating a model with new cross sectional parameters and a new material definition was carried out in a short amount of time.

Based on the founds in this thesis, the dynamic behaviour due to wind is acceptable for the suspension bridge with an aluminium girder. However, the cost will be a crucial factor when choosing a material, and this has only been discussed briefly. There are some aspects beyond the scope of this thesis that would also be important to look further into, such as fatigue.

Sammendrag

Denne oppgaven undersøker den dynamiske oppførselen til ei hengebru over Langenuen med brukasse av aluminium. Det har vært fokus på buffetingrespons og flutterstabilitet. Et foreløpig design er brukt som grunnlag for å opprette en parametrisk elementmodell ved bruk av Abaqus Scripting. Videre har modale data som frekvenser, vibrasjonsmoder og generaliserte masser blitt hentet ut fra Abaqus. For å finne buffetingrespons og kritisk flutterstabilitetsgrense er aerodynamisk deriverte funnet fra vindtunneltester av Storebæltbroen blitt brukt, siden disse ikke finnes for Langenuen bru. Skript i Matlab er brukt for å regne ut resultatene. Resultatene fra de dynamiske analysene er sammenlignet ved å bytte materialet i brukassen fra aluminium til stål.

Buffetinganalysen viste at standardavvikene av buffetingresponsen generelt er høyere for aluminium enn for stål. I horisontal retning hadde aluminiumalternativet en maksimum $\sigma_{r_y r_y}$ på omtrent 750 mm for $x = \frac{L}{2}$ med en vindhastighet på 40 m/s, mens stålalternativet hadde 320 mm. I vertikal retning fikk aluminiumalternativet en maksimum $\sigma_{r_z r_z}$ på omtrent 500 mm for $x = \frac{L}{4}$ mens stålalternativet fikk 380 mm. Dette var som forventet, siden aluminium har lavere stivhet enn stål.

Både aluminium- og stålalternativet har akseptabel stabilitet mot flutter. Kravet til kritisk vindhastighet ble beregnet til 63.6 m/s, mens V_{CR} for henholdsvis aluminimum og stål var 74.2 m/s og 106.2 m/s.

Det er historisk sett ingen kjente prosjekter med hengebruer som har brukasse i aluminium, så av den grunn er det ventet at endringer til designet vil oppstå underveis i prosjektet. Dette var også tilfelle for denne oppgaven. Tverrsnittsparametrene har blitt oppdatert underveis, og diametrene til hovedkablene og hengekablene har blitt optimalisert for en utnyttelsesgrad på 30 $\%$ under statisk egenvekt. Med et parametrisk skript har det vært lett å raskt gjøre endringer i input og deretter generere en ny elementmodell. Det viste seg å være en god investering å bruke ekstra tid på å gjøre modellen parametrisk, siden det tok kort tid å få en modell med nye tverrsnittsparametre og ny materialdefinisjon.

Basert på funnene i denne oppgaven er den dynamiske oppførselen for hengebrua med brukasse i aluminium akseptabel når den er påkjent vind. Kostnad vil være en viktig faktor ved valg av material, og er bare blitt diskutert kort. Det er noen viktige aspekter utenfor omfanget av denne oppgaven som det vil være viktig å undersøke nærmere, slik som utmatting.

Preface

This thesis is the result of 20 weeks of work, and concludes our five year Master's degree in Civil and Environmental Engineering at NTNU. We have gained unique insight within bridge aerodynamics and parametric modelling with the use of python. These subjects have been very rewarding and educational, but also challenging. The Abaqus Scripting was new to us, which combined with a new programming language resulted in many hours of trying and failing before we had a satisfying model.

We would like to thank Professor Ole Øiseth for his guidance and constructive feedback. We would also like to thank Dr. Øyvind Wiig Petersen for the many hours of help in the modelling and interpretation of the results. PhD-candidate Tore Helgedagsrud has on behalf of Dr.techn. Olav Olsen provided us with information about the girder cross section. Lastly, we would like to thank our fellow students at the office for mid-hour shuffleboard tournaments and the good spirit.

Mari Voll Dombu and Markus Gjelstad
Trondheim, 05.06.19

Contents

List of Figures

List of Tables

Abbreviations

AD	=	Aerodynamic derivative
CFD	=	Computational fluid dynamics
DOF	=	Degree of freedom
EOM	=	Equation of motion
FE	=	Finite element
FRF	=	Frequency response function
SDOF	=	Single degree of freedom

1 Introduction

Designing of bridges have undergone an astonishing development from the primitive river crossings with tree logs and rocks, through the large arch viaducts of ancient Rome, and until long span suspension bridges of today. Over the years span lengths have increased rapidly, opening for new structural challenges. A crucial factor is their ability to withstand strong winds. Dynamic behaviour must be taken into consideration when designing long and slender suspension bridges, because the wind could cause critical response or instabilities.

Steel is the governing material of the bridge decks in existing suspension bridges. The industry is well experienced with the usage of steel, it is easily accessible, it is cost efficient relative to similar materials, it is ductile and has a high tensile strength. In other words, there are many arguments in favour of steel. As of today, no bridge decks of suspension bridges are yet made of aluminium. It is approximately three times more expensive than steel, there is a lack of experience in the industry on the usage of aluminium, and cutting edge design is required in the design of the first suspension bridge with an aluminium girder. However, the application of the material in other structures has proven to be successful, and especially the lower self weight could be an argument for investigating the possibility of using aluminium in bridge decks.

Figure 1: Illustration of the Langenuen crossing [1]

The Norwegian Public Roads Administration is planning a ferry free highway along the Norwegian coast from Kristiansand to Trondheim. The aim is to connect the coastal cities in a better way and reduce the transportation time. Several fjord crossing concepts have been developed, including a suspension bridge crossing over Langenuen. This thesis will look into the feasibility of designing the bridge using aluminium as the girder material. Buffeting response analysis and flutter calculations will be conducted in order to confirm whether an aluminium girder is feasible or not for this bridge crossing. In addition to investigating the dynamic stability of this bridge, other aspects must also be taken into account, e.g problems regarding fatigue and thermal expansion.

The thesis starts with an introduction to the theory applied in the calculations. This includes fundamental relations in structural dynamics, aerodynamic derivatives, buffeting response and motion induced instabilities. The next chapter, chapter 3, introduces the preliminary design of the Langenuen bridge, and defines the values for static force coefficients and aerodynamic derivatives that are used in the rest of the thesis. A description of the structure and ideas behind the parametric script are presented in chapter 4. Chapter 5 follows with a brief overview of the matlab scripts that has been used for result calculations. These results are presented and discussed in chapter 6. In chapter 7 a further discussion will take place, where sources of uncertainties, concerns to the parametric modelling and other aspects will be presented. Finally, concluding remarks and proposed subjects to further work are presented in chapter 8.

2 Theory

Most of the theory presented here is taken from the book *Theory of Bridge Aerodynamics* [2].

2.1 Fundamental relations

The still-air dynamic behaviour of a structure with n_{DOF} degrees of freedom can be described by the equation of motion

$$\mathbf{M}\ddot{\mathbf{r}}(t) + \mathbf{C}\dot{\mathbf{r}}(t) + \mathbf{K}\mathbf{r}(t) = \mathbf{Q}_{tot}(t) \tag{2.1}$$

where **M**, **C** and **K** are the mass, damping and stiffness matrices, and vector $\mathbf{Q}_{tot}(t)$ contains the external loads acting on the system. Vectors **r**, $\dot{\mathbf{r}}$ and $\ddot{\mathbf{r}}$ represent the displacement, velocity and acceleration of the system, respectively. When disregarding damping and external loads, the undamped eigenvalue problem is obtained

$$(\mathbf{K} - \omega_n^2\mathbf{M})\mathbf{\Phi} = 0 \tag{2.2}$$

where ω_n represents the eigenfrequencies, and $\mathbf{\Phi}$ contains the corresponding eigenmodes. The number of eigenfrequencies are equal to n_{DOF}. The natural frequency ω_n for a SDOF system when disregarding damping is given by

$$\omega_n^2 = \frac{k}{m} \tag{2.3}$$

where k is the element stiffness and m the element mass. Another approach of expressing the dynamic behaviour is by the use of generalized coordinates. By premultiplying equation (2.1) with $\mathbf{\Phi^T}$ and introducing $\mathbf{r}(t) = \mathbf{\Phi}\boldsymbol{\eta}(t)$, the system can be written on generalized form as

$$\tilde{\mathbf{M}}\ddot{\boldsymbol{\eta}}(t) + \tilde{\mathbf{C}}\dot{\boldsymbol{\eta}}(t) + \tilde{\mathbf{K}}\boldsymbol{\eta}(t) = \tilde{\mathbf{Q}}_{tot}(t) \tag{2.4}$$

where $\boldsymbol{\eta}(t)$ are generalized coordinates, and $\mathbf{\Phi}$ are space dependent mode shapes. $\tilde{\mathbf{M}}$, $\tilde{\mathbf{C}}$ and $\tilde{\mathbf{K}}$ are the modal mass, damping and stiffness matrices of the system. They are given by

$$\begin{aligned} \tilde{\mathbf{M}} &= diag[\tilde{M}_i] & \tilde{M}_i &= \textstyle\int_L \left(\boldsymbol{\phi}_i^T \cdot \mathbf{M} \cdot \boldsymbol{\phi}_i\right) dx \\ \tilde{\mathbf{C}} &= diag[\tilde{C}_i] & \tilde{C}_i &= 2\tilde{M}_i\omega_i\zeta_i \\ \tilde{\mathbf{K}} &= diag[\tilde{K}_i] & \tilde{K}_i &= \omega_i^2\tilde{M}_i \end{aligned} \tag{2.5}$$

The modal load vector $\tilde{\boldsymbol{Q}}_{tot}(t)$ is defined by

$$\tilde{\boldsymbol{Q}}_{tot}(t) = \int_L \left(\boldsymbol{\phi}_i^T \cdot \boldsymbol{q}_{tot}\right) dx \tag{2.6}$$

where $\mathbf{q}_{tot}$ is the cross sectional load vector containing the total load per unit length. It is given by

$$\mathbf{q}_{tot} = \begin{bmatrix} q_y & q_z & q_\theta \end{bmatrix}_{tot}^T \tag{2.7}$$

2.2 Wind and motion induced loads

Tall buildings and suspension bridges are examples of slender structures that are sensitive to dynamic wind loading. Different wind velocities result in various types of wind forces, each resulting in a unique response behaviour of the structure. The wind loading is typically categorized into four force contributions

1. Static wind forces due to a mean wind velocity
2. Fluctuating wind forces due to vortex shedding
3. Fluctuating wind forces due to buffeting
4. Motion induced forces as a result of interaction between the wind and system motion

A graphical representation is given in figure 2. No load effects occur exclusively at a given wind velocity range, but each will dominate the response as it appears from the figure. Vortex shedding usually occurs at low wind velocities, and will not be investigated any further in this thesis.

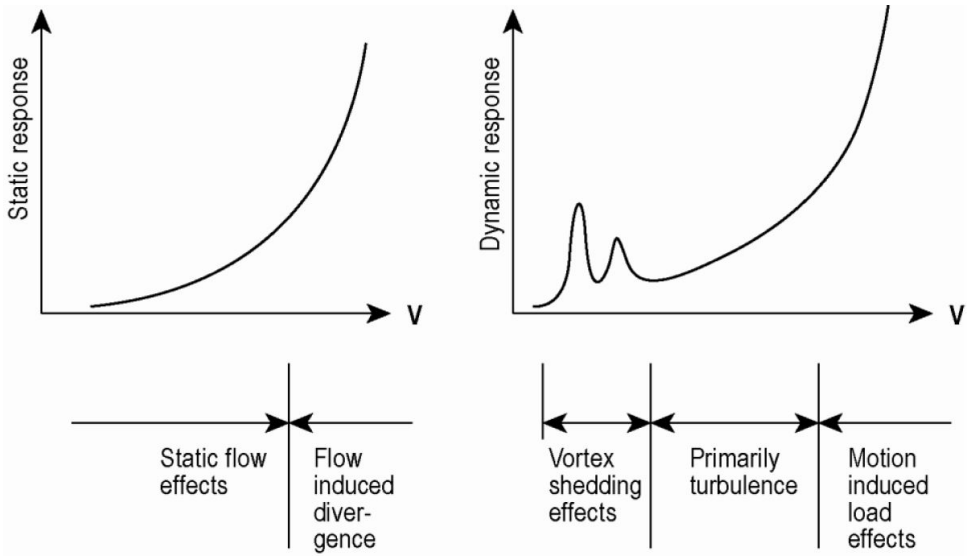


Figure 2: Response for various wind velocities [2].

The theory presented in the following applies to structures that are line-like. For this assumption it applies that the wind load acts as a concentrated force in a reference point on the section of the structure being considered, typically in the shear center [3].

A bridge deck is considered. The total wind load per unit length is the sum of all load contributions acting on the girder

$$q_{tot} = q(x,t) + q_{ae}(x,t,r,\dot{r},\ddot{r}) \tag{2.8}$$

here, $q(x,t)$ are the flow induced forces due to the instantaneous wind velocity pressure at a point, and q_{ae} denotes the motion induced forces.

In order to establish the buffeting wind load, some assumptions are necessary.

- The wind field is homogeneous and stationary
- The distance from the ground to the bridge deck is large and constant
- The wind flow in the longitudinal direction of the structure is of small importance for the response

The wind load can then be established from the mean wind velocity V and the fluctuating wind components in y and z-direction, the latter given by

$$\mathbf{v}(x,t) = \begin{bmatrix} u & w \end{bmatrix}^T \tag{2.9}$$

The instantaneous wind velocity pressure is given by Bernoulli's equation

$$q_U(t) = \frac{1}{2}\rho[U(t)]^2 \tag{2.10}$$

where ρ is the density of the air.

2.2.1 Buffeting theory

The cross sectional forces acting on a bridge deck due to wind velocities are categorized into drag, lift, and moment forces, as can be shown in the equation below

$$\begin{bmatrix} q_D(x,t) \\ q_L(x,t) \\ q_M(x,t) \end{bmatrix} = \frac{1}{2}\rho V_{rel}^2 \begin{bmatrix} D \cdot C_D(\alpha) \\ D \cdot C_L(\alpha) \\ B^2 \cdot C_M(\alpha) \end{bmatrix} \tag{2.11}$$

where B and D is the width and depth of the cross section, V_{rel} is the relative wind velocity, and the coefficients C_D, C_L and C_M are load coefficients dependent on the angle of attack α of the wind velocity. The relative wind velocity vector V_{rel} consists of the static wind velocity V, the

turbulent wind velocity components u and w, and the structural velocities $\dot{r}_y$ and $\dot{r}_z$

$$V_{rel}^2 = (V + u - \dot{r}_y)^2 + (w - \dot{r}_z)^2 \tag{2.12}$$

Figure 3 shows the quantities of the drag, lift and moment forces acting on a bridge deck

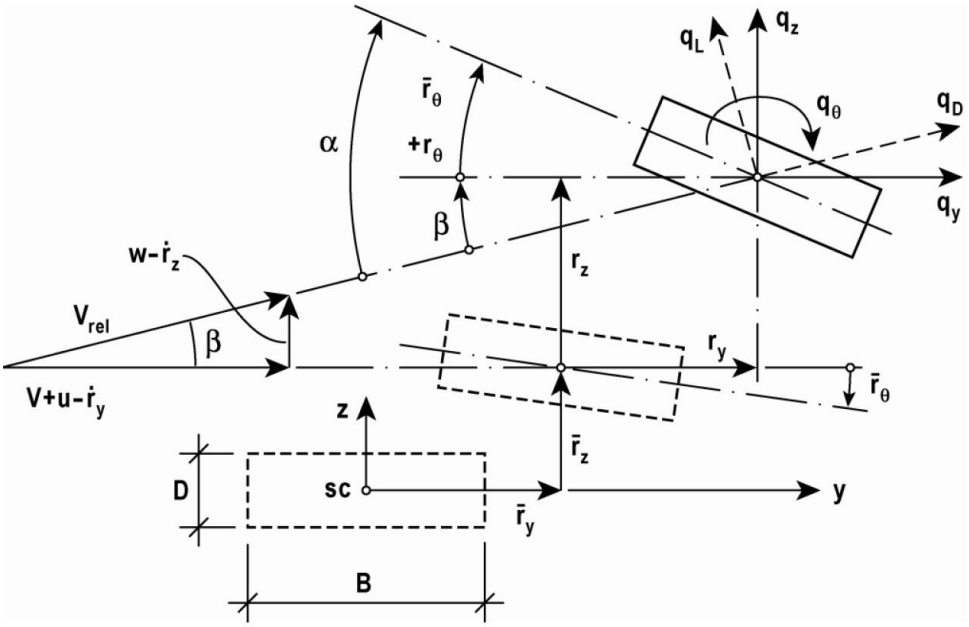


Figure 3: Cross sectional forces acting on a member [2].

The forces defined in equation (2.11) are given in the coordinate system of the relative wind velocity, and can be transformed into the coordinate system of the member being considered

$$\begin{bmatrix} q_y(x,t) \\ q_z(x,t) \\ q_\theta(x,t) \end{bmatrix} = \begin{bmatrix} cos(\beta) & -sin(\beta) & 0 \\ sin(\beta) & cos(\beta) & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} q_D(x,t) \\ q_L(x,t) \\ q_M(x,t) \end{bmatrix} \tag{2.13}$$

where β is the relative angle of attack

$$\beta = tan^{-1}\Big(\frac{w - \dot{r}_z}{V + u - \dot{r}_y}\Big) \tag{2.14}$$

In order to take advantage of the linear random vibration framework, two linearizations will be made for the buffeting load. The first linearization involves the wind velocity and structural displacements. It is assumed that the mean wind velocity is significantly larger than both the turbulent wind components and the horizontal velocity of the structure, i.e. $V \gg u$, $V \gg w$ and $V \gg \dot{r}_y$. Also, it is assumed that the relative angle of attack is small, giving tan $\beta \approx \beta$. The

first assumption ensures that higher order terms of the relative wind velocity can be neglected

$$V_{rel}^2 = (V + u - \dot{r}_y)^2 + (w - \dot{r}_z)^2 \approx V^2 + 2Vu - 2V\dot{r}_y \tag{2.15}$$

Further on, the fluctuating part of the angle of attack α is simplified to

$$\alpha = \bar{r}_\theta + r_\theta + \beta \approx \bar{r}_\theta + r_\theta + \frac{w}{V} - \frac{\dot{r}_z}{V} \tag{2.16}$$

The second linearization is that the quasi-static load coefficients can be approximated in a linearized manner

$$\begin{bmatrix} C_D(\alpha) \\ C_L(\alpha) \\ C_M(\alpha) \end{bmatrix} = \begin{bmatrix} \bar{C}_D \\ \bar{C}_L \\ \bar{C}_M \end{bmatrix} + \alpha_f \begin{bmatrix} C'_D \\ C'_L \\ C'_M \end{bmatrix} \tag{2.17}$$

here $\bar{C}_D$, $\bar{C}_L$ and $\bar{C}_M$ are the load coefficients evaluated at a mean angle of attack $\bar{\alpha}$, and C'_D, C'_L and C'_M are the derivatives of the static load coefficients evaluated at $\bar{\alpha}$. The fluctuating part of the angle of attack is denoted α_f. The linearization is illustrated in figure 8.

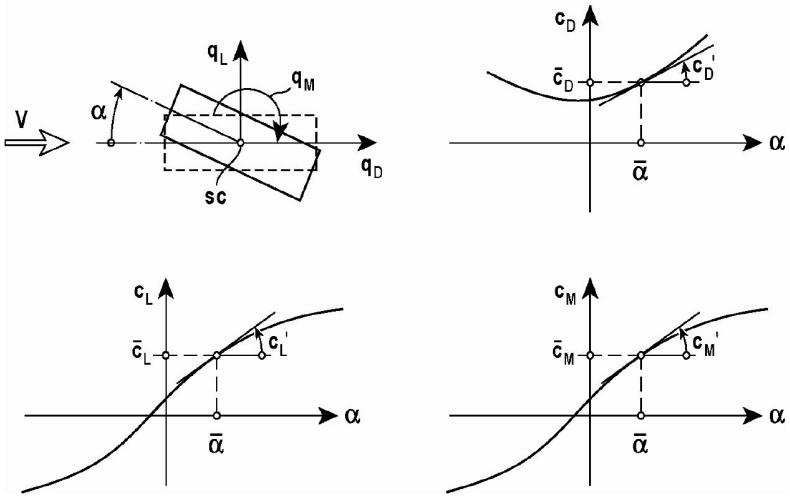


Figure 4: Linearization of the static load coefficients [2].

The two linearization assumptions lead to the following expression for the total wind load

$$\mathbf{q}_{tot}(x,t) = \bar{\mathbf{q}}(x) + \mathbf{B}_q(x) \cdot \mathbf{v}(x,t) + \mathbf{C}_{ae}(x) \cdot \dot{\mathbf{r}}(x,t) + \mathbf{K}_{ae}(x) \cdot \mathbf{r}(x,t) \tag{2.18}$$

where

$$\mathbf{r}(x,t) = \begin{bmatrix} r_y & r_z & r_\theta \end{bmatrix}^T \tag{2.19}$$

$$\bar{\mathbf{q}}(x) = \begin{bmatrix} \bar{q}_y \\ \bar{q}_z \\ \bar{q}_\theta \end{bmatrix} = \frac{\rho V^2 B}{2} \begin{bmatrix} (D/B)\bar{C}_D \\ \bar{C}_L \\ B\bar{C}_M \end{bmatrix} \tag{2.20}$$

$$\mathbf{B}_q(x) = \frac{\rho V B}{2} \begin{bmatrix} 2(D/B)\bar{C}_D & ((D/B)C'_D - \bar{C}_L) \\ 2\bar{C}_L & (C'_L + (D/B)\bar{C}_D) \\ 2B\bar{C}_M & BC'_M \end{bmatrix} \tag{2.21}$$

$$\mathbf{C}_{ae} = -\frac{\rho V B}{2} \begin{bmatrix} 2(D/B)\bar{C}_D & ((D/B)C'_D - \bar{C}_L) & 0 \\ 2\bar{C}_L & (C'_L + (D/B)\bar{C}_D) & 0 \\ 2B\bar{C}_M & BC'_M & 0 \end{bmatrix} \tag{2.22}$$

$$\mathbf{K}_{ae} = \frac{\rho V^2 B}{2} \begin{bmatrix} 0 & 0 & (D/B)C'_D \\ 0 & 0 & C'_L \\ 0 & 0 & BC'_M \end{bmatrix} \tag{2.23}$$

The terms in equation (2.18) represent a static load due to a mean wind velocity, a buffeting load due to turbulent wind velocity components, and a self-excited load due to interaction between the wind flow and the motion of the structure.

By considering only the time varying fluctuating parts, equation (2.18) is reduced to

$$\mathbf{q}(x,t) = \mathbf{B}_q \cdot \mathbf{v} + \mathbf{C}_{ae} \cdot \dot{\mathbf{r}} + \mathbf{K}_{ae} \cdot \mathbf{r} \tag{2.24}$$

The Fourier transform of equation (2.24) is given by

$$\mathbf{a}_q = \mathbf{B}_q \cdot \mathbf{a}_v + \left(iw\mathbf{C}_{ae} + \mathbf{K}_{ae}\right) \cdot \mathbf{a}_r \tag{2.25}$$

where

$$\begin{aligned} \mathbf{a}_q(x,\omega) &= \begin{bmatrix} a_{q_y} & a_{q_z} & a_{q_\theta} \end{bmatrix}^T \\ \mathbf{a}_r(x,\omega) &= \begin{bmatrix} a_{r_y} & a_{r_z} & a_{r_\theta} \end{bmatrix}^T \\ \mathbf{a}_v(x,\omega) &= \begin{bmatrix} a_u & a_w \end{bmatrix}^T \end{aligned} \tag{2.26}$$

The frequency domain version of equation (2.21) is obtained by introducing the frequency

dependent admittance functions

$$\mathbf{B}_q(x,\omega) = \frac{\rho V B}{2}\begin{bmatrix} 2(D/B)\bar{C}_D A_{yu} & ((D/B)C'_D - \bar{C}_L)A_{yw} \\ 2\bar{C}_L A_{zu} & (C'_L + (D/B)\bar{C}_D)A_{zw} \\ 2B\bar{C}_M A_{\theta u} & BC'_M A_{\theta w} \end{bmatrix} \tag{2.27}$$

An approximate expression is suggested as

$$A_{mn}(\omega) = \frac{1}{\left(1 + a_{mn}B\omega/V\right)^{b_{mn}}}, \quad m = y, z, \theta, \quad n = u, w \tag{2.28}$$

where a_{mn} and b_{mn} are constants depending on the cross section of the member being considered. Equation (2.28) equals 1 for ω equal to zero, and approaches zero for high frequencies. A_{mn} is conservatively set to 1.
Including the structural aerodynamic matrices, the modified EOM is given by

$$\tilde{\mathbf{M}}\ddot{\boldsymbol{\eta}}(t) + (\tilde{\mathbf{C}} - \tilde{\mathbf{C}}_{ae})\dot{\boldsymbol{\eta}}(t) + (\tilde{\mathbf{K}} - \tilde{\mathbf{K}}_{ae})\boldsymbol{\eta}(t) = \tilde{\mathbf{Q}}(t) \tag{2.29}$$

where $\tilde{\mathbf{C}}_{ae}$ and $\tilde{\mathbf{K}}_{ae}$ are the generalized aerodynamic damping and stiffness matrices, given by

$$\tilde{\mathbf{K}}_{ae} = \begin{bmatrix} \ddots & & \\ & \tilde{K}_{ae_{ij}} & \\ & & \ddots \end{bmatrix} \quad \text{and} \quad \tilde{\mathbf{C}}_{ae} = \begin{bmatrix} \ddots & & \\ & \tilde{C}_{ae_{ij}} & \\ & & \ddots \end{bmatrix} \tag{2.30}$$

where

$$\begin{bmatrix} \tilde{K}_{ae_{ij}} \\ \tilde{C}_{ae_{ij}} \end{bmatrix} = \int_L \begin{bmatrix} \boldsymbol{\phi}_i^T \cdot \mathbf{K}_{ae} \cdot \boldsymbol{\phi}_j \\ \boldsymbol{\phi}_i^T \cdot \mathbf{C}_{ae} \cdot \boldsymbol{\phi}_j \end{bmatrix} dx \tag{2.31}$$

2.2.2 Aerodynamic derivatives

In the 1920's and 1930's Theodorsen developed theoretical expressions for a thin airfoil subjected to wind [4]. It showed that the interaction between the wind flow and the structure caused changes in the properties of the system. The theory was further developed by Scanlan & Tomko in 1971 using a flat plate such that it would be applicable to bridge engineering [5]. Following their notation, the structural stiffness and damping matrices expressed in the frequency domain, expressed with the aerodynamic derivatives, are given as follows

$$\mathbf{C}_{ae} = \frac{\rho B^2}{2}\omega_i(V)\begin{bmatrix} P_1^* & P_5^* & BP_2^* \\ H_5^* & H_1^* & BH_2^* \\ BA_5^* & BA_1^* & B^2A_2^* \end{bmatrix} \tag{2.32}$$

$$\mathbf{K}_{ae} = \frac{\rho B^2}{2}\left[\omega_i(V)\right]^2 \begin{bmatrix} P_4^* & P_6^* & BP_3^* \\ H_6^* & H_4^* & BH_3^* \\ BA_6^* & BA_4^* & B^2A_3^* \end{bmatrix} \tag{2.33}$$

where P_i^*, H_i^* and A_i^*, i = 1 - 6, are the non-dimensional aerodynamic derivatives. They depend on the frequency of motion and the mean wind velocity, and are expressed as functions of the reduced velocity V_{red}

$$V_{red} = \frac{V}{B\omega_i(V)} \tag{2.34}$$

The relation between the notation of Scanlan & Tomko and the quasi-static load coefficients, as defined in the previous section, is presented below

$$\begin{bmatrix} P_1^* & H_1^* & A_1^* \\ P_2^* & H_2^* & A_2^* \\ P_3^* & H_3^* & A_3^* \\ P_4^* & H_4^* & A_4^* \\ P_5^* & H_5^* & A_5^* \\ P_6^* & H_6^* & A_6^* \end{bmatrix} = \begin{bmatrix} -2\bar{C}_D\frac{D}{B}\frac{V}{B\omega_i(V)} & -(C'_L + \bar{C}_D\frac{D}{B})\frac{V}{B\omega_i(V)} & -C'_M\frac{V}{B\omega_i(V)} \\ 0 & 0 & 0 \\ C'_D\frac{D}{B}(\frac{V}{B\omega_i(V)})^2 & C'_L(\frac{V}{B\omega_i(V)})^2 & C'_M(\frac{V}{B\omega_i(V)})^2 \\ 0 & 0 & 0 \\ (\bar{C}_L - C'_D\frac{D}{B})\frac{V}{B\omega_i(V)} & -2\bar{C}_L\frac{V}{B\omega_i(V)} & -2\bar{C}_M\frac{V}{B\omega_i(V)} \\ 0 & 0 & 0 \end{bmatrix} \tag{2.35}$$

In total there are 18 aerodynamic derivatives. The coefficients related to the equilibrium condition in horizontal, vertical and torsional direction are denoted P_i^*, H_i^* and A_i^*, respectively.

The ADs can be experimentally estimated through wind tunnel testing. However, due to the setup of a wind tunnel, extracting the coefficients related to lateral motion is either challenging or cannot be done. These coefficients are often of less importance. A typical solution is to either disregard these coefficients in the analysis, or use the quasi-static coefficients. Another approach for determining ADs is by the use of numerical simulation, e.g. CFD.

Theodorsen's aerodynamic derivatives for an ideal flat plate are given by

$$\begin{bmatrix} H_1^* & A_1^* \\ H_2^* & A_2^* \\ H_3^* & A_3^* \\ H_4^* & A_4^* \end{bmatrix} = \begin{bmatrix} -2\pi F V_{red} & -\frac{\pi}{2}FV_{red} \\ \frac{\pi}{2}\big(1 + F + 4GV_{red}\big)V_{red} & -\frac{\pi}{8}\big(1 - F - 4GV_{red}\big)V_{red} \\ 2\pi\big(FV_{red} - G/4\big)V_{red} & \frac{\pi}{2}\big(FV_{red} - G/4\big)V_{red} \\ \frac{\pi}{2}\big(1 + 4GV_{red}\big) & \frac{\pi}{2}GV_{red} \end{bmatrix} \tag{2.36}$$

F and G are the real and imaginary parts of Theodorsen's circulatory function. They are given by

$$F\left(\frac{\hat{\omega}_i}{2}\right) = \frac{J_1 \cdot (J_1 + Y_0) + Y_1 \cdot (Y_1 - J_0)}{(J_1 + Y_0)^2 + (Y_1 - J_0)^2} \tag{2.37}$$

$$G\left(\frac{\hat{\omega}_i}{2}\right) = -\frac{J_1 \cdot J_0 + Y_1 \cdot Y_0}{(J_1 + Y_0)^2 + (Y_1 - J_0)^2} \tag{2.38}$$

where J_n and Y_n are the first and second kinds of Bessel functions.

2.3 Buffeting response

The cross-spectral density matrix of the buffeting wind load $\mathbf{q}_b = \mathbf{B_q v}$ is defined as

$$\mathrm{S}_{\tilde{\mathbf{Q}}_b}(\omega) = \int_0^L \int_0^L \mathbf{\Phi}^T(x_1)\mathbf{B_q}(x_1)\mathbf{S_v}(\omega, \Delta x)\mathbf{B}_\mathbf{q}^T(x_2)\mathbf{\Phi}(x_2)dx_1dx_2 \tag{2.39}$$

where $\mathbf{S_v}$ is the cross-spectral density of the turbulence components of the wind velocity, given by

$$\mathbf{S_v} = \begin{bmatrix} S_{uu} & S_{wu} \\ S_{uw} & S_{ww} \end{bmatrix} \tag{2.40}$$

From The Norwegian Public Roads Administration's *Håndbok N400*, wind measurements should be performed at the bridge site if the bridge is in wind load class 3 and the main span is over 300 meters [6]. For this thesis wind measurements are not available, instead the wind spectra will be obtained by using the Kaimal spectrum in *Håndbok N400*. The single point auto-spectral density is defined as

$$S_i(\omega) = \frac{1}{2\pi}\frac{\sigma_i^2 \quad {}^xL_i(z)}{V}\frac{A_i}{(1+1.5A_i\hat{\omega}_i)^{5/3}}, \quad i = u, v, w \tag{2.41}$$

where ${}^xL_i$ is the integral length scale, A_i are factors found in the handbook, and $\hat{\omega}_i$ is given by

$$\hat{\omega}_i = \frac{\omega \, {}^xL_i(z)}{2\pi V}, \quad i = u, v, w \tag{2.42}$$

and the elements of the cross-spectral density matrix from equation (2.40) are given by

$$S_{i_1 i_2}(\Delta x, \omega) = \sqrt{S_{i_1}(\omega)S_{i_2}(\omega)} \cdot exp\big(-C_{ij}\frac{\omega|\Delta x|}{2\pi V}\big), \quad i = u, v, w, \quad j = y, z \tag{2.43}$$

The v component of the turbulent wind is disregarded as earlier mentioned, and the coefficients are given below.

Table 1: Coefficients from N400

C_{uy}	C_{uz}	C_{wy}	C_{wz}	A_u	A_w
10	10	6.5	3	6.8	9.4

The cross-spectral density of the modal response is

$$\mathbf{S}_{\boldsymbol{\eta}}(\omega) = \tilde{\mathbf{H}}^*(\omega)\mathbf{S}_{\tilde{\mathbf{Q}}_b}(\omega)\tilde{\mathbf{H}}^T(\omega) \tag{2.44}$$

where $\tilde{\mathbf{H}}$ is the FRF, expressed as

$$\tilde{\mathbf{H}}(\omega) = \left[-\omega^2\tilde{\mathbf{M}} + i\omega\big(\tilde{\mathbf{C}} - \tilde{\mathbf{C}}_{ae}\big) + \big(\tilde{\mathbf{K}} - \tilde{\mathbf{K}}_{ae}\big)\right]^{-1} \tag{2.45}$$

$\tilde{\mathbf{H}}^*$ is the complex conjugate. The cross-spectral density of the physical response is

$$\mathbf{S}_r(\omega) = \mathbf{\Phi}\mathbf{S}_{\boldsymbol{\eta}}\mathbf{\Phi}^T \tag{2.46}$$

The correlation coefficient matrix of the buffeting response in a given point x_r is given by

$$\boldsymbol{\rho}_{rr}(x_r) = \frac{\mathbf{Cov}_{rr}(x_r)}{\sigma_{r_i} \cdot \sigma_{r_j}} \tag{2.47}$$

where $\mathbf{Cov}_{rr}(x_r)$ is the covariance matrix. It contains the variances for each DOF and the covariances between these.

$$\mathbf{Cov}_{rr}(x_r) = \int_0^\infty \mathbf{S}_r(\omega)d\omega = \begin{bmatrix} \sigma^2_{r_y r_y} & Cov_{r_y r_z} & Cov_{r_y r_\theta} \\ Cov_{r_z r_y} & \sigma^2_{r_z r_z} & Cov_{r_z r_\theta} \\ Cov_{r_\theta r_y} & Cov_{r_\theta r_z} & \sigma^2_{r_\theta r_\theta} \end{bmatrix} \tag{2.48}$$

2.4 Motion induced instabilities

A motion induced instability is triggered when either the stiffness or damping of a structure is analytically equal to zero. It is caused by a critical wind velocity resulting in a rapid change in the structural response, potentially leading to construction failure. It is common to differentiate between four instability phenomena, based on their nature and the type of displacements that occur:

- Static divergence
- Galloping
- Instability in pure torsion
- Flutter

The critical instability phenomenon is the one that gives the lowest critical wind velocity V_{CR}. For suspension bridges this is usually flutter. In general, any instability limit can be found by examining the properties of the impedance matrix. It is defined as

$$\hat{\mathbf{E}}_\eta(\omega, V) = \mathbf{I} - \boldsymbol{\kappa}_{ae} - \left(\omega \cdot diag\left[\frac{1}{\omega_i}\right]\right)^2 + 2i\omega \cdot diag\left[\frac{1}{\omega_i}\right] \cdot (\boldsymbol{\zeta} - \boldsymbol{\zeta}_{ae}) \tag{2.49}$$

where $\mathbf{I}$ is the identity matrix, $\boldsymbol{\kappa}_{ae}$ is the aerodynamic stiffness matrix divided by the structural stiffness matrix, and $\boldsymbol{\zeta}_{ae}$ is the aerodynamic damping matrix. The components of the last two are given by

$$\kappa_{ae_{ij}} = \frac{\tilde{K}_{ae_{ij}}}{\omega_i^2 \tilde{M}_i} = \frac{\rho B^2}{2\tilde{m}_i} \cdot \left[\frac{\omega_i(V)}{\omega_i}\right]^2 \cdot \frac{\int_L (\phi_i^T \tilde{\boldsymbol{K}}_{ae} \phi_j) dx}{\int_L (\phi_i^T \phi_i) dx} \tag{2.50}$$

$$\zeta_{ae_{ij}} = \frac{\tilde{C}_{ae_{ij}}}{2\omega_i \tilde{M}_i} = \frac{\rho B^2}{4\tilde{m}_i} \cdot \frac{\omega_i(V)}{\omega_i} \cdot \frac{\int_L (\phi_i^T \tilde{\boldsymbol{C}}_{ae} \phi_j) dx}{\int_L (\phi_i^T \phi_i) dx} \tag{2.51}$$

where $\omega_i(V)$ is the resonance frequency associated with mode i as a function of the mean wind velocity, and ω_i is the corresponding still air frequency for mode i. The response of a structure will diverge if the wind velocity reaches V_{cr}. The critical wind velocity can be found by setting the absolute value of the determinant of the impedance matrix equal to zero.

$$\left| det(\hat{\boldsymbol{E}}_\eta(\omega, V)) \right| = 0 \tag{2.52}$$

However, as the impedance matrix consists of both real and complex values, the following

conditions must be fulfilled simultaneously

$$Re(det(\hat{\boldsymbol{E}}_\eta)) = 0 \tag{2.53}$$

$$Im(det(\hat{\boldsymbol{E}}_\eta)) = 0 \tag{2.54}$$

Both $\boldsymbol{\kappa}_{ae}$ and $\boldsymbol{\zeta}_{ae}$ are functions of $\omega_i(V)$, which implies that the solution cannot be obtained analytically. Instead, an iterative process is necessary.

2.4.1 Flutter

Flutter is a dynamic instability phenomenon that occurs at high wind velocities when coupling effects between two or more vibration modes results in an uncontrollable motion of the bridge [7]. It is due to a negative system damping, i.e. when the amount of energy from motion induced wind forces exceeds the amount of energy dissipated due to structural damping. For wind velocities above the critical limit the bridge is unstable, and may experience excessive amplitude growth and violent vibrations.

2.4.2 Bimodal flutter

A bimodal approach is based on the assumption that the flutter limit can be determined by considering two vibration modes, each restricted to motion in one direction. This is usually the lowest torsional mode and the lowest shape-wise similar vertical mode. The solution to equations (2.53) and (2.54) is greatly simplified, and it is possible to calculate the flutter limit directly. However, in this thesis only multimodal flutter calculations will be conducted.

2.4.3 Multimodal flutter

The requirement of motion in only one direction for each mode is no longer applicable for multimodal flutter. Instead, each mode shape may contain components in both y, z and θ direction. The matrices in equation (2.49) has dimensions $\mathrm{n_{modes}}$ by $\mathrm{n_{modes}}$. The eigenvalue problem can be solved in the frequency domain by the following equation

$$\left[\tilde{\mathbf{M}}S^2 + \left(\tilde{\mathbf{C}} - \tilde{\mathbf{C}}_{ae}(\omega, V)\right) S + \left(\tilde{\mathbf{K}} - \tilde{\mathbf{K}}_{ae}(\omega, V)\right)\right] \psi = 0 \tag{2.55}$$

where the eigenvalue for each mode is given by

$$S_i = -\zeta_i\omega_i \pm i\omega_i\sqrt{1-\zeta_i} \tag{2.56}$$

and ψ is the modal eigenvector. From equation (2.56), it is clear that the real and imaginary parts are

$$Re(S_i) = -\zeta_i \omega_i \tag{2.57}$$

$$Im(S_i) = \omega_i \sqrt{1 - \zeta_i^2} \tag{2.58}$$

where ω_i is the eigenfrequency of each mode, and ζ_i is the corresponding damping ratio. Equation (2.57) corresponds to the system damping. It implies that the system becomes unstable when the damping ratio is negative.

3 Preliminary design of Langenuen fjord crossing

The crossing of Langenuen is one of several fjord crossings included in the planning of a ferry free highway route between Kristiansand and Trondheim. The project is still in its conceptual stage, i.e. only preliminary design has been done. Geometric values in a report provided by Norconsult is used as a basis for the initial design of the Langenuen bridge [1].

Figure 5: Illustration of a possible fjord crossing over Langenuen [1].

There are three proposed locations of the bridge with varying span lengths. The final location is not decided, so in order to perform an analysis location A is chosen.

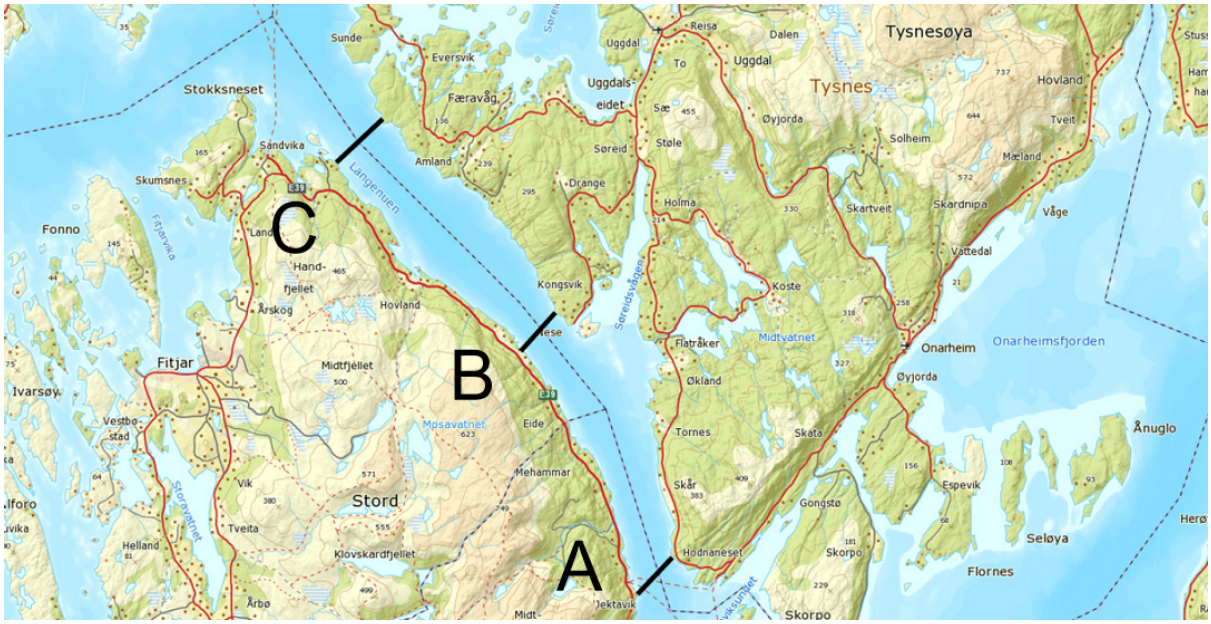


Figure 6: Possible bridge locations [1].

3.1 General arrangement

The main dimensions for the bridge at location A are given in figure 7.

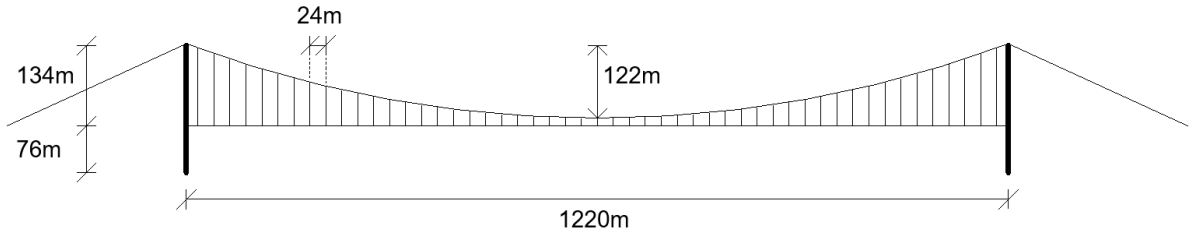


Figure 7: Dimensions of the Langenuen bridge at location A

In the following an overview of the bridge components used for the modelling will be introduced, i.e. parts that are assumed of main importance with regard to mass and stiffness contributions.

3.1.1 Pylons

The planned bridge will have two fixed concrete pylons anchored to the ground at each side of the fjord. Each pylon has an A-shaped form with a rigel supporting the bridge deck. A hollow rectangular cross section is proposed for the components.

Figure 8: Illustration of a suggested pylon design [1].

3.1.2 Girder

The Norwegian Public Roads Administration recommends that the road is designed with four lanes with a speed limit of 110km/h, according to road class H3 [8]. Road shoulders and additional width requirements due to inclined hangers as well as a pedestrian lane, yields a girder width of about 34 meters. The girder height is set to 4 meters. Bulkheads will be inserted at an interval of 4 meters. Figure 9 shows a proposed bridge deck.

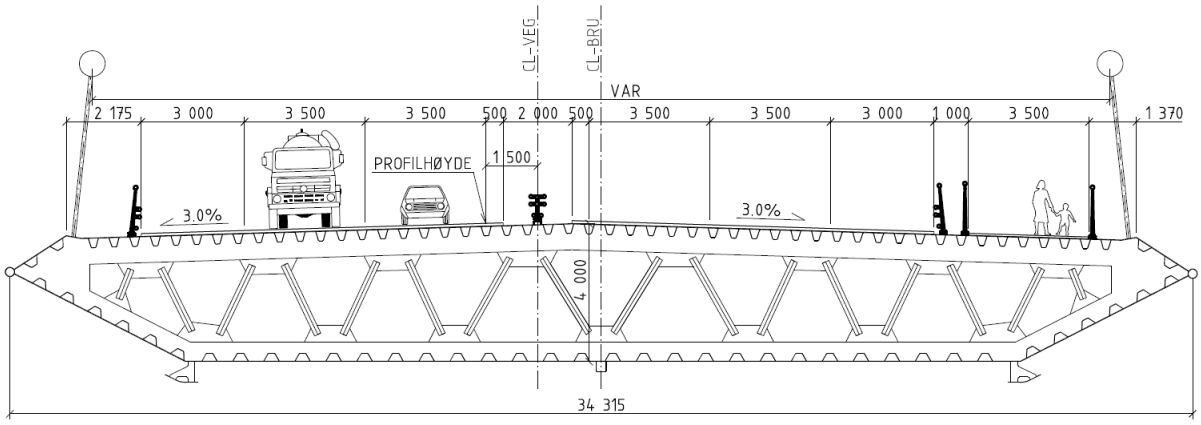


Figure 9: Proposed bridge deck [1].

The proposed bridge deck is made of steel. However, in this thesis the feasibility of an aluminium girder is of interest. For that reason the cross sectional parameters for the proposed bridge deck from the report will not be used except for the outer geometry, such as the girder width and height. Cross sectional parameters of a shape-wise equivalent aluminium girder are provided by Dr.techn. Olav Olsen, and will be used in the analyses.

3.1.3 Cables and hangers

The cables and hangers will be made of high tensile steel with a tensile strength of 1570 MPa. Static self weight will be considered when determining the dimensions. The utilization of these are low enough to allow for dynamic loads in addition to the static. As the girder will be modelled using aluminium, that has a lower weight than steel, a reduction of the diameter of the cables and hangers is expected. The hangers will be placed every 24 meters along the girder length. Due to the A-shaped pylons the hangers will be inclined, but this is not regarded as a problem. However, it could cause coupling between horizontal and torsional modes.

3.2 Static force coefficients

Wind tunnel testing of a possible cross section of Langenuen bridge has not been done. However, the geometry of the cross section of the Great Belt East Bridge is approximately similar to the bridge deck in figure 9, and will therefore be used for establishing both the static load coefficients and the aerodynamic derivatives. Found in the literature, the former has been estimated by wind tunnel testing and CFD [9] by looking at the flow field at various angles of attack. It can be shown from figure 10 that the values obtained by CFD are in good agreement with the experimental values from wind tunnel testing.

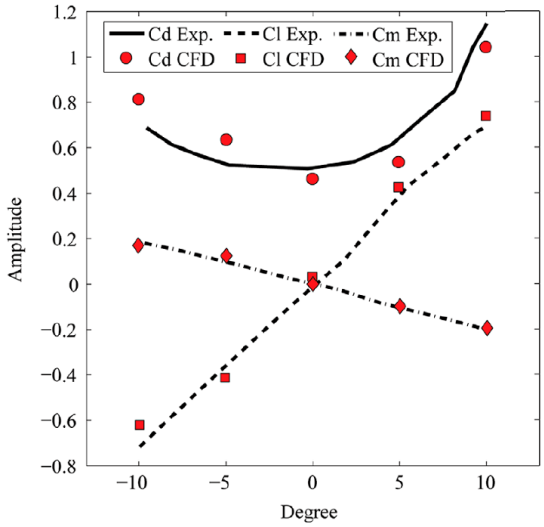


Figure 10: Plot of computed static aerodynamic coefficients at various angles of attack [9]

The static load coefficients used in this thesis are obtained by studying figure 10 for $\alpha = 0$. The values are tabulated below

Table 2: Static aerodynamic coefficients for $\alpha = 0$

$\bar{C}_D$	C'_D	$\bar{C}_L$	C'_L	$\bar{C}_M$	C'_M
0.5	0	0	4.4	0	-1.15

3.3 Aerodynamic derivatives

The aerodynamic derivatives are obtained from wind tunnel testing of the cross section of the Great Belt East Bridge [10]. They are tabulated in table 3.

Table 3: Provided aerodynamic derivatives from the Great Belt East Bridge [10]

V_{red}	0.27	0.35	0.54	0.72	0.91	1.64	2.15	2.71
P*1	−0.03	−0.03	−0.05	−0.07	−0.08	−0.15	−0.20	−0.25
P*2	−0.02	−0.02	−0.03	−0.04	−0.05	−0.09	−0.12	−0.15
P*3	−0.01	−0.02	−0.04	−0.08	−0.12	−0.41	−0.70	−1.10
P*4	0	0	0	0	0	0	0	0
P*5	−0.02	−0.02	−0.03	−0.04	−0.05	−0.09	−0.12	−0.15
P*6	0	0	0	0	0	0	0	0
H*1	−0.46	−0.66	−1.76	−2.43	−3.24	−6.08	−8.71	−11.04
H*2	0.53	0.67	0.10	1.28	1.49	2.34	2.87	3.53
H*3	0.30	0.47	1.10	1.91	3.12	10.72	18.76	29.86
H*4	1.65	1.53	1.23	1.14	0.96	−0.29	−0.68	−2.41
H*5	0.11	0.14	0.22	0.30	0.37	0.68	0.89	1.12
H*6	0	0	0	0	0	0	0	0
A*1	−0.21	−0.25	−0.50	−0.66	−0.89	−1.66	−2.26	−2.95
A*2	−0.04	−0.05	−0.09	−0.13	−0.19	−0.40	−0.55	−0.77
A*3	0.13	0.18	0.35	0.58	0.88	2.85	5.04	7.84
A*4	0.07	0.05	−0.02	−0.03	−0.11	−0.45	−0.65	−0.69
A*5	−0.07	−0.08	−0.13	−0.17	−0.22	−0.39	−0.52	−0.65
A*6	0	0	0	0	0	0	0	0

4 Parametric finite element model

A finite element model of the Langenuen bridge is made in Abaqus using the Abaqus Scripting Reference Guide [11]. All commands for creating the model is written in a python script that is run directly in Abaqus. This way, modifying parameters and implementing new commands in the pre-processesing is fast and efficient. Furthermore, commands can be performed directly in the graphical user interface after the script is run. The entire python script can be found in appendix D. Figure 11 shows the final FE model as a result of the python script. In the following, the aspects to creating the model will be explained in more detail.

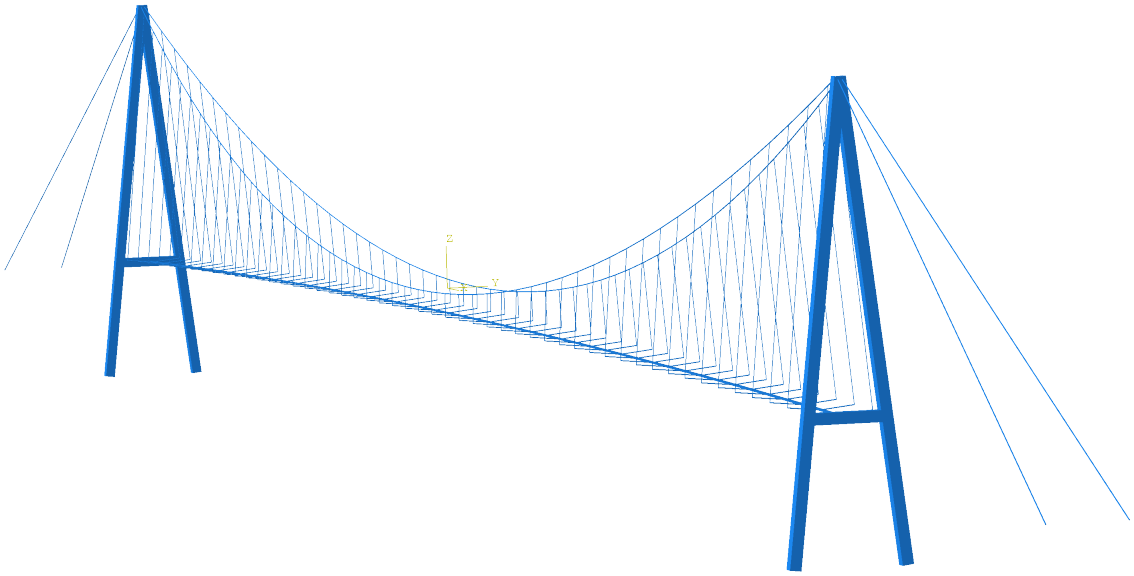


Figure 11: Finite element model of the Langenuen bridge

4.1 Parametric modelling

The python script is constructed to be parametric. Relevant geometric and material properties are defined initially as input values, followed by input dependent commands that generate the model automatically. The 3D geometry of the bridge is defined by mathematical polynomials, where each has the already mentioned initial input values as variables. Lists containing nodal coordinates in both x, y and z-direction are created subsequently based on the polynomial expressions, which then are input arguments in the creation of the model parts.

The bridge components are modelled as parts in Abaqus. All parts in the FE model are modelled using wire polylines. In addition to the pylons, cables, girder and hangers presented in the previous chapter, superficial connection elements will also be modelled. The reason for this is that only the pylons are assigned exact geometry. In the model the remaining parts are

assigned cross sectional properties. In order to recreate the correct geometry of the bridge, the connection elements are placed such that the distances between the other parts are maintained.

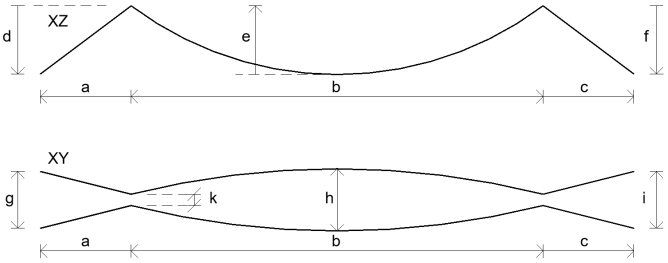
Figure 12 shows the geometric input in the python script. If a change of the geometry is to be made, it is easily done by changing the value of one or several of the variables as they are defined below.

```
# ------------------------------------------------------------------------------------------
# ----------------------------- INPUT VALUES -----------------------------------------------
# ------------------------------------------------------------------------------------------
# CABLE GEOMETRY [m]
a = 290.0                       # Horizontal distance between left tower and anchorage
b = 1220.0                      # Length of main span
c = 290.0                       # Horizontal distance between right tower and anchorage
d = a/1.8                       # Height between left tower and anchorage
e = 122.0                       # Height from tower to midspan of cables
f = c/1.8                       # Height between right tower and anchorage
g = 34.0                        # Width between cables at left anchorages
h = 29.5                        # Width between cables at midspan
i = 34.0                        # Width between cables at right anchorages
k = 1.6                         # Width between cables at tower tops
# TOWER GEOMETRY [m]
gap= 12.0                       # Vertical distance between cables and girder at the midspan
j= 70.0                         # Vertical distance between girder and bottom of towers
# GIRDER GEOMETRY [m]
vert_curv= 7.5                  # Height between lowest and highest parts of girder.
# NODE SPACING (x-dir) [m]
spacing_side_span = 25.0        # Horizontal spacing of cable nodes at outer part of towers
spacing_mid_span = 24.0         # Horizontal spacing of cable nodes(hangers) at the midspan
spacing_girder = 4.0            # Horizontal spacing of girder nodes
```

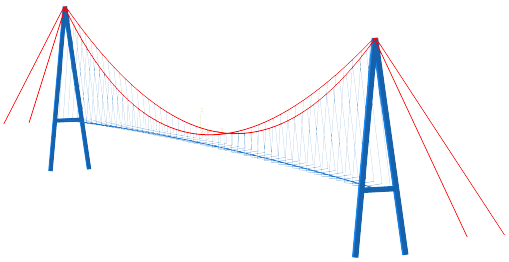
Figure 12: Geometric input in the python script

4.2 Part creation

The main cable system consists of two cables, as highlighted in figure 13(b). The geometry is defined by first and second order polynomials that entirely depend on the variable definitions as shown in figure 13(a). Origo is defined in the midpoint of the main span in for the x-axis, at the lowermost point of the cable for the z-axis. The arrows indicate how the geometry will be changed when changing a value of a variable. For instance will a change of variable *b* result in a new cable sag ratio *e*/*b*, but variables *a* and *c* will remain independent and therefore unchanged.



(a) Variable definitions in the xz and xy-plane, respectively



(b) Finite element model - main cables

Figure 13: Cable geometry

As earlier mentioned, all parts are modelled as wires in Abaqus. The command for creating the first cable is shown at line 312 in figure 14 below. Here, ”cablelist1” represents the list containing the nodal coordinates for one of the two cables. The same procedure yields for the other parts in the model.

```
309  # PART CREATION
310  bridge.Part(name='Cable1', dimensionality=THREE_D, type=DEFORMABLE_BODY)
311  cable1 = bridge.parts['Cable1']
312  cable1.WirePolyLine(points=cablelist1, mergeType=SEPARATE, meshable=ON)
```

Figure 14: Abaqus Scripting code for creating the wire for part ’cable1’

The geometry of the girder and the pylons is principally constructed the same way. The former is calculated from a second degree polynomial, while the latter is calculated from linear polynomials. The remaining parts, i.e. hangers and connection elements, are automatically generated after the geometry of the cables, pylons and girder is determined.

4.3 Section properties

The cross section of the pylons is modelled using the same box profile over the whole height. The width and height has dimensions 5x5 meters while the thickness is set to 0.5 meters. This gives a sufficient capacity, but is not investigated in detail. However, this will not have a large impact on the dynamic behaviour. Concrete class B45 is assumed in the calculations, giving an elastic modulus of $36 \cdot 10^{9}$ N/m^2.

The cross sections of the remaining parts are modelled using generalized profiles. No width or height dimensions are required as input for this profile type, instead the following cross sectional properties and material definitions are used.

Table 4: Cross sectional properties

Part	A [m^2]	I_y [m^4]	I_z [m^4]	I_T [m^4]	E [N/m^2]	ν	ρ [kg/m^3]
Pylons					$36 \cdot 10^{9}$	0.2	2400
Cables	0.23	0.024	0.024	0.047	$200 \cdot 10^{9}$	0.3	8676
Girder	1.82	5.760	151.2	20.33	$70 \cdot 10^{9}$	0.3	2700
Hangers	0.007	$9.82 \cdot 10^{-6}$	$9.82 \cdot 10^{-6}$	$1.96 \cdot 10^{-5}$	$200 \cdot 10^{9}$	0.3	7850
Conn.el.	1.0	1000	1000	1000	$200 \cdot 10^{9}$	0.3	0

As mentioned in the previous chapter, the diameters of the cables and hangers were determined considering static self weight only. An iterative process was necessary in order to find values rendering an optimal utilization. This will be presented in chapter 6.

Since the girder cross section is modelled with a generalized profile, it does not account for the inertia. It must therefore be added afterwards with the command *BEAM ADDED INERTIA from Abaqus Keywords, as it is not possible to write it directly using the Abaqus scripting commands. The command "keywordBlock.replace" from the Abaqus scripting library must therefore be used in order to implement the Abaqus Keyword command. The command is shown in line 826 in figure 15, where the keyword *BEAM ADDED INERTIA is manually written in line 831, followed by the lines containing the inertia values.

```
822  # ----------------------------------------------------------------------------------------
823  # ---------------------- ADD BEAM GIRDER INERTIA ----------------------------------------
824  # ----------------------------------------------------------------------------------------
825  mdb.models['Model-1'].keywordBlock.synchVersions(storeNodesAndElements=False)
826  mdb.models['Model-1'].keywordBlock.replace(111, """
827  *Beam General Section, elset=\"Girder set\", poisson = 0.3, density=0., section=GENERAL
828  1.82, 5.76, 0., 151.2, 20.33
829  0.,1.,0.
830  7e+10, 2.69231e+10
831  *BEAM ADDED INERTIA
832  4914,    0.,          0.,      0.,      15309,   408740,      0.
833  1977,    0.,          0.,      0.,      2019,    158358,      0.
834  6000,    0.,          2.0,     0.,      3,       450000,      0.
835  33,      -14.75,      1.4,     0.,      0.,      0.,          0.
836  33,      14.75,       1.4,     0.,      0.,      0.,          0.""")
837
838  """
839  From above
840  1: Aluminium girder
841  2: Aluminium diaphragmas
842  3: Asphalt
843  4: Hanger heads left
844  5: Hanger heads right
845  """
```

Figure 15: Python code for adding beam inertia

In order to calculate the inertia that needs to be added to the girder wire element in the FE model, a simplified sketch of the cross section of the proposed bridge deck has been made. The contributing parts are the aluminium girder, aluminium diagphragmas, asphalt and hanger heads, as shown in figure 16 below.

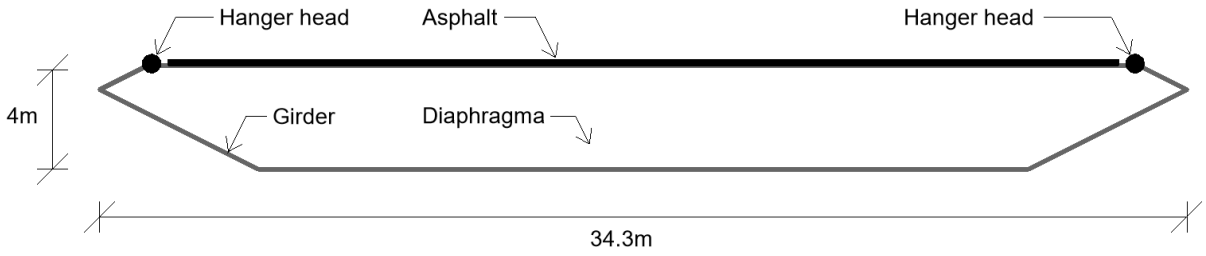


Figure 16: Positions of the additional inertia components for the wire element in the FE model

Table 5 shows the properties and resulting linear mass and rotational inertia for each component from figure 16. The resulting values from the table are inserted to the python script as can be seen in figure 15. Note that the hanger head contribution is split in two parts, one on each side, as shown in figure 16.

Table 5: Inertia values

Inertia component		Linear mass [kg/m]	Rotational inertia I_{11} [kgm^2/m]	Rotational inertia I_{22} [kgm^2/m]
Girder		4914	15309	408780
ρ_{alu}	2700 kg/m^3			
I_y	5.67 m^4			
I_z	151.4 m^4			
A	1.82 m^2			
Asphalt		6000	3	450000
$\rho_{asphalt}$	2500 kg/m^3			
b	30 m			
h	0.08 m			
z'	2 m			
y'	0 m			
Diaphragma		1977	2019	158358
ρ_{alu}	2700 kg/m^3			
t	0.027 m			
b	31 m			
h	3.5 m			
dx	4 m			
Hanger head		67	0	0
m	800 kg			
z'	2 m			
y'	14.75 m			
dx	24 m			

As stated in the Abaqus Scripting reference guide, additional inertia for Euler-Bernoulli beams, such as B33, is ignored [11]. For that reason the girder is modelled using B32 Timoshenko beam elements, so that the added inertia will be included. For the remaining parts, Eurler-Bernoulli B33 beam elements are used.

4.4 Connection elements

Since the model is based on 1D wire elements, supplementary elements connecting the girder with the hangers were added to the model in order to recreate the correct geometry and give a proper torsional behaviour of the girder. The connection elements are highlighted in figure 17.

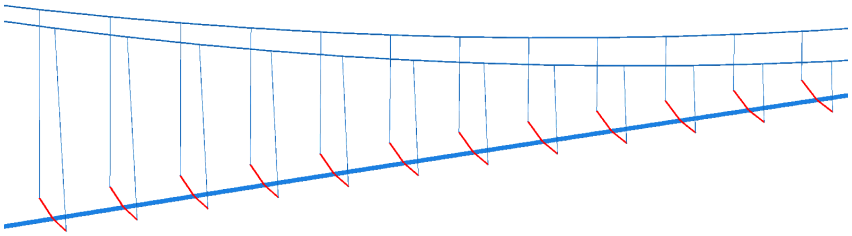
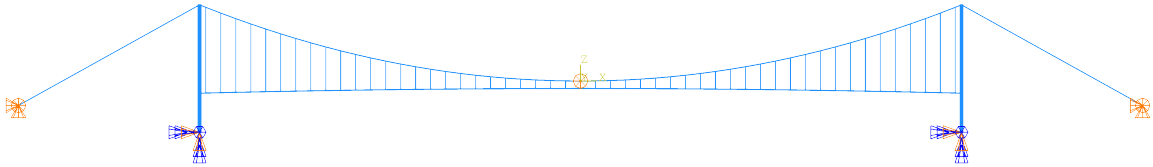


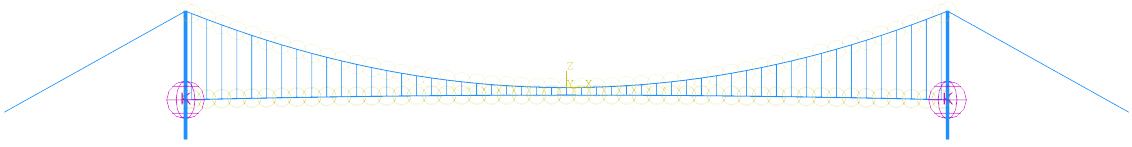
Figure 17: Connection elements

4.5 Boundary conditions and interactions

The pylons as well as the ends of the main cables are fixed against translation and rotation at the bottom. All parts in the model are tied except for the interaction between the pylons and the girder. Some relative movement between the pylons and the girder in the longitudinal direction should be allowed for to imitate roller sections. Horizontal springs are added in both ends to allow for some deformation but also give stiffness. Tying of the cables and hangers is not the best way of representing the true behaviour of the bridge, but it is an approach that does not affect the results in a large manner. Figure 18 shows the boundary conditions and interactions in the FE model.



(a) Boundary conditions

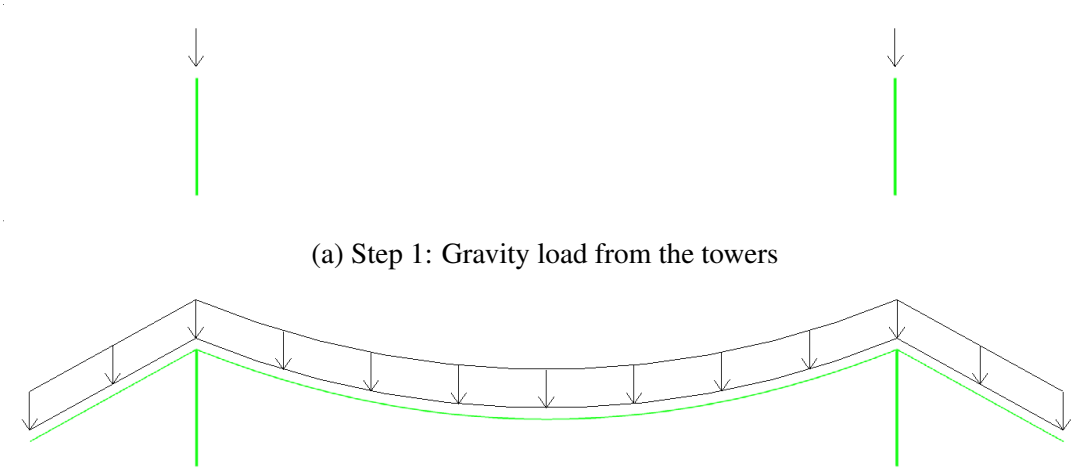


(b) Ties and springs

Figure 18: Boundary conditions and interactions in the FE model

4.6 Loading of the model

Gravity loads are applied in static steps as shown in figure 19. A non-linear analysis is requested in order to take into account large displacements and geometric non-linearities. It is important to make sure that the model is tensioned properly and has the desired geometry after the gravity load is applied, as the geometric stiffness will affect the results from the modal analysis. By introducing the parameter "vert_curve", the vertical curvature of the girder can be adjusted until it stays in the desired equilibrium position after the static steps are run. There are also other methods for tensioning the bridge model, for instance by introducing a change in temperature. However, the steps shown here represents the building steps chronologically, which can be useful during the construction phase.



(a) Step 1: Gravity load from the towers

(b) Step 2: Added gravity load from the cables

(c) Step 3: Added gravity load from the girder and hangers

Figure 19: Loading steps in the FE model

The flow chart in figure 20 illustrates the structure of the python script that generates the FE model. The yellow boxes are input parameters, while the grey are components that are automatically generated. It is worth mentioning that this is not necessarily the order that Abaqus executes the commands, but the flow chart is created to give a better understanding of the dependencies between the commands in the python script.

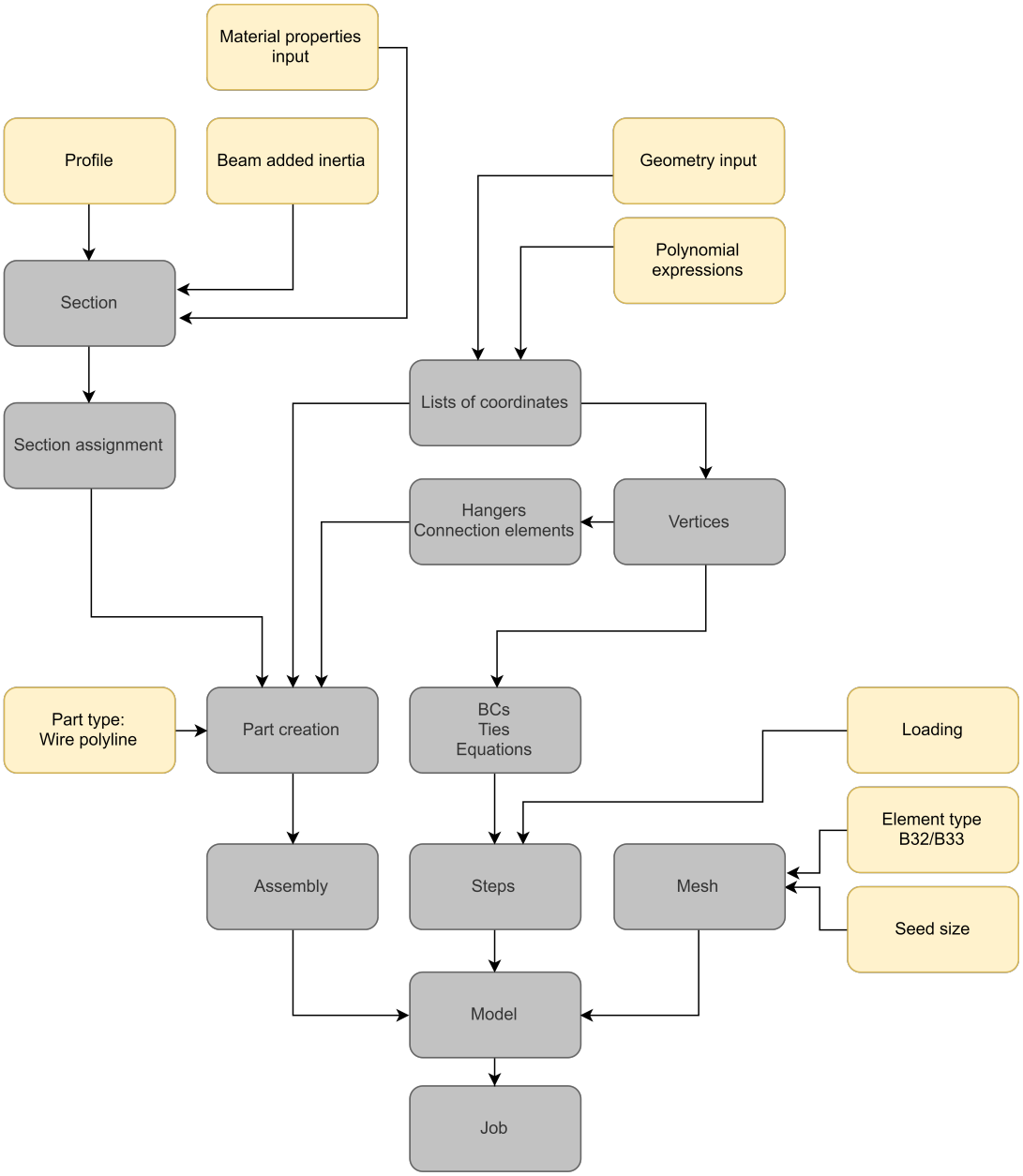


Figure 20: Flow chart illustrating the dependency between the commands in the python script

5 Matlab programs

Matlab scripts provided by Professor Ole Øiseth have been used to do calculations and find results. One script extracts information from the FE model by reading information from the dat.-file generated by Abaqus, and saves mode shapes as well as generalized masses and eigenfrequencies. These are used in the wind analyses. The modal damping and stiffness matrices $\tilde{\mathbf{C}}$ and $\tilde{\mathbf{K}}$ are calculated from the modal mass matrix $\tilde{\mathbf{M}}$ by using the formulae in equation (2.5). The damping ratio ζ is set to 0.005, which is a realistic assumption.

ADs used in the wind analyses are found by doing curve fitting to the provided AD values from the Great Belt East Bridge. The script uses the following rational function expression

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

where a_i and d_l are unknown coefficients that must be estimated by linear and non-linear curve fitting. ADs can also be calculated directly by using Theodorsen's expressions in equation (2.36).

Another script calculates the spectral densities and standard deviations of the buffeting response. Apart from information found from the Abaqus export, input for the buffeting response calculations are mean wind velocity V, selected modes, ρ_{air}, girder width B and height D, static load coefficients and ADs. By using the relations from chapter 2.3, the response spectral density $\mathbf{S}_r$ can be calculated. Variance, standard deviations and correlations can be calculated from this.

The last script does flutter calculations. Interesting output from the these is a value for the critical wind velocity V_{CR} as well as plots for Im(S) and Re(S). Necessary input to find these is $\tilde{\mathbf{M}}$, $\tilde{\mathbf{C}}$, $\tilde{\mathbf{K}}$ from the abaqus export and ρ_{air}, B and ADs.

6 Results

This chapter covers the results from the modal analysis and the dynamic wind analyses. In chapters 6.1 - 6.4 the results from the bridge with the aluminium girder will be presented. Chapter 6.5 gives the corresponding results for a similar model with a steel girder. These will hereby be denoted the aluminium and steel options.

When considering the bridge loaded with static self weight, a utilization of 30 $\%$ in the cables and hangers is desirable. For the aluminium option, this is achieved with a diameter of 550 mm for the cables and 100 mm for the hangers.

6.1 Modal data

Abaqus identifies the modes of the structure by solving the eigenvalue problem as in equation (2.2). Each mode shape has been evaluated in the graphical user interface, by studying the participation factors in the .dat file, and by comparing the amplitudes for motion in y, z and θ directions. The most relevant modes for the girder has been chosen for further assessment.

Table 6: Modal data from the FE model with an aluminium girder

Name	Abaqus mode no.	Direction	Gen. mass $\tilde{M}_i$ [kg]	f [Hz]	ω [rad/s]
Hs1	1	y	$9.862 \cdot 10^6$	0.053	0.34
Va1	2	z	$1.062 \cdot 10^7$	0.11	0.66
Ha1	3	y	$8.739 \cdot 10^6$	0.13	0.83
Vs1	4	z	$5.372 \cdot 10^6$	0.14	0.87
Vs2	5	z	$8.448 \cdot 10^6$	0.20	1.25
Va2	6	z	$1.020 \cdot 10^7$	0.21	1.33
Hs2	8	y	$4.297 \cdot 10^6$	0.25	1.56
Ha2	10	y	$2.470 \cdot 10^6$	0.26	1.65
Hs3	11	y	$6.716 \cdot 10^6$	0.27	1.71
Vs3	13	z	$8.830 \cdot 10^6$	0.28	1.75
Va3	14	z	$1.031 \cdot 10^7$	0.34	2.16
Hs4	16	y	$3.978 \cdot 10^6$	0.39	2.43
Ts1	17	θ	$6.571 \cdot 10^6$	0.39	2.48
Ta1	29	θ	$5.923 \cdot 10^6$	0.45	2.81
Ts2	40	θ	$5.214 \cdot 10^6$	0.68	4.26
Ta2	59	θ	$5.810 \cdot 10^6$	0.88	5.54
Ts3	74	θ	$6.656 \cdot 10^6$	1.11	6.96
Ta3	93	θ	$5.321 \cdot 10^6$	1.32	8.31

In table 6, the first six modes for each direction are presented, along with their respective generalized masses and frequencies. In the first column, the modes are classified as symmetrical

and asymmetrical. The chosen modes from table 6 are visualized in figure 21 below. Some have excitation in mainly one direction, while others have a coupling between several. The number of half sinusoidal waves increases with the frequency.

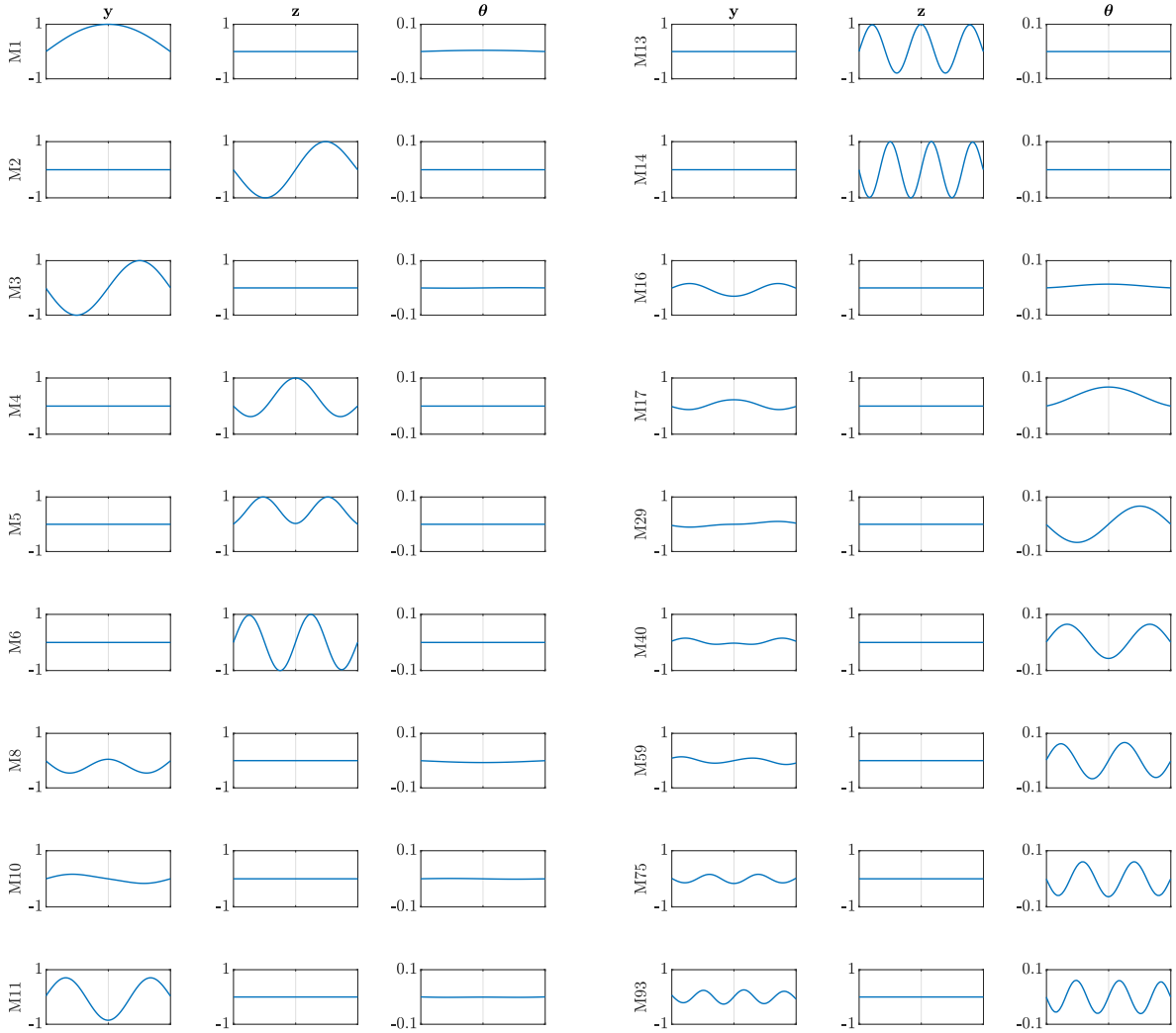


Figure 21: Componentwise normalized mode shape plots for selected modes in Abaqus for the aluminium option

6.2 Aerodynamic derivatives

As mentioned in chapter 3.3, the ADs from the Great Belt East Bridge has been used for the Langenuen bridge. Plots of the ADs from the curve fitting by the use of rational functions are shown in figure 22, and corresponding ADs using Theodorsen's flat plate theory are shown in figure 23.

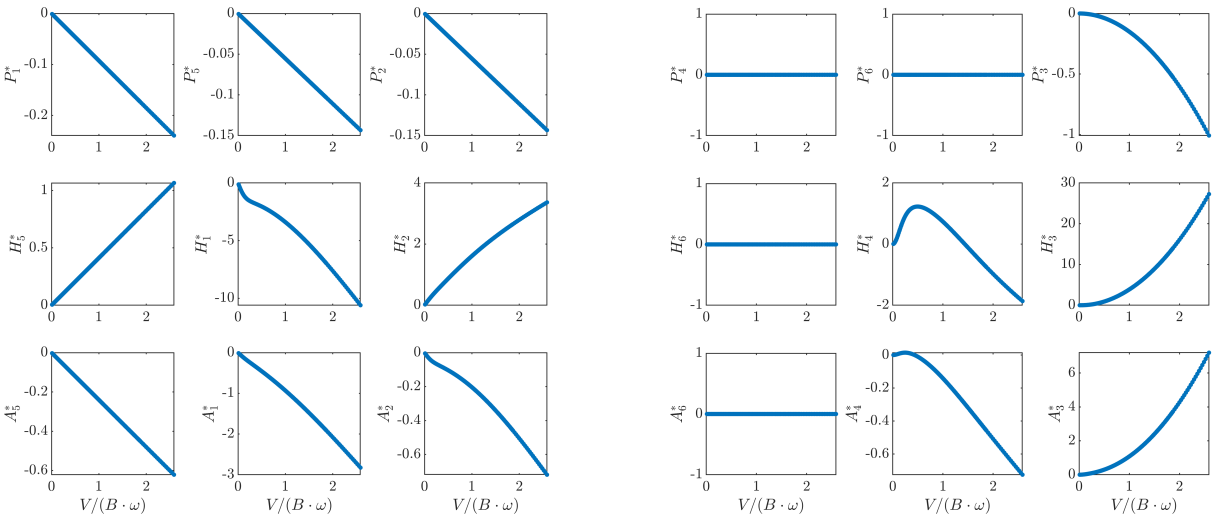


Figure 22: ADs used in the aerodynamic damping matrix $\tilde{\mathbf{C}}_{ae}$ and stiffness matrix $\tilde{\mathbf{K}}_{ae}$

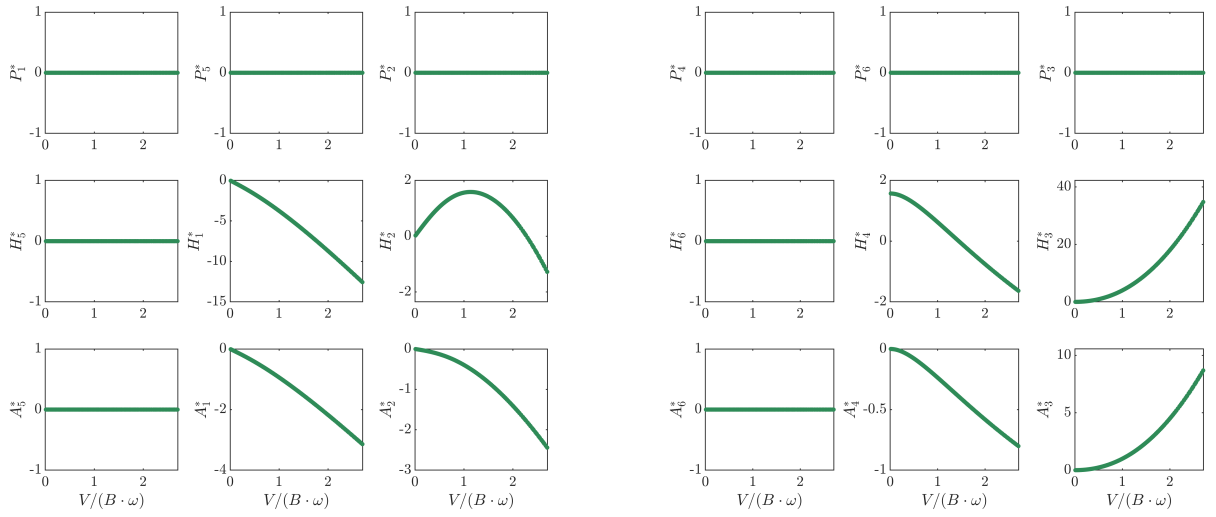


Figure 23: ADs calculated according to Theodorsen's flat plate theory

As can be seen from figures 22 and 23, the ADs related to vertical and torsional motion, i.e. the two by two right bottom ADs, are shape-wise similar except H*2. H*2 have opposite directions

for the two. The remaining are defined as zero for Theodorsen's flat plate theory, but some have non-zero values from the wind tunnel testing. There are also small differences in the values for the ADs obtained by the two methods, which is as expected. This will result in different critical flutter velocities.

6.3 Buffeting response

The results in the following are obtained by considering the 50 first modes, i.e. with frequencies up to 5.11 rad/s. The buffeting response is calculated by using the wind spectra from *Håndbok N400*, as stated in chapter 2.3. The wind spectra for Langenuen are calculated for a mean wind velocity of 20 m/s, and are shown in the figure below. One can see that low frequencies are dominating in the wind spectra.

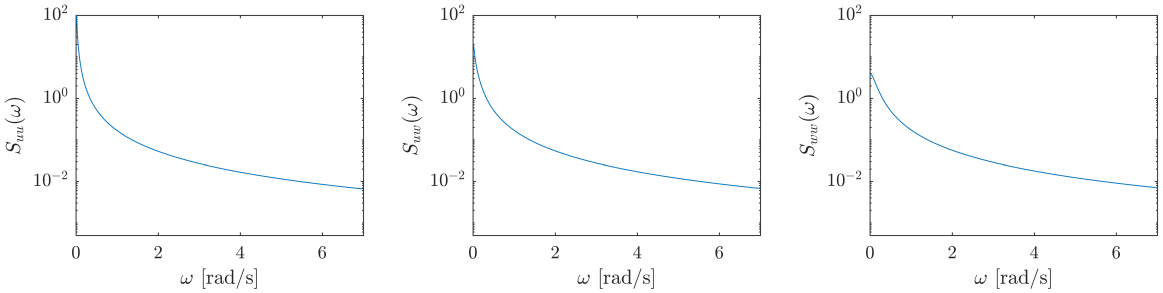


Figure 24: Horizontal and vertical spectra S_{uu} and S_{ww}, and cross spectrum S_{uw}

6.3.1 Response spectral density

The spectral density shows at which frequencies dynamic magnification is likely to happen. The response spectral densities are plotted for different positions along the girder for the y, z and θ directions. Figures 25 to 27 represents the quarter span. The corresponding plots for the midspan and three-quarter span can be found in appendix C.1.

The peaks in the spectral density plots can be seen in context with the frequency values in table 6. All the chosen horizontal modes can be found as peaks in figure 25, together with modes of higher frequencies.

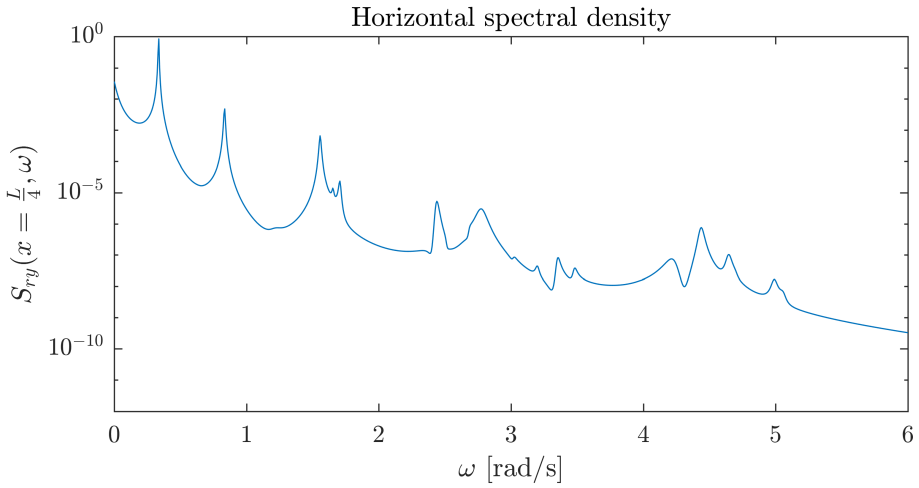


Figure 25: Horizontal response spectrum in the quarter span. V_{mean} = 20 m/s

For figure 26, all the tabulated vertical modes except V2 and V4 can be found as peaks. As can be seen from the mode shape plot in figure 21, these modes have little or no motion in the quarter span, which can be why they give no distinct peaks in the spectrum.

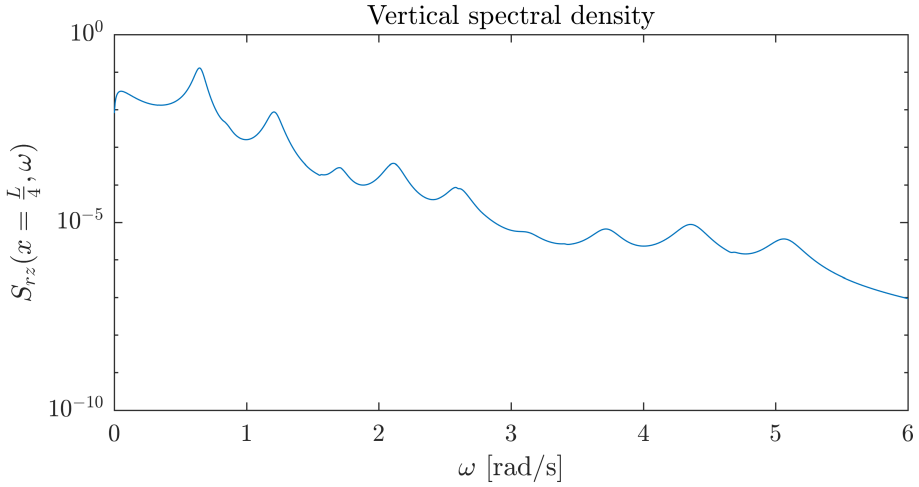


Figure 26: Vertical response spectrum in the quarter span. V_{mean} = 20 m/s

The three distinct peaks to the right in figure 27 represent the three first torsional modes in table 6. To the left in the plot there are some modes that are not pure torsional, but also has some contribution in the horizontal direction. This is clear by comparing with the peaks in figure 25. The first horizontal mode has a large excitation, and this will also contribute to torsion in the girder, shown as the highest peak in figure 27. This could be due to the inclination of the hangers, which causes a coupling between horizontal and torsional motion. The three last torsional modes in table 6 are out of the frequency range of the plot.

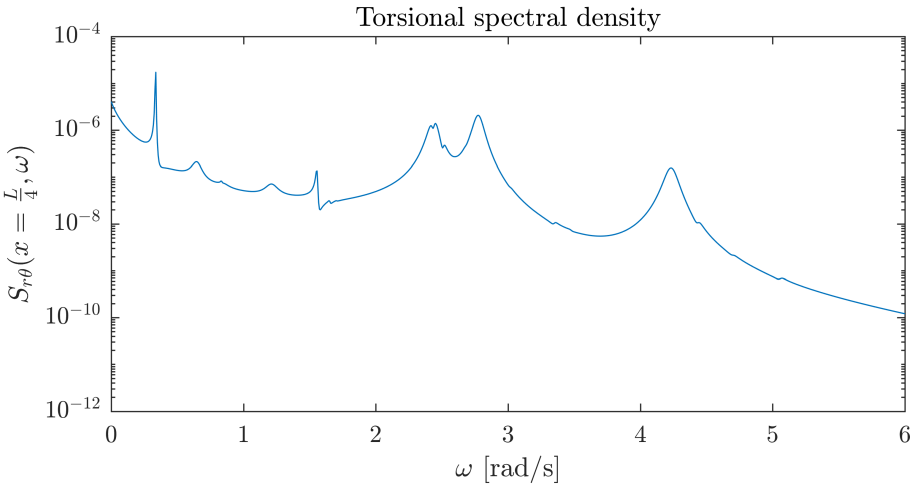


Figure 27: Torsional response spectrum in the quarter span. V_{mean} = 20 m/s

The spectra for the three-quarter span are quite similar to those in figures 25 to 27. The midspan spectra are missing some of the peaks that are visible in the others. This could be due to little or no motion at midspan in the asymmetrical modes.

6.3.2 Standard deviation and correlation

In order to get a better understanding of the spectral density plots, the standard deviations have been calculated for different mean wind velocities along the girder. Figures 28-30 show the standard deviations for the aluminium option in the three directions y, z and θ.

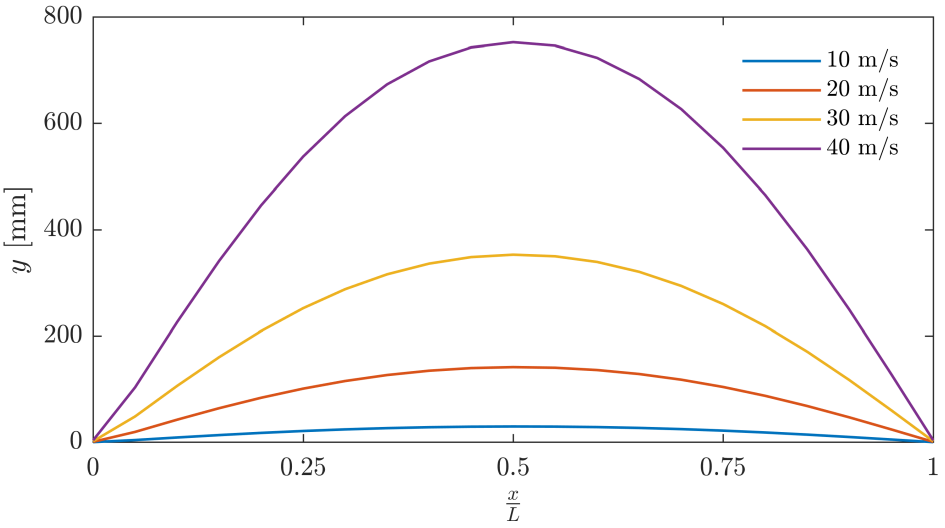


Figure 28: Standard deviation $\sigma_{r_y r_y}$ in the horizontal direction for the aluminium option

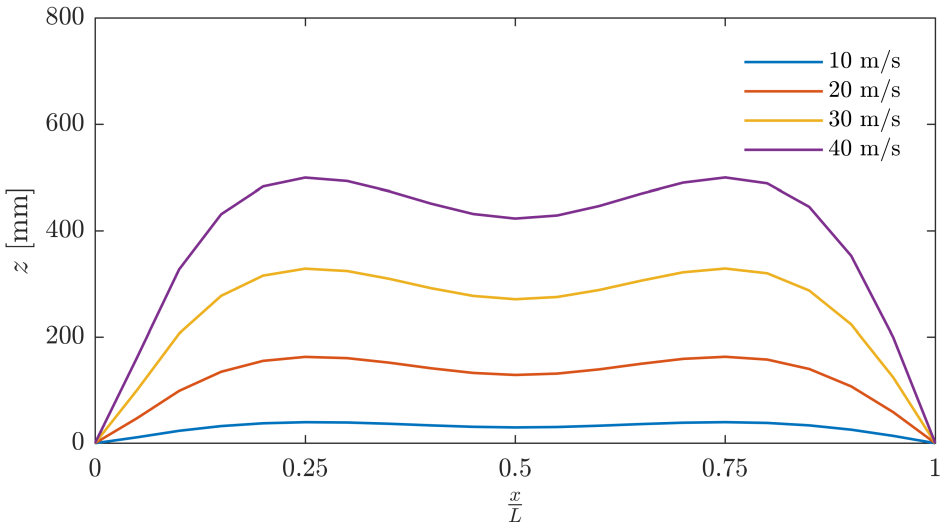


Figure 29: Standard deviation $\sigma_{r_z r_z}$ in the vertical direction for the aluminium option

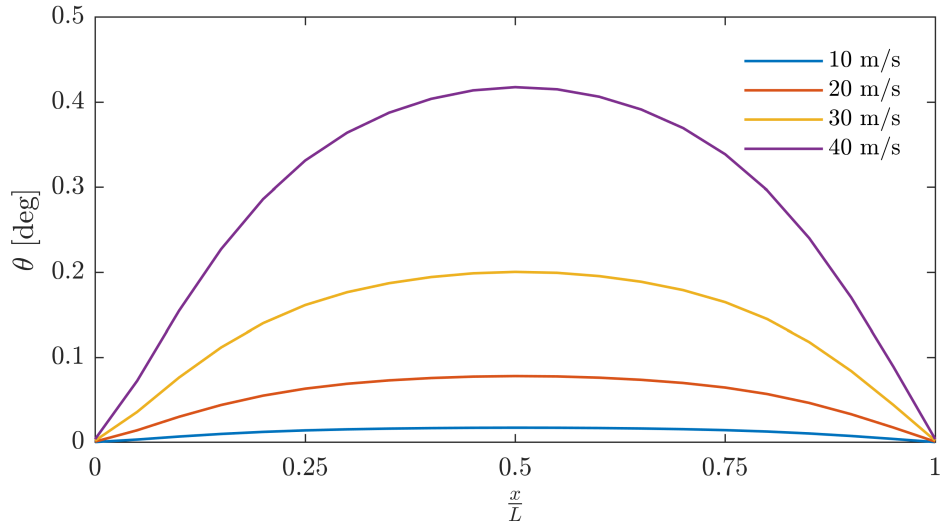


Figure 30: Standard deviation $\sigma_{r_\theta r_\theta}$ in the torsional direction for the aluminium option

It can be seen from the figures above that in y and θ directions the buffeting response is highest at midspan. For z the response is highest around the quarter and three-quarter span. Generally, the horizontal response is higher than the vertical one. For all the figures, the response is symmetrical about midspan.

Correlation matrices are calculated according to equation (2.47) in the theory for a mean wind velocity of 20 m/s.

$$\boldsymbol{\rho}_{rr}(x = \frac{L}{4}) = \begin{bmatrix} 1.000 & 0.022 & -0.491 \\ 0.022 & 1.000 & -0.254 \\ -0.491 & -0.254 & 1.000 \end{bmatrix}$$

$$\boldsymbol{\rho}_{rr}(x = \frac{L}{2}) = \begin{bmatrix} 1.000 & 0.039 & -0.553 \\ 0.039 & 1.000 & -0.290 \\ -0.553 & -0.290 & 1.000 \end{bmatrix}$$

$$\boldsymbol{\rho}_{rr}(x = \frac{3L}{4}) = \begin{bmatrix} 1.000 & 0.024 & -0.495 \\ 0.024 & 1.000 & -0.254 \\ -0.495 & -0.254 & 1.000 \end{bmatrix}$$

The correlation between y and θ is the highest of those in the matrices. The configuration of the hangers can be a reason to this, as has been mentioned. The lowest correlation is between y and z. This is beneficial, because maximum bending about both axes will not occur simultaneously.

6.4 Flutter

Table 7 shows the critical mode combinations for flutter, along with their respective critical wind velocity V_{CR}, as well as critical circular frequencies ω_{CR} and reduced velocities V_{red} calculated from V_{CR}. Theodorsen's critical velocities are also included for comparison. Note that the first column indicates the number of different mode types that are included from each direction, not the order.

Table 7: Flutter combinations for the aluminium option

Combination	Abaqus mode no.	V_{CR} [m/s]	V_{CR} [m/s] Theodorsen	ω_{CR} [rad/s]	V_{red}
Symmetrical					
1V + 1T	4, 17	99.5	124.3	1.86	1.56
1V + 1T	5, 17	91.8	118.7	1.98	1.35
2V + 1T	4, 5, 17	78.0	98.2	2.13	1.07
3V + 1T	4, 5, 13, 17	77.4	97.6	2.14	1.06
2V + 1H/T + 1T	4, 5, 16, 17	77.7	97.6	2.11	1.08
2V + 1H/T + 1T	4, 5, 17, 19	74.2	91.0	2.06	1.05
3V + 1H/T + 1T	4, 5, 13, 16, 17	77.0	95.8	2.11	1.66
3V + 2H/T + 1T	4, 5, 13, 16, 17, 19	73.3	89.5	2.05	1.04
3V + 4H/T + 1T	4, 5, 8, 11, 13, 16, 17, 19	72.0	83.7	1.54	1.36
Asymmetrical					
1V + 1T	2, 29	90.7	111.8	2.26	1.17
3V + 2T	2, 6, 14, 29, 59	90.7	111.7	2.26	1.17
First 20 modes	1:20	72.3	83.7	1.54	1.37

The mode combinations are results of an elimination process. Initially flutter was checked for all 100 modes, but it was found that the first 20 modes gave the same critical wind velocity of 72.3 m/s. For this reason the search was narrowed down to combinations of the first 20 modes. It was found that combinations of a torsional mode and one or several shape-wise similar vertical modes with lower frequencies gave the lowest V_{CR}, corresponding to what is known from the literature. Even though mode 8, 11 and 16 are included in the flutter combination with the lowest critical velocity in the table, their main excitation in the FE model is in the cables and hangers. This deviates from the true behaviour because the connections are approximately hinged, not rigid as in the FE model, and thus allows for relative rotation. For this reason these particular modes will be disregarded in the determination of the critical flutter combination. Therefore, the critical velocity is 74.2 m/s for the aluminium option.

Figure 31 displays the real and imaginary parts of the eigenvalue for the combination of the

first symmetrical torsional mode with the first two shape-wise similar vertical modes. The flutter limit is reached when the real part of the eigenvalue reaches zero.

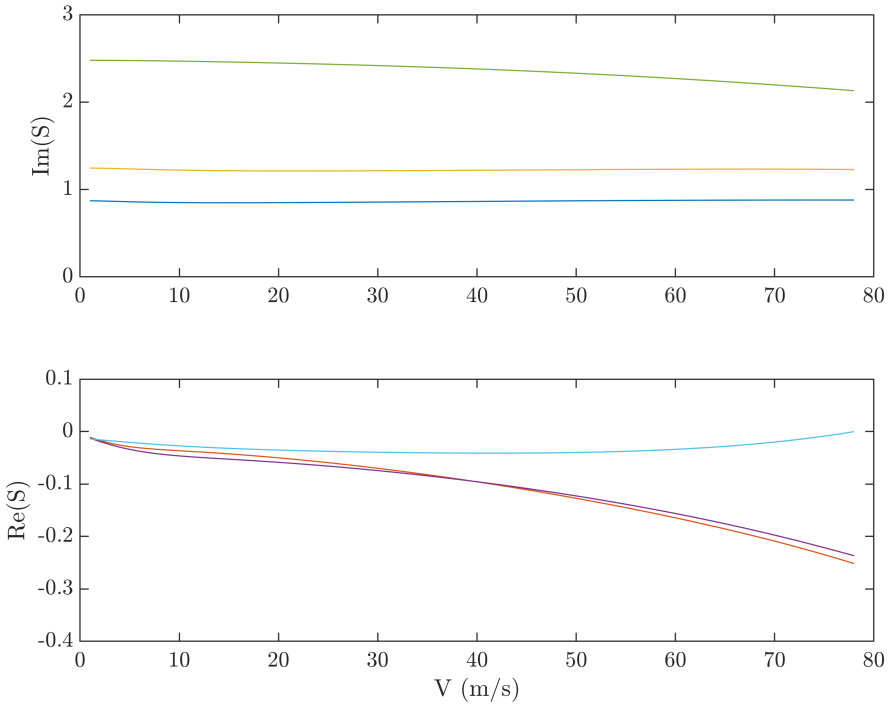


Figure 31: Real and imaginary parts of eigenvalue S for a combination of modes Vs1, Vs2 and Ts1

As seen from the imaginary part of the eigenvalue, the bridge looses stiffness with an increasing mean wind velocity. The reason for this change is that the torsional mode affects the mean wind dependent aerodynamic stiffness matrix. It is worth noticing that the stiffness contribution of the vertical modes is approximately constant until the flutter limit is reached.

The aerodynamic damping matrix affects the real part of the eigenvalue, which increases towards zero for an increasing wind velocity. It can be seen that the torsional mode is the driving mode in the coupling process, as the damping drops to zero at 78 m/s.

The computed values of V_{red} from table 3 are in the range 0.27-2.71. The flutter combinations obtained in table 7 are within this range.

6.5 Comparison between aluminium and steel options

A comparison of the results obtained from the aluminium bridge has been performed by changing the girder material to steel. This includes changing the elastic modulus from 70 000 MPa to 210 000 MPa and the density from 2700 kg/m^3 to 7850 kg/m^3, as well as changing the diameter of the cables and hangers so that their utilization of 30% under static self weight is maintained for the steel option. This is equivalent to an increase from 550 mm to 700 mm for the main cables, and from 100 mm to 120 mm for the hangers. The rest of the FE model is kept unchanged. Table 8 shows the new frequencies for the steel option, where the grey column contains the previously obtained results from the aluminium option.

Table 8: Modal data from the FE model with a steel girder

Name	Abaqus mode no.	Direction for steel	Gen. mass $\tilde{M}_i$ [kg] for steel	f_{alum} [Hz]	f_{steel} [Hz]	Deviation from aluminium [%]
Hs1	1	y	$1.683 \cdot 10^7$	0.053	0.059	+11
Va1	2	z	$1.816 \cdot 10^7$	0.11	0.11	+0
Ha1	4	y	$1.572 \cdot 10^7$	0.13	0.16	+23
Vs1	3	z	$9.005 \cdot 10^6$	0.14	0.14	+0
Vs2	5	z	$1.458 \cdot 10^7$	0.20	0.20	+0
Va2	6	z	$2.100 \cdot 10^7$	0.21	0.22	+5
Hs2	9	y	$4.876 \cdot 10^6$	0.25	0.26	+4
Ha2	11	y	$4.120 \cdot 10^6$	0.26	0.27	+4
Hs3	13	y	$9.548 \cdot 10^6$	0.27	0.33	+22
Vs3	12	z	$1.566 \cdot 10^7$	0.28	0.30	+7
Va3	14	z	$1.755 \cdot 10^7$	0.34	0.38	+12
Hs4	18	y	$6.467 \cdot 10^6$	0.39	0.41	+5
Ts1	25	θ	$8.768 \cdot 10^6$	0.39	0.44	+13
Ta1	33	θ	$1.007 \cdot 10^7$	0.45	0.57	+21
Ts2	56	θ	$6.637 \cdot 10^6$	0.68	0.86	+26
Ta2	67	θ	$9.198 \cdot 10^6$	0.88	1.13	+28
Ts3	90	θ	$6.952 \cdot 10^6$	1.11	1.41	+27
Ta3	100	θ	$7.220 \cdot 10^6$	1.32	1.64	+24

As seen from table 8, some of the frequencies deviate little, but in general they are higher for the steel option. There is also a tendency that the torsional frequencies deviate more than the horizontal and vertical ones. Even though the mentioned parameters are changed, the modes have the same shape as those found for aluminium.

When the material is changed from aluminium to steel, both the mass and stiffness of the girder are changed. The material stiffness $\mathbf{k}_m$ is proportional to the elastic modulus E, which is 3 times higher for steel. The mass is proportional to the density ρ. The values 7850 $\mathrm{kg/m^3}$ for steel and 2700 $\mathrm{kg/m^3}$ for aluminium gives a ratio of about 2.9. If these numbers are inserted in the basic relation given in equation (2.3), one could say that the total difference in the natural frequency ω_n is about 1.6%. This is not very much, so it is clear that also other parts than only the girder contribute to the difference in frequencies. A change in the cable diameter affects the area, which contributes to the material stiffness $\mathbf{k}_m$ and mass. The geometric stiffness $\mathbf{k}_G$ is increased in the cables and hangers because of higher axial forces. As a result, it is likely that the change is relatively higher for the stiffness than for the mass. Thus, including the contribution from the cables and hangers, the observation that natural frequencies of the system increase is supported.

The buffeting response spectra for steel are found in appendix C.2. From these, the standard deviations of the response for the steel option have been calculated and plotted in the same manner as for the aluminium option in chapter 6.3.2. They are shown in the figures below.

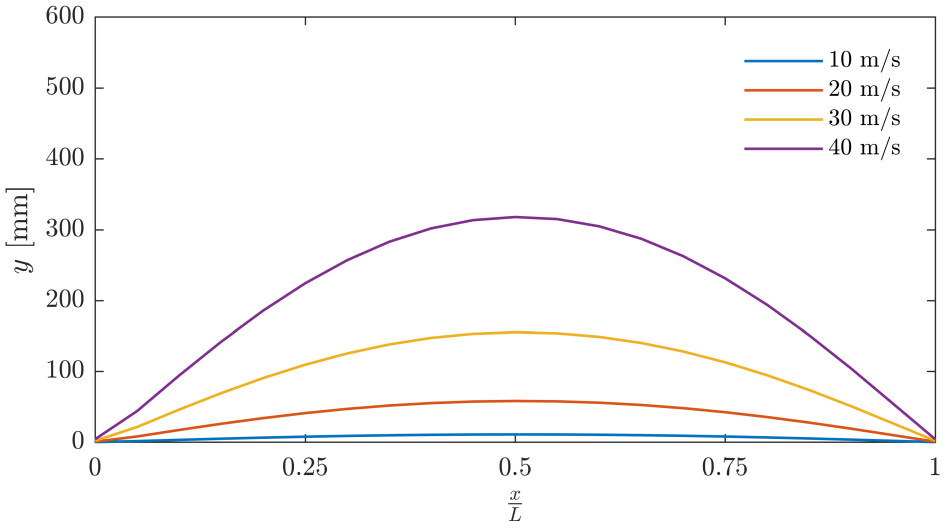


Figure 32: Standard deviation $\sigma_{r_y r_y}$ in the horizontal direction for the steel option

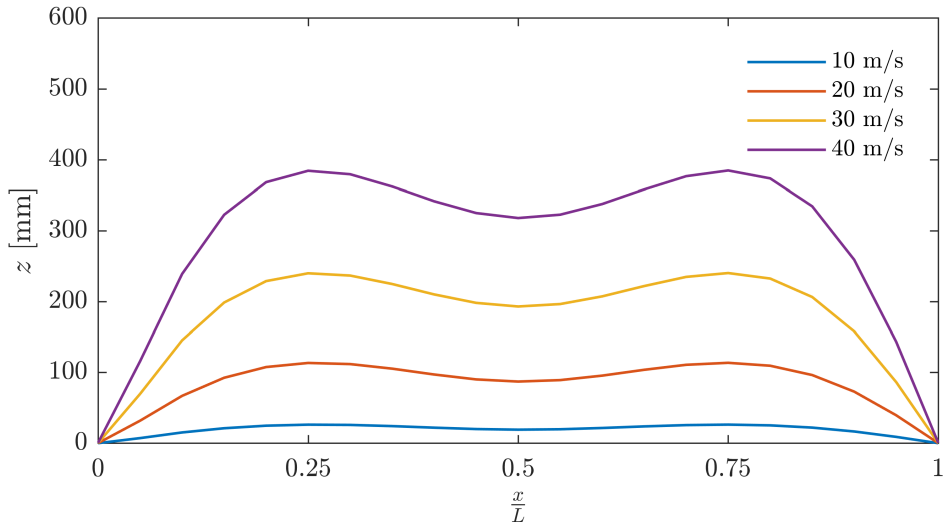


Figure 33: Standard deviation $\sigma_{r_z r_z}$ in the vertical direction for the steel option

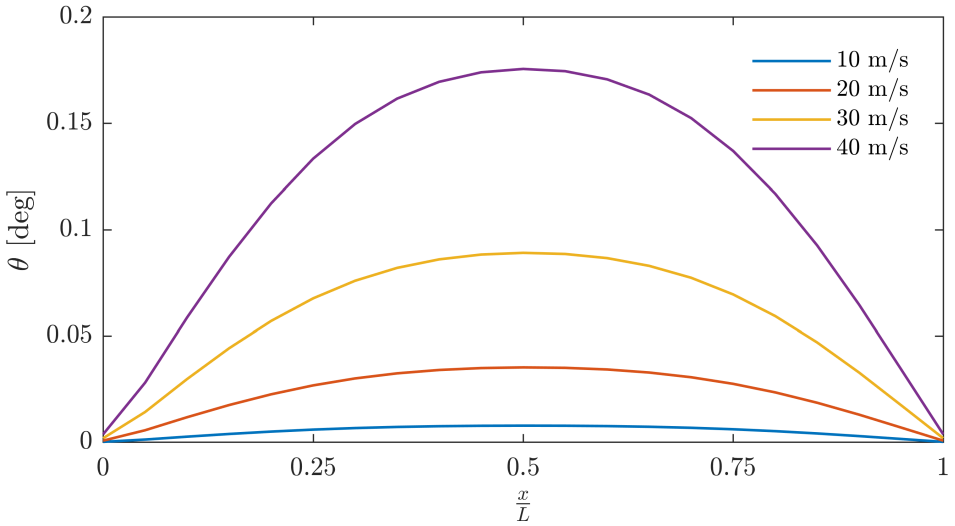


Figure 34: Standard deviation $\sigma_{r_\theta r_\theta}$ in the torsional direction for the steel option

The standard deviations in figures 32 - 34 are shape-wise similar to those of the aluminium option. In all three directions, they are higher for the aluminium option than for the steel option. When V is set to 40 m/s, the steel option has a standard deviation of about 300 mm in the horizontal direction at midspan, in oppose to 800 mm for the aluminium option. In the vertical direction however, the standard deviations are about 320 mm and 430 mm, respectively. In the torsional direction the values of the steel option are less than half as much as for the aluminium option.

The vertical standard deviations are generally higher than the horizontal ones, which was the opposite case for aluminium. A change of the girder material from aluminium to steel results in a larger change in the horizontal buffeting response than in the vertical response.

In the vertical direction, the standard deviation is larger at the quarter span than at midspan. This could be due to the contributions from the mode shapes. Asymmetrical modes have little or no excitation at midspan, and will therefore not contribute to the standard deviation here. On the other hand, one cannot use the same argument for the symmetrical modes. They may have a deflection at midspan, and it is therefore not given that the maximum standard deviation will take place at quarter span. This is the case for the horizontal and torsional standard deviations, as the maximum values are found around midspan. A reason for this could be that the symmetrical modes are more dominating.

As for the aluminium option, the correlation matrices are calculated for the steel option at a mean wind velocity of 20 m/s. In the three different locations along the girder, they are given by

$$\boldsymbol{\rho}_{rr}(x = \frac{L}{4}) = \begin{bmatrix} 1.000 & 0.020 & -0.393 \\ 0.020 & 1.000 & -0.189 \\ -0.393 & -0.189 & 1.000 \end{bmatrix}$$

$$\boldsymbol{\rho}_{rr}(x = \frac{L}{2}) = \begin{bmatrix} 1.000 & 0.042 & -0.423 \\ 0.042 & 1.000 & -0.234 \\ -0.423 & -0.234 & 1.000 \end{bmatrix}$$

$$\boldsymbol{\rho}_{rr}(x = \frac{3L}{4}) = \begin{bmatrix} 1.000 & 0.021 & -0.394 \\ 0.021 & 1.000 & -0.192 \\ -0.394 & -0.192 & 1.000 \end{bmatrix}$$

The correlation between the buffeting response in horizontal, vertical and torsional direction is in general higher for the aluminium option than for the steel option except for one. An interesting found is that the correlation between directions y and z at midspan, $\rho_{r_y r_z}$ is higher for the steel option.

The same methodology as for the aluminium option is used to determine the critical wind velocity for the steel option. Modes 8,9,17,18 have a high excitation in the cables, and will therefore not give a correct critical velocity. For this reason these modes cannot be included in the critical flutter combinations. When disregarding these, the critical wind velocity V_{cr} is 106.2 m/s, as shown in table 9.

Table 9: Flutter combinations for the steel option

Combination	Abaqus mode no.	V_{CR} [m/s]	V_{CR} [m/s] Theodorsen	ω_{CR} [rad/s]	V_{red}
Symmetrical					
1V + 1T	3, 25	131.6	151.4	1.90	2.02
1V + 1T	5, 25	123.7	145.8	2.03	1.78
2V + 1T	3, 5, 25	106.8	129.0	2.23	1.39
3V + 1T	3, 5, 12, 25	106.2	129.2	2.24	1.38
3V + 1H/T + 1T	3, 5, 12, 13, 25	106.5	128.8	2.24	1.39
3V + 5H/T + 1T	3, 5, 8, 9, 12, 13, 17, 18, 25	104.5	117.0	1.61	1.89
Asymmetrical					
1V + 1T	2, 33	153.9	188.6	2.87	1.56
3V + 1T	2, 6, 14, 33	153.9	188.6	2.78	1.56
First 25 modes	1:25	104.8	117.1	1.61	1.90

The elimination process for finding the flutter combinations of the bridge, described in chapter 6.4, was also conducted when the girder material was set to steel. It showed that the critical flutter combinations for the steel option coincided to those of the aluminium option, i.e. the three first symmetrical vertical modes in combination with the first symmetrical torsional mode, as shown in table 9. A summary of the critical velocity of the aluminium and steel option, as well as the design wind velocity, is given below. The calculations for the design wind velocity at Langenuen can be found in appendix A.

Table 10: Critical wind velocities [m/s]

Aluminium	Steel	Design wind velocity
74.2	106.2	63.6

As seen from the table, the critical wind velocity for the aluminium option is higher than the design wind velocity, which indicates that flutter will not be critical for aluminium. The only parameters that were changed in the steel alternative were the elastic modulus, density and diameters of the cables and hangers. As the focus in this thesis has been assessing a bridge having an aluminium girder with cross sectional values provided by Dr.techn. Olav Olsen, an

optimization of the steel girder has not been performed. When calculating the critical wind velocity for the steel option with the same cross sectional geomatry as for aluminium, the obtained value is significantly higher than both the design wind value and the critical wind velocity for aluminium, which was as expected.

There could be uncertainties to the design wind velocity. A small change in a coefficient from equation (A.3) may lead to a significant change. By changing the return period from 50 to 500 years, the resulting design velocity will increase by a factor of 1.12. In other words, one must choose the return period carefully. It is not given that a bridge will fail if its critical velocity is just below the design wind velocity. Furthermore, there are measures that could improve V_{CR}, e.g. guide vanes. This was the case for the Hardanger bridge, where guide vanes increased the critical wind velocity of about 10 m/s [1].

7 Further discussion

7.1 Sources of uncertainty

During the work with this thesis there have been made several decisions that possibly could lead to uncertainties in the results. They are categorized in the following.

As previously mentioned the design given for the Langenuen fjord crossing is preliminary, which means that a final bridge geometry has not been decided. The pylons are modelled from the guidelines given in the report from Norconsult, i.e. A-shaped pylons with a certain height. However, there is no information on the dimensions of the cross sections, so they were based on the drawings of the Hardanger bridge. With a calculated utilization of approximately 60% they withstand the loads in static condition, but a detailed analysis for the dynamic condition has not been performed. This also applies for the cables and hangers, as their cross sectional geometry are based on a utilization of 30% under static self weight. The material properties, such as the elastic modulus and density, are based on a qualitative assessment for the different bridge parts. It is assumed concrete B45 for the pylons, high tensile steel with tensile strength 1570 MPa for the cables and hangers. Further on, the mass moment of inertia of the girder is a source of uncertainty. The girder design has not yet been decided, which implies that the assumed positions of the different girder components give an uncertain mass moment of inertia.

In addition, other decisions made prior to the wind calculations may lead to inaccurate results. The ADs used as a basis for the analyses belong to the cross section of the Great Belt East Bridge. The reason for choosing these was that wind tunnel testing of a proposed bridge deck of Langenuen bridge has not been done, and thus a geometrically similar cross section has been used. This may lead to inaccurate results as it is only an approach. The ratio between the girder width B and height D for Langenuen and The Great Belt East Bridge is not exactly the same, but has a deviation of about 11 %. It can also be mentioned that data found from wind tunnel testing must be used carefully, as they in general can be subject to uncertainties. Furthermore, the use of rational function in the curve fitting of ADs may not be the best method. The assumed wind field, presented by the $\mathbf{S}_{\text{v}}$ calculated from the formulae in *Håndbok N400*, may not be an exact representation of the wind field over time.

7.2 Parametric modelling

A fair amount of the time spent on this thesis has been used to build a complete parametric FE model with the use of Abaqus scripting in python. Although it has been time consuming, this way of modelling has proven itself to be convenient.

There are historically no known suspension bridges that has aluminium as the only structural material in the girder, so for that reason some changes to the design are expected to occur during the time of the project. This was also the case for this thesis. The cross sectional properties have been updated during the work with this thesis, and the cable and hanger diameters are optimized with respect to a utilization of 30 $\%$ under static self weight. With a parametric script, it is easy to quickly do changes in the input and then generate the new FE model.

For this thesis it was effortless to find results for both an aluminium and steel option, and do comparisons between these. This would not be the situation if the model was made directly in Abaqus CAE.

In general, there are many advantages with parametric modelling in python. Python is a well known and universal language that is free of charge. Abaqus uses python for generating a journal file (.jnl) that logs all commands done in the interface. It is therefore possible to implement python code for commands done in the CAE into the script, which is an advantage especially in early stage of the programming.

On the other hand, creating a parametric model can be time consuming. The structure of a parametric script must be planned carefully. A fundamental change of the geometry might not be possible without redoing a lot of the work, e.g. an entirely new tower design or a change from a suspension bridge to a truss bridge. In other words, all commands rely on previous commands and a few input values. Another disadvantage is the lack of visualization while programming.

When weighing the advantages and disadvantages against each other, the project stage is relevant. For an early phase it might not be rational to spend time on making a parametric model, but this could be essential in a design phase. For this thesis choosing a parametric model has been successful, as it was effortless to do necessary adjustments.

7.3 Other aspects

So far the topic of interest has been the comparison between results from the wind analyses of the aluminium and steel option. However, benefits and drawbacks of the materials themselves have not been considered. There are several aspects that should be considered in the evaluations. Some important ones will be discussed in the following.

An important argument for designing a suspension bridge with an aluminium girder is that the lower self weight of the bridge deck opens for a reduction in the cable diameter. Since spinning of cables are both expensive and very time consuming, there could be money to save here. For instance, the cable spinning of the Hardanger bridge lasted for 4 months [12]. In this thesis, the diameter reduction for the main cable was 21%. Another argument in favor of aluminium is that it has a passive film protecting it from corroding. This is not the case for steel, which needs to be protected against corrosion, e.g. by painting and dehumidifying [12]. It is also worth mentioning that aluminium is produced in Norway, while steel girders for bridges often are imported from China.

An argument in favour of steel is the material cost. Aluminium is often more than twice as expensive as steel. In the industry, there is far more experience with the usage of steel. This makes it easier to continue designing bridges with inspiration from previously built structures. For aluminium, cutting edge design is required in order to achieve a well functioning and sustainable bridge.

An important aspect yet to be mentioned is fatigue. Aluminium is more susceptible to fatigue than steel, and this could cause restrictions to the girder design and the realization of a long span suspension bridge with aluminium in the girder.

8 Concluding remarks

This thesis investigates the dynamic behaviour of a suspension bridge with an aluminium girder located in Langenuen, Norway. Dynamic wind analyses have been conducted with a special focus on buffeting response and flutter stability. A FE model has been created using a python script, where modal data has been extracted. Wind tunnel data from the cross section of the Great Belt East Bridge have been used in the analyses because no such data exist for the Langenuen bridge. The results from the dynamic analyses have been compared by changing the girder material from aluminium to steel.

It was found that the correlation between the natural frequencies of the aluminium and steel option was significant. This was as expected because both the stiffness and mass increased when the material of the girder was changed. The vertical and horizontal frequencies deviated less than the torsional ones, and the majority increased.

The standard deviations of the buffeting response are generally higher for aluminium than for steel. In the horizontal direction, the aluminium option had a maximum $\sigma_{r_y r_y}$ of about 750 mm at midspan for a wind velocity of 40 m/s, while the steel option had 320 mm. In the vertical direction, the aluminium option had a maximum $\sigma_{r_z r_z}$ of about 500 mm in the quarter span, in oppose to 380 mm for the steel option. This was as expected, because the stiffness of aluminium is lower than for steel.

Both the aluminium and steel options have acceptable stability against flutter. The design wind velocity was calculated to be 63.6 m/s, while V_{CR} for aluminium and steel was 74.2 m/s and 106.2 m/s, respectively.

The results obtained in this thesis are based on a parametric model which ensures that changes to the geometry and material of the model are quickly implemented. It turned out to be a good investment to spend extra time on making the model parametric in oppose to traditional, as creating a bridge with new cross sectional parameters and a new material definition was carried out in a short amount of time.

There are also other aspects that could be important when deciding on a material. Steel has been a commonly used material for suspension bridges due to its relatively low cost and high strength. Aluminium is more expensive, but weighs less, and it could therefore be possible to save material in the cables and the hangers. Aluminium is produced in Norway, but cutting edge design is required in the design of the first aluminium suspension bridge. The opportunities are

there, but the cost is a factor that could be in disfavour of aluminium when deciding the material.

8.1 Further work

A drawback of aluminium that has not been discussed in detail in this thesis, is the vulnerability to fatigue. This is important to investigate further, as it could be critical for the girder design as well as the feasibility of the entire project.

Another aspect that has not been discussed is the thermal expansion coefficient α. For aluminium this is more than twice as much as for steel. This needs to be taken into account when designing the joint between the girder and the abutments.

References

[1] Norconsult. Bru over Langenuen og Søreidsvika - Skisseprosjekt; 2015.

[2] Strømmen EN. Structural Dynamics. vol. 2 of Springer Series in Solid and Structural Mechanics. 2014th ed. Cham: Springer International Publishing; 2014.

[3] Øiseth O. Lecture notes on wind induced dynamic response of structures. Institutt for konstruksjonsteknikk, NTNU; 2016.

[4] Theodorsen T. General Theory of Aerodynamic Instability and the Mechanism of Flutter; 1949.

[5] Scanlan RH, Tomko JJ. Airfoil and Bridge Deck Flutter Derivatives. Airfoil and Bridge Deck Flutter Derivatives. 1971 01;97:1717–1733.

[6] Statens vegvesen. Bruprosjektering: Håndbok N400. Vegdirektoratet; 2015.

[7] Norconsult. Bru over Langenuen Skisseprosjekt - Overslag kritisk vindhastighet; 2015.

[8] Statens vegvesen. Veg- og gateutforming: Håndbok N100. Vegdirektoratet; 2019.

[9] Brusiani F, de Miranda S, Patruno L, Ubertini F, Vaona P. On the evaluation of bridge deck flutter derivatives using RANS turbulence models. Journal of Wind Engineering Industrial Aerodynamics. 2013;119:39–47.

[10] Øiseth O, et al.. Updated input for STEP 1.2. IABSE WG 10 Super-long Span Bridge Aerodynamics; 2017.

[11] ABAQUS Scripting Reference Guide, Version 6.14. Simulia; 2014.

[12] Statens vegvesen. Hardanger bridge information; 2011.

A Design wind velocity of Langenuen bridge

The following expressions are taken from NS-EN 1991-1-4:2005+NA:2009. The mean wind velocity is given by

$$V_m(z) = C_r(z) \cdot C_0(z) \cdot V_b \tag{A.1}$$

where $C_r(z)$ is the roughness factor, $C_0(z)$ is the orography factor, V_b is the basic wind velocity, and z is the height above terrain for the given structure.

$$C_r(z) = k_r \cdot ln\left(\frac{z}{z_0}\right) \quad \text{for} \quad z_{min} \leq z \leq z_{max} \tag{A.2}$$

It is assumed that the bridge surroundings reflect a terrain category 1, which give k_r=0.17 and z_0=0.01. It is also assumed that the height of the bridge girder is approximately 80 meters.

The basic wind velocity is given by

$$V_b = c_{dir} \cdot c_{season} \cdot c_{alt} \cdot c_{prob} \cdot V_{b,0} \tag{A.3}$$

Here, c_{dir} = c_{season} = c_{alt} = 1.0. Given a return period of 50 years, c_{prob} = 1.0. The bridge will be crossing Tysnes and Fitjar/Stord kommune which all have a reference wind velocity of 26 m/s. Thus

$$V_b = 1.0 \cdot 1.0 \cdot 1.0 \cdot 1.0 \cdot 26 = 26\text{m/s} \tag{A.4}$$

The formula for the design wind velocity is given in Handbook N400, chapter 5.4.3.8

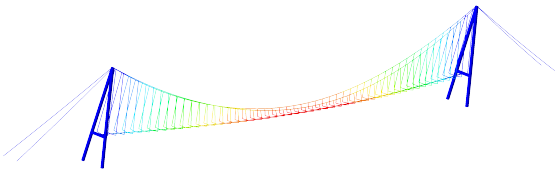
$$\frac{V_{crit}}{\gamma_{v_{crit}}} \geq V_m \quad = (z = z_m, T = 600, R = 500) \tag{A.5}$$

Inserting values the critical wind velocity for Langenuen bridge is

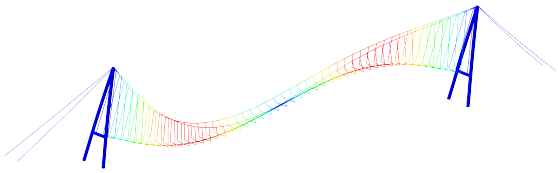
$$V_{crit} \geq \gamma_{v_{crit}} \cdot V_m = 1.6 \cdot 0.17 \cdot ln\left(\frac{80}{0.01}\right) \cdot 1.0 \cdot 26 = \underline{\underline{63.6\text{m/s}}} \tag{A.6}$$

B Mode shapes from Abaqus

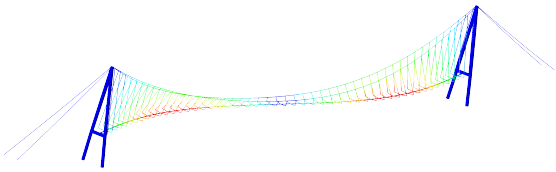
The mode shapes presented below are the ones for the aluminium option, given in table 6.



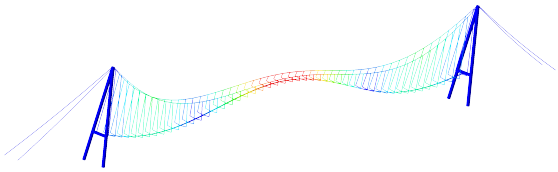
(a) Hs1, $f = 0.053$ Hz



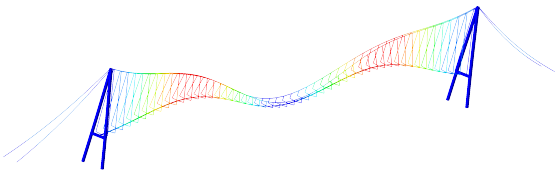
(b) Va1, $f = 0.11$ Hz



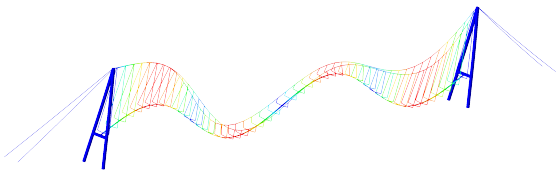
(c) Ha1, $f = 0.13$ Hz



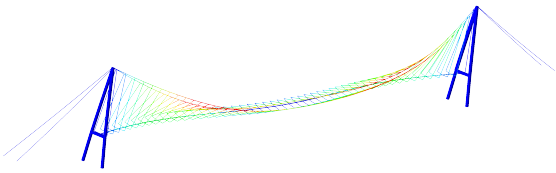
(d) Vs1, $f = 0.14$ Hz



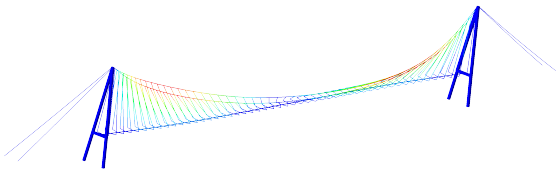
(e) Vs2, $f = 0.20$ Hz



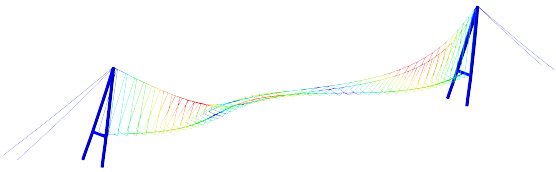
(f) Va2, $f = 0.21$ Hz



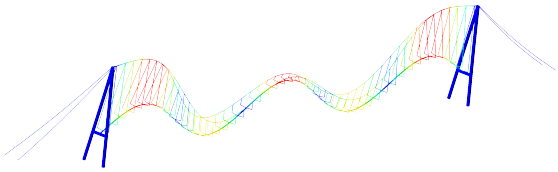
(g) Hs2, $f = 0.25$ Hz



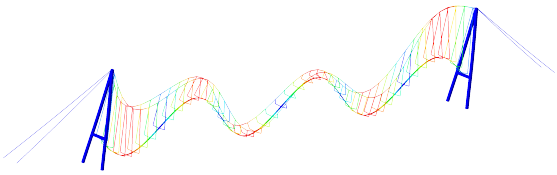
(h) Ha2, $f = 0.26$ Hz



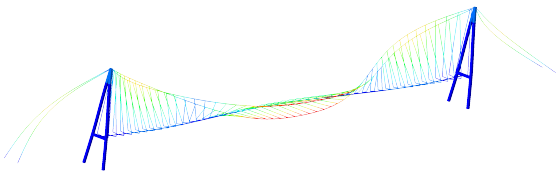
(i) Hs3, $f = 0.27$ Hz



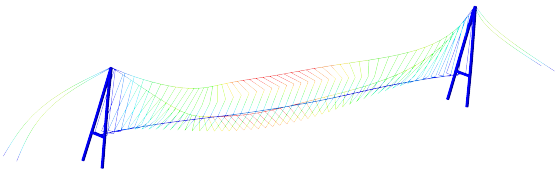
(j) Vs3, $f = 0.28$ Hz



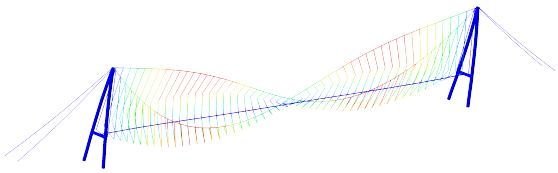
(k) Va3, $f = 0.34$ Hz



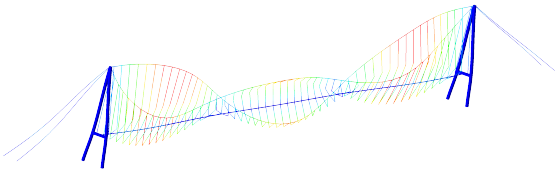
(l) Hs4, $f = 0.39$ Hz



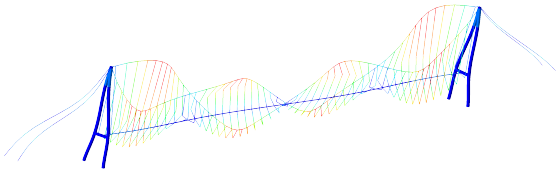
(m) Ts1, $f = 0.39$ Hz



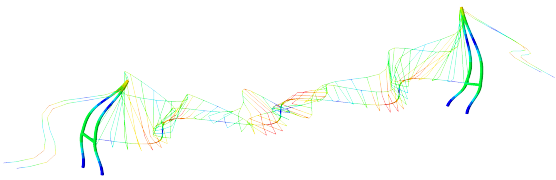
(n) Ta1, $f = 0.45$ Hz



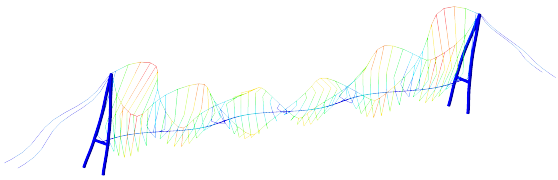
(o) Ts2, $f = 0.68$ Hz



(p) Ta2, $f = 0.88$ Hz



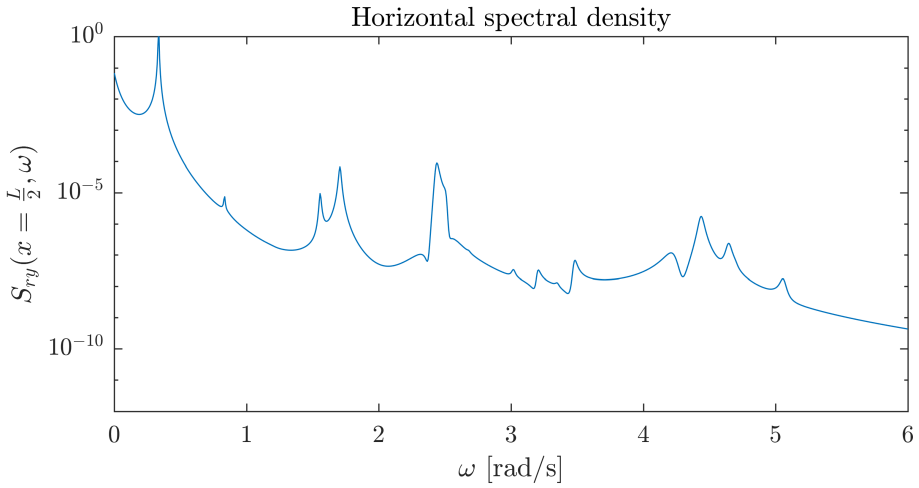
(q) Ts3, $f = 1.11$ Hz



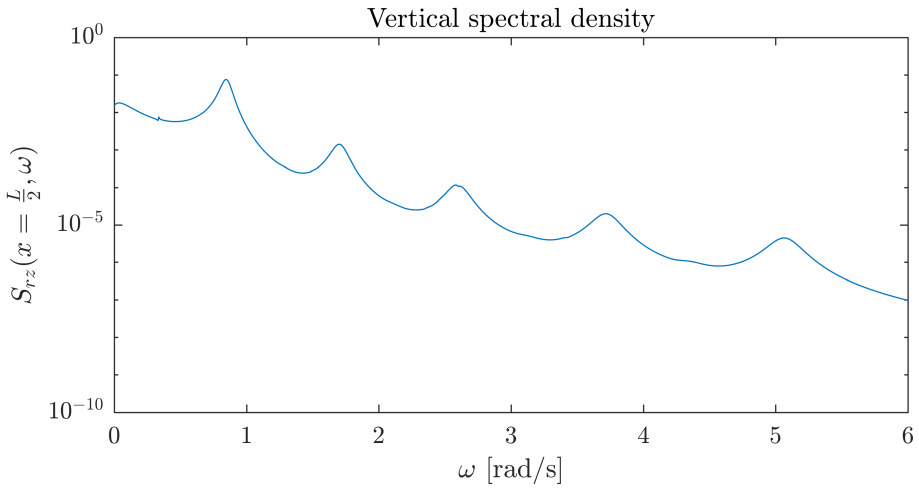
(r) Ta3, $f = 1.32$ Hz

C Response spectral densities

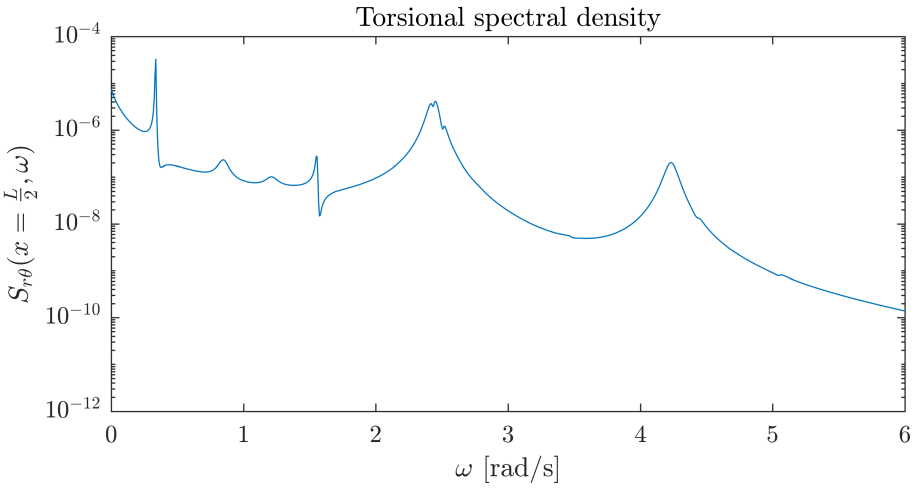
C.1 Aluminium option



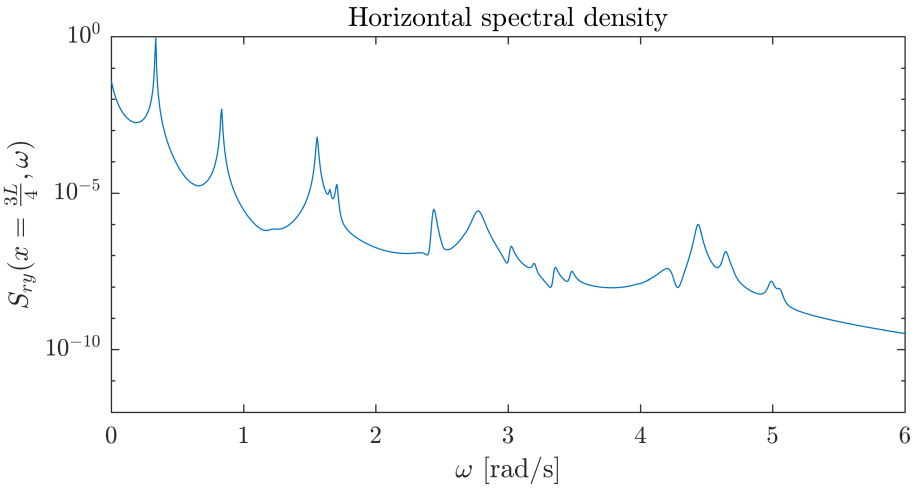
(a) Horizontal response spectrum at midspan



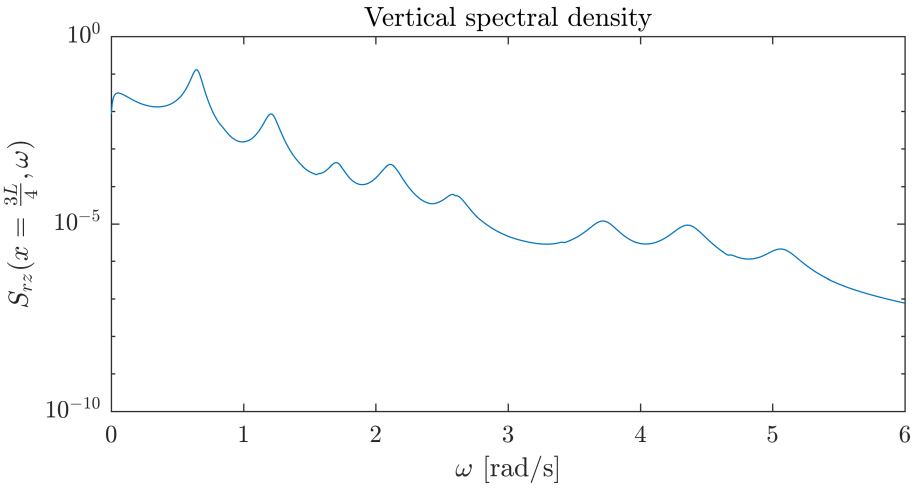
(b) Vertical response spectrum at midspan



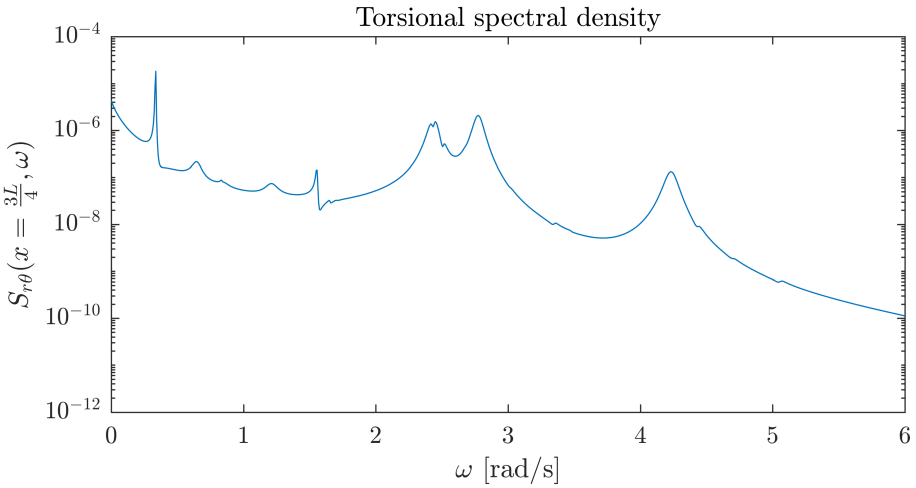
(c) Torsional response spectrum at midspan



(d) Horizontal response spectrum in the three-quarter span

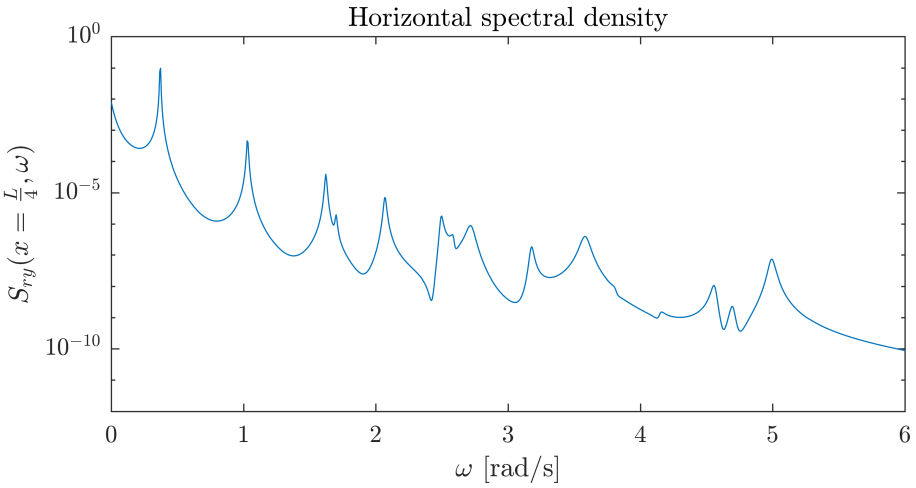


(e) Vertical response spectrum in the three-quarter span

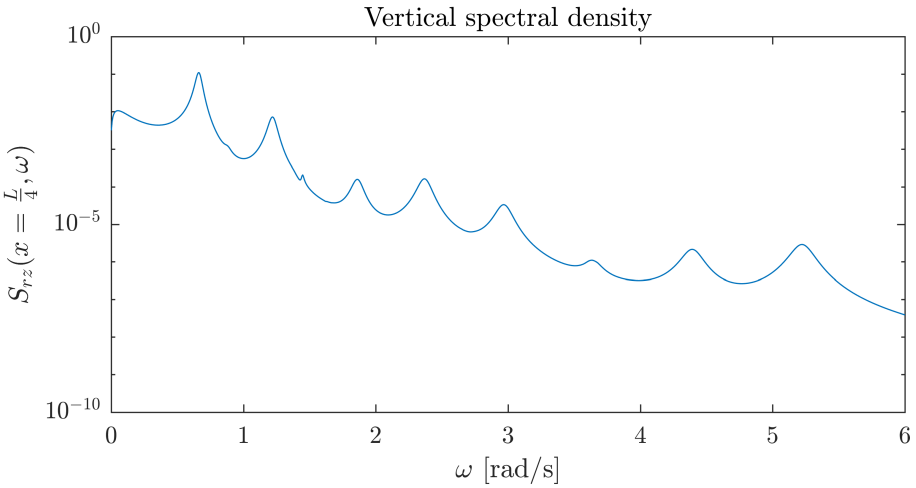


(f) Torsional response spectrum in the three-quarter span

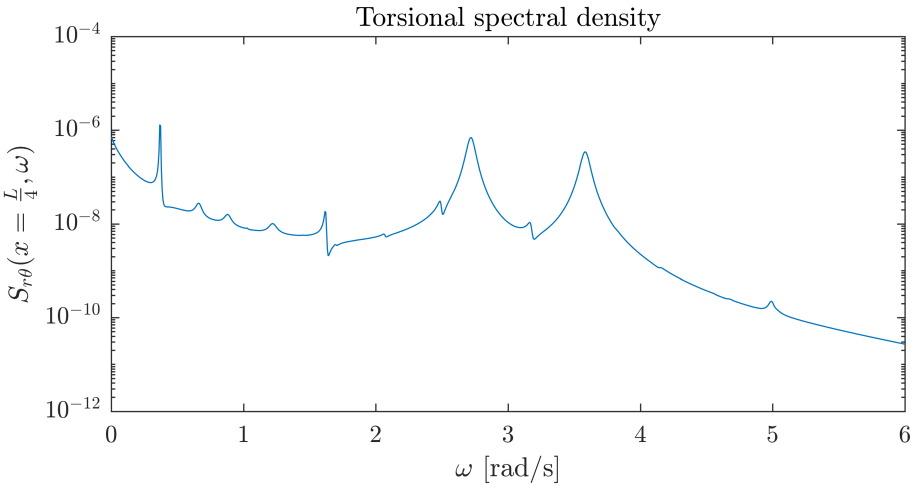
C.2 Steel option



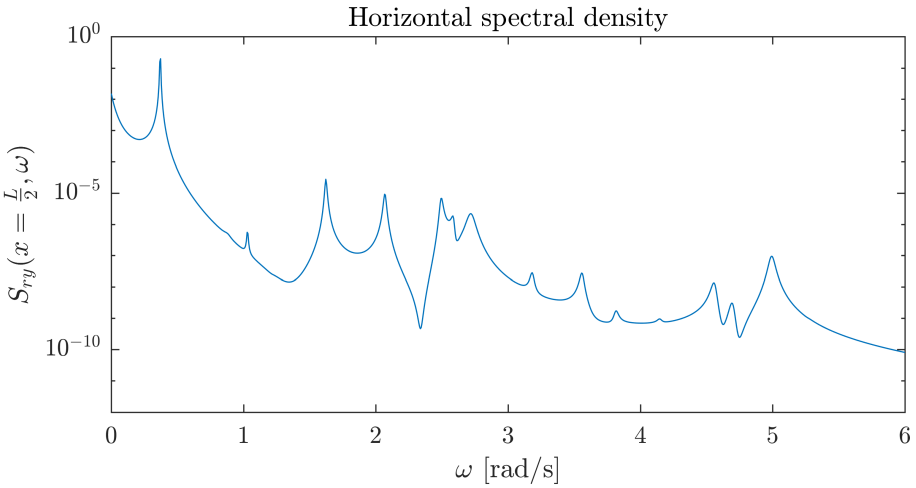
(a) Horizontal response spectrum in the quarter span



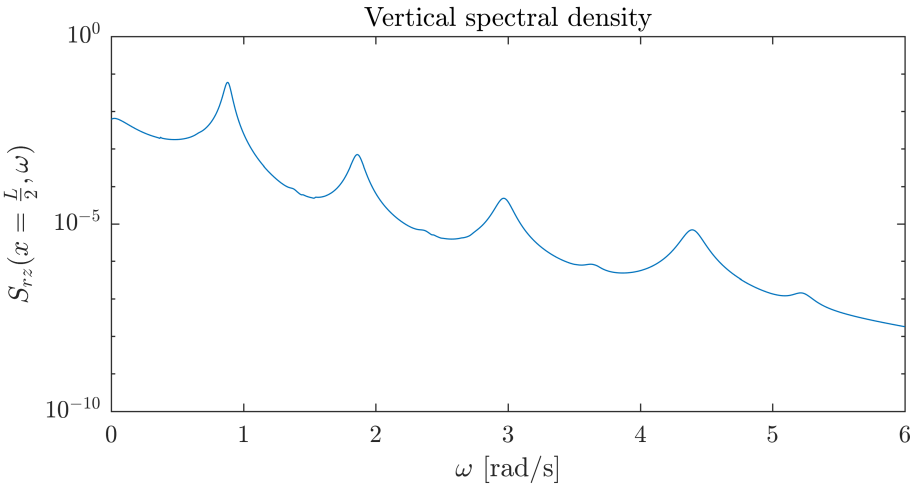
(b) Vertical response spectrum in the quarter span



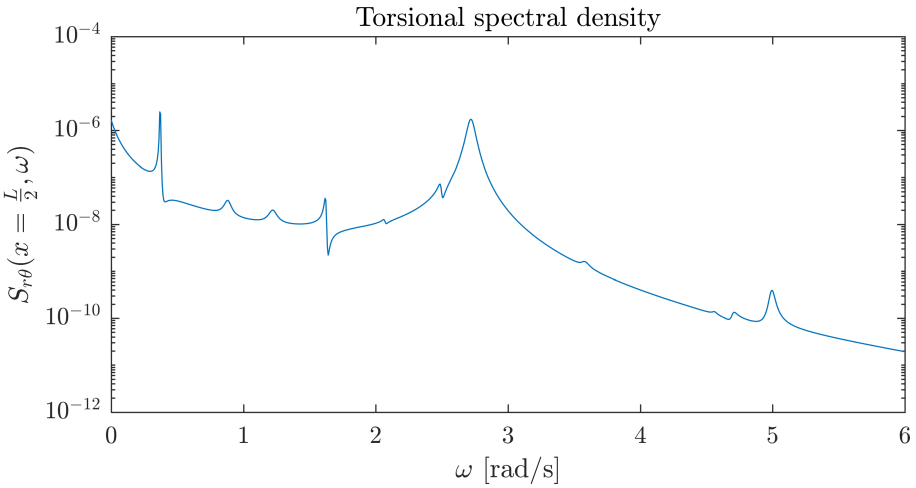
(c) Torsional response spectrum in the quarter span



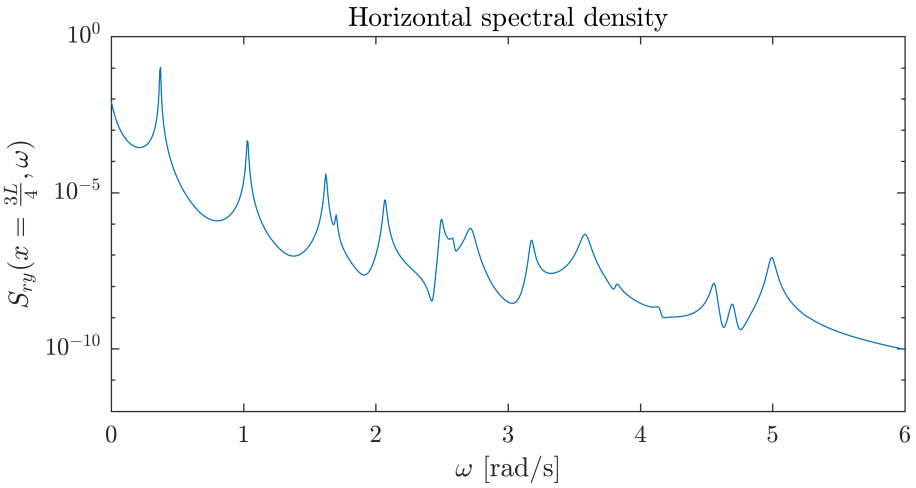
(d) Horizontal response spectrum at midspan



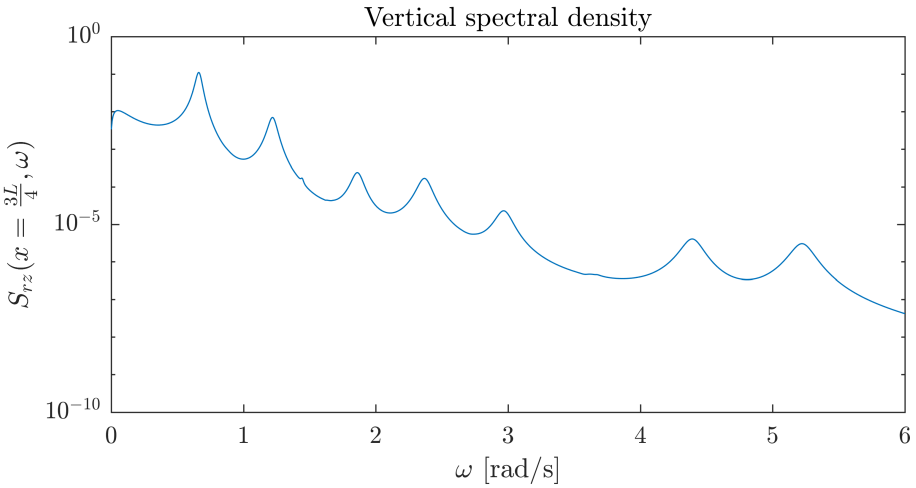
(e) Vertical response spectrum at midspan



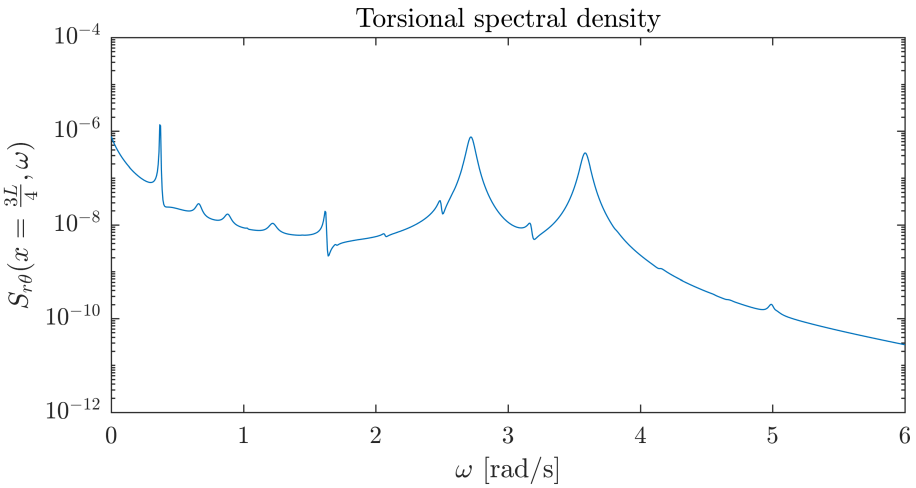
(f) Torsional response spectrum at midspan



(g) Horizontal response spectrum in the three-quarter span



(h) Vertical response spectrum in the three-quarter span



(i) Torsional response spectrum in the three-quarter span

D Python script to generate the FE model

```
# ------------------------------------------------------------------------------------------------
# ----------------------------------  FOLDER DEFINITIONS -----------------------------------------
# ------------------------------------------------------------------------------------------------
from abaqus import *
from abaqusConstants import *
import __main__
import section
import regionToolset
import displayGroupMdbToolset as dgm
import part
import material
import assembly
import step
import interaction
import load
import mesh
import optimization
import job
import sketch
import visualization
import xyPlot
import displayGroupOdbToolset as dgo
import connectorBehavior
import numpy as np
import math

# ------------------------------------------------------------------------------------------------
# ----------------------------------  INPUT VALUES -----------------------------------------------
# ------------------------------------------------------------------------------------------------
# CABLE GEOMETRY [m]
a = 290.0                       # Horizontal distance between left tower and anchorage
b = 1220.0                      # Length of middle span
c = 290.0                       # Horizontal distance between right tower and anchorage
d = a/1.8                       # Height between left tower and anchorage
e = 122.0                       # Height from tower to midspan of cables
f = c/1.8                       # Height between right tower and anchorage
g = 34.0                        # Width between cables at left anchorages
h = 29.5                        # Width between cables at midspan
i = 34.0                        # Width between cables at right anchorages
k = 1.6                         # Width between cables at tower tops
# TOWER GEOMETRY [m]
gap= 12.0                       # Vertical distance between cables and girder at the midspan
j= 70.0                         # Vertical distance between girder and bottom of towers
# GIRDER GEOMETRY [m]
vert_curv= 7.5                  # Height between lowest and highest parts of girder.
                                # NB must be either .0 or .5 due to tower list
# NODE SPACING (x-dir) [m]
spacing_side_span = 25.0        # Horizontal spacing of cable nodes at outer part of towers
spacing_mid_span = 24.0         # Horizontal spacing of cable nodes(hangers) at the midspan
spacing_girder = 4.0            # Horizontal spacing of girder nodes
# MESH SEED SIZE [m]
sz_towers = 500.0
sz_rigels = 20.0
sz_cables = 500.0
```

```
sz_girder = 10.0
sz_hangers = 500000.0
sz_coneltowers = 500.0
sz_conelgirder = 500.0
# GIRDER MATERIAL PROPERTIES
# Densitygirder = 7850.0        # STEEL
# Ematgirder = 210e9            # STEEL
Densitygirder = 2700.0          # ALUMINUM
Ematgirder = 70e9               # ALUMINUM
# NB maa oppdatere density ogsaa i keywordBlock.replace

# ------------------------------------------------------------------------------------------------
# ----------------------------- GENERATING LISTS OF COORDINATES ----------------------------------
# ------------------------------------------------------------------------------------------------
# CABLES
n_divisions_mid = b/spacing_mid_span
n_divisions_side1 = a/spacing_side_span
n_divisions_side2 = c/spacing_side_span

x1 = np.arange(-b/2, b/2, spacing_mid_span)
if n_divisions_mid%1 == 0:
    pass
else:
    delay = (b - math.floor(b / spacing_mid_span) * spacing_mid_span) * 0.5
    x1 = x1 + delay
    x1 = np.append(-b/2, x1)
    x1 = np.append(x1, b/2)
y1 = -(2*(h-k)/(b*b))*x1*x1+h/2
z1 = (4*e/(b*b))*x1*x1

x2 = np.arange(-b/2, b/2, spacing_mid_span)
if n_divisions_mid%1 == 0:
    pass
else:
    delay = (b - math.floor(b / spacing_mid_span) * spacing_mid_span) * 0.5
    x2 = x2 + delay
    x2 = np.append(-b/2, x2)
    x2 = np.append(x2, b/2)
y2 = (2*(h-k)/(b*b))*x2*x2-h/2
z2 = (4*e/(b*b))*x2*x2

x3 = np.arange(-(0.5*b+a), (-0.5*b+1), spacing_side_span)
if n_divisions_side1%1 == 0:
        pass
else:
        delay_side1 = (a - math.floor(a/spacing_side_span) * spacing_side_span)
        x3 = x3 + delay_side1
        x3 = np.append(-(0.5*b+a), x3)
y3 = -(g-k)/(2*a)*x3-(b*(g-k)/(4*a))+(k/2)
z3 = d/a*x3+(e+b*d/(2*a))

x4 = np.arange(-(0.5*b+a), (-0.5*b+1), spacing_side_span)
if n_divisions_side1%1 == 0:
        pass
else:
        delay_side1 = (a - math.floor(a/spacing_side_span) * spacing_side_span)
```

```
        x4 = x4 + delay_side1
        x4 = np.append(-(0.5*b+a), x4)
y4 = (g-k)/(2*a)*x4+(b*(g-k)/(4*a))-(k/2)
z4 = d/a*x4+(e+b*d/(2*a))

x5 = np.arange((0.5*b), (0.5*b+c+1), spacing_side_span)
if n_divisions_side2%1 == 0:
        pass
else:
        x5 = np.append(x5, (0.5*b+c))
y5 = (i-k)/(2*c)*x5-(b*(i-k)/(4*c))+(k/2)
z5 = -f/c*x5+(e+b*f/(2*c))

x6 = np.arange((0.5*b), (0.5*b+c+1), spacing_side_span)
if n_divisions_side2%1 == 0:
        pass
else:
        x6 = np.append(x6, (0.5*b+c))
y6 = -(i-k)/(2*c)*x6+b*(i-k)/(4*c)-(k/2)
z6 = -f/c*x6+(e+b*f/(2*c))

# Join into two main cables
x3 = np.delete(x3, -1)
x1 = np.delete(x1, -1)
x1_main = np.append(x3, x1)
x1_main = np.append(x1_main, x5)  # x-vector cable 1
y3 = np.delete(y3, -1)
y1 = np.delete(y1, -1)
y1_main = np.append(y3, y1)
y1_main = np.append(y1_main, y5)  # y-vector cable 1
z3 = np.delete(z3, -1)
z1 = np.delete(z1, -1)
z1_main = np.append(z3, z1)
z1_main = np.append(z1_main, z5)  # z-vector cable 1
x4 = np.delete(x4, -1)
x2 = np.delete(x2, -1)
x2_main = np.append(x4, x2)
x2_main = np.append(x2_main, x6)  # x-vector cable 2
y4 = np.delete(y4, -1)
y2 = np.delete(y2, -1)
y2_main = np.append(y4, y2)
y2_main = np.append(y2_main, y6)  # y-vector cable 2
z4 = np.delete(z4, -1)
z2 = np.delete(z2, -1)
z2_main = np.append(z4, z2)
z2_main = np.append(z2_main, z6)  # z-vector cable 2

cablelist1 = list(zip(x1_main, y1_main, z1_main))
cablelist2 = list(zip(x2_main, y2_main, z2_main))

# PYLONS/TOWERS
x7=np.linspace(-b/2,-b/2,(204*2+1))
x8=np.linspace(b/2,b/2,(204*2+1))
z7=np.linspace(-j-gap,e,(204*2+1))
y7=h/(2*e)*z7-h/2
y8=-h*z7/(2*e)+h/2
```

```
pylon1_x = np.append(x7, np.delete(x7, -1))
pylon1_y = np.append(y7, np.delete(y8[::-1],0))
pylon1_z = np.append(z7, np.delete(z7[::-1],0))
pylon1 = list(zip(pylon1_x, pylon1_y, pylon1_z))

pylon2_x = np.append(x8, np.delete(x8, -1))
pylon2_y = np.append(y7, np.delete(y8[::-1],0))
pylon2_z = np.append(z7, np.delete(z7[::-1],0))
pylon2 = list(zip(pylon2_x, pylon2_y, pylon2_z))

# GIRDER
x9 = np.arange(-b/2+2.0, (b/2)+1, spacing_girder)
n_divisions_girder = b/spacing_girder

if n_divisions_girder%1 == 0:
    pass
else:
    delay = (b - math.floor(b / spacing_girder) * spacing_girder) * 0.5
    x9 = x9 + delay
    x9 = np.append(-b/2, x9)
    x9 = np.append(x9, b/2)

x9 = np.append(-b/2, x9)
x9 = np.append(x9, b/2)
y9=np.linspace(0,0,len(x9))
z9=-vert_curv/((b/2)**2)*(x9**2)-gap

deck = list(zip(x9, y9, z9))

# ------------------------------------------------------------------------------------------------
# ------------------------------------- INITIALIZATION -------------------------------------------
# ------------------------------------------------------------------------------------------------
bridge = mdb.models['Model-1']
myAssembly = bridge.rootAssembly
myAssembly.DatumCsysByDefault(CARTESIAN)

# ------------------------------------------------------------------------------------------------
# ------------------------------------- TOWERS ---------------------------------------------------
# ------------------------------------------------------------------------------------------------
# PART CREATION
bridge.Part(name='Tower1', dimensionality=THREE_D, type=DEFORMABLE_BODY)
tower1 = bridge.parts['Tower1']
tower1.WirePolyLine(points=pylon1, mergeType=SEPARATE, meshable=ON)

bridge.Part(name='Tower2', dimensionality=THREE_D, type=DEFORMABLE_BODY)
tower2 = bridge.parts['Tower2']
tower2.WirePolyLine(points=pylon2, mergeType=SEPARATE, meshable=ON)

# PART SETS
set_tower1 = bridge.parts['Tower1'].Set(name = 'Tower1 set', edges = tower1.edges)
set_tower2 = bridge.parts['Tower2'].Set(name = 'Tower2 set', edges = tower2.edges)

# SECTION ASSIGNMENT
bridge.BoxProfile(name='profileTowers', a = 5.0, b= 5.0, uniformThickness = ON, t1 = 0.5)
Emat2=36000e6
```

```
Gmat2=Emat2/(2*(1+0.2))
my2ndsection = bridge.BeamSection(name = 'sectionTowers', profile = 'profileTowers',
poissonRatio=0.2, integration=BEFORE_ANALYSIS,
        table=((Emat2, Gmat2), ), alphaDamping=0.0, beamShape=CONSTANT,
    betaDamping=0.0, centroid=(0.0, 0.0), compositeDamping=0.0,
        consistentMassMatrix=False, dependencies=0, shearCenter=(0.0, 0.0),
        temperatureDependency=OFF, thermalExpansion=OFF, density = 2400.0)

bridge.parts['Tower1'].SectionAssignment(region = set_tower1, sectionName = 'sectionTowers')
bridge.parts['Tower1'].assignBeamSectionOrientation(region = set_tower1, method = N1_COSINES,
    n1 = (0.0, 0.0, -1.0))
bridge.parts['Tower2'].SectionAssignment(region = set_tower2, sectionName = 'sectionTowers')
bridge.parts['Tower2'].assignBeamSectionOrientation(region = set_tower2, method = N1_COSINES,
    n1 = (0.0, 0.0, -1.0))

# ASSEMBLY
inst_tower1 = myAssembly.Instance(name='Tower1 instance', part=tower1, dependent=ON)
inst_tower2 = myAssembly.Instance(name='Tower2 instance', part=tower2, dependent=ON)

# BCs
vert_tower1_bottoms = inst_tower1.vertices.findAt(((pylon1[0]), ), ((pylon1[-1]), ),)
vert_tower2_bottoms = inst_tower2.vertices.findAt(((pylon2[0]), ), ((pylon2[-1]), ),)

set_vert_tower1_bottoms = myAssembly.Set(name = 'setVertices Tower1_bottoms',
    vertices = vert_tower1_bottoms)
set_vert_tower2_bottoms = myAssembly.Set(name = 'setVertices Tower2_bottoms',
    vertices = vert_tower2_bottoms)

bridge.EncastreBC(createStepName = 'Initial', localCsys = None, name = 'towersbottomsBC1',
    region = set_vert_tower1_bottoms)
bridge.EncastreBC(createStepName = 'Initial', localCsys = None, name = 'towersbottomsBC2',
    region = set_vert_tower2_bottoms)

# MESH
tower1.seedPart(size=sz_towers, deviationFactor=0.01, minSizeFactor=0.01)
elemType1=mesh.ElemType(elemCode=B33)
tower1.setElementType(regions=set_tower1, elemTypes=(elemType1,))
tower1.generateMesh()

tower2.seedPart(size=sz_towers, deviationFactor=0.01, minSizeFactor=0.01)
elemType1=mesh.ElemType(elemCode=B33)
tower2.setElementType(regions=set_tower2, elemTypes=(elemType1,))
tower2.generateMesh()

# ------------------------------------------------------------------------------------------
# ------------------------------------ RIGELS ----------------------------------------------
# ------------------------------------------------------------------------------------------
blabla = np.argwhere(z7 == -gap-vert_curv)
punkt1_tower1 = tower1.vertices[blabla[0][0]].pointOn[0]
punkt2_tower1 = tower1.vertices[len(pylon1_z)-1-blabla[0][0]].pointOn[0]
punkt1_tower2 = tower2.vertices[blabla[0][0]].pointOn[0]
punkt2_tower2 = tower2.vertices[len(pylon2_z)-1-blabla[0][0]].pointOn[0]

# PART CREATION
bridge.Part(name='Rigel1', dimensionality=THREE_D, type=DEFORMABLE_BODY)
rigel1 = bridge.parts['Rigel1']
```

```
rigel1.WirePolyLine(points = (punkt1_tower1, (-0.5*b, 0, -gap-vert_curv), punkt2_tower1),
    mergeType = IMPRINT, meshable = ON)

bridge.Part(name='Rigel2', dimensionality=THREE_D, type=DEFORMABLE_BODY)
rigel2 = bridge.parts['Rigel2']
rigel2.WirePolyLine(points = (punkt1_tower2, (0.5*b, 0, -gap-vert_curv), punkt2_tower2),
    mergeType = IMPRINT, meshable = ON)

# PART SETS
set_rigel1 = bridge.parts['Rigel1'].Set(name = 'Rigel1 set', edges = rigel1.edges)
set_rigel2 = bridge.parts['Rigel2'].Set(name = 'Rigel2 set', edges = rigel2.edges)

# SECTION ASSIGNMENT
bridge.BoxProfile(name='profileRigels', a = 5.0, b= 5.0, uniformThickness = ON, t1 = 0.5)
Emat2=36000e6
Gmat2=Emat2/(2*(1+0.2))
my2ndsection = bridge.BeamSection(name = 'sectionRigels', profile = 'profileRigels',
    poissonRatio=0.2, integration=BEFORE_ANALYSIS,
        table=((Emat2, Gmat2), ), alphaDamping=0.0, beamShape=CONSTANT,
    betaDamping=0.0, centroid=(0.0, 0.0), compositeDamping=0.0,
        consistentMassMatrix=False, dependencies=0, shearCenter=(0.0, 0.0),
        temperatureDependency=OFF, thermalExpansion=OFF, density = 2400.0)

bridge.parts['Rigel1'].SectionAssignment(region = set_rigel1, sectionName = 'sectionRigels')
bridge.parts['Rigel1'].assignBeamSectionOrientation(region = set_rigel1, method = N1_COSINES,
    n1 = (0.0, 0.0, -1.0))
bridge.parts['Rigel2'].SectionAssignment(region = set_rigel2, sectionName = 'sectionRigels')
bridge.parts['Rigel2'].assignBeamSectionOrientation(region = set_rigel2, method = N1_COSINES,
    n1 = (0.0, 0.0, -1.0))

# ASSEMBLY
inst_rigel1 = myAssembly.Instance(name='Rigel1 instance', part=rigel1, dependent=ON)
inst_rigel2 = myAssembly.Instance(name='Rigel2 instance', part=rigel2, dependent=ON)

# MESH
rigel1.seedPart(size=sz_rigels, deviationFactor=0.01, minSizeFactor=0.01)
elemType1=mesh.ElemType(elemCode=B33)
rigel1.setElementType(regions=set_rigel1, elemTypes=(elemType1,))
rigel1.generateMesh()

rigel2.seedPart(size=sz_rigels, deviationFactor=0.01, minSizeFactor=0.01)
elemType1=mesh.ElemType(elemCode=B33)
rigel2.setElementType(regions=set_rigel2, elemTypes=(elemType1,))
rigel2.generateMesh()

# ---------------------------------------------------------------------------------------------
# ------------------------------------ CABLES -------------------------------------------------
# ---------------------------------------------------------------------------------------------
# PART CREATION
bridge.Part(name='Cable1', dimensionality=THREE_D, type=DEFORMABLE_BODY)
cable1 = bridge.parts['Cable1']
cable1.WirePolyLine(points=cablelist1, mergeType=SEPARATE, meshable=ON)
bridge.Part(name='Cable2', dimensionality=THREE_D, type=DEFORMABLE_BODY)
cable2 = bridge.parts['Cable2']
cable2.WirePolyLine(points=cablelist2, mergeType=SEPARATE, meshable=ON)
```

```
# PART SETS
set_cable1 = bridge.parts['Cable1'].Set(name = 'Cable1 set', edges = cable1.edges)
set_cable2 = bridge.parts['Cable2'].Set(name = 'Cable2 set', edges = cable2.edges)

# SECTION ASSIGNMENT
bridge.GeneralizedProfile(name='profileCables', area=0.2376, i11=0.009, i12=0.0, i22=0.009,
    j=0.017967, gammaO=0.0,  gammaW=0.0)
Emat=200e9
Gmat=Emat/(2*(1+0.3))
mySection=bridge.BeamSection(name='sectionCables', profile='profileCables',
    poissonRatio=0.3, integration=BEFORE_ANALYSIS,
        table=((Emat, Gmat), ), alphaDamping=0.0, beamShape=CONSTANT,
    betaDamping=0.0, centroid=(0.0, 0.0), compositeDamping=0.0,
        consistentMassMatrix=False, dependencies=0, shearCenter=(0.0, 0.0),
        temperatureDependency=OFF, thermalExpansion=OFF, density = 7850.0)

bridge.parts['Cable1'].SectionAssignment(region = set_cable1, sectionName = 'sectionCables')
bridge.parts['Cable1'].assignBeamSectionOrientation(region = set_cable1, method = N1_COSINES,
    n1 = (0.0, 1.0, 0.0))
bridge.parts['Cable2'].SectionAssignment(region = set_cable2, sectionName = 'sectionCables')
bridge.parts['Cable2'].assignBeamSectionOrientation(region = set_cable2, method = N1_COSINES,
    n1 = (0.0, 1.0, 0.0))

# ASSEMBLY
inst_cable1 = myAssembly.Instance(name='Cable1 instance', part=cable1, dependent=ON)
inst_cable2 = myAssembly.Instance(name='Cable2 instance', part=cable2, dependent=ON)

# BCs
vert_cable1_anchorage = inst_cable1.vertices.findAt(((cablelist1[0]), ), ((cablelist1[-1]), ),)
vert_cable2_anchorage = inst_cable2.vertices.findAt(((cablelist2[0]), ), ((cablelist2[-1]), ),)
vert_cable1_midspan = inst_cable1.vertices.findAt(((x1[np.argmin(z1)], y1[np.argmin(z1)],
    z1[np.argmin(z1)]), ),)
vert_cable2_midspan = inst_cable2.vertices.findAt(((x2[np.argmin(z2)], y2[np.argmin(z2)],
    z2[np.argmin(z2)]), ),)

set_vert_cable1_anchorage = myAssembly.Set(name = 'setVertices Cable1_Anchorage',
    vertices = vert_cable1_anchorage)
set_vert_cable1_midspan = myAssembly.Set(name = 'setVertices Cable1_Midspan',
    vertices = vert_cable1_midspan)
set_vert_cable2_anchorage = myAssembly.Set(name = 'setVertices Cable2_Anchorage',
    vertices = vert_cable2_anchorage)
set_vert_cable2_midspan = myAssembly.Set(name = 'setVertices Cable2_Midspan',
    vertices = vert_cable2_midspan)

bridge.PinnedBC(createStepName = 'Initial', localCsys = None, name = 'anchorageBC1',
    region = set_vert_cable1_anchorage)
constraint_midspan1 = bridge.DisplacementBC(createStepName = 'Initial', name = 'midspanBC1',
    region = set_vert_cable1_midspan, u2 = 0)
bridge.PinnedBC(createStepName = 'Initial', localCsys = None, name = 'anchorageBC2',
    region = set_vert_cable2_anchorage)
constraint_midspan2 = bridge.DisplacementBC(createStepName = 'Initial', name = 'midspanBC2',
    region = set_vert_cable2_midspan, u2 = 0)

# MESH
cable1.seedPart(size=sz_cables, deviationFactor=0.01, minSizeFactor=0.01)
elemType1=mesh.ElemType(elemCode=B33)
```

```
cable1.setElementType(regions=set_cable1, elemTypes=(elemType1,))
cable1.generateMesh()

cable2.seedPart(size=sz_cables, deviationFactor=0.01, minSizeFactor=0.01)
elemType1=mesh.ElemType(elemCode=B33)
cable2.setElementType(regions=set_cable2, elemTypes=(elemType1,))
cable2.generateMesh()

# ------------------------------------------------------------------------------------------
# ------------------------------------ GIRDER ----------------------------------------------
# ------------------------------------------------------------------------------------------
# PART CREATION
bridge.Part(name='Girder', dimensionality=THREE_D, type=DEFORMABLE_BODY)
girder = bridge.parts['Girder']
girder.WirePolyLine(points=deck, mergeType=SEPARATE, meshable=ON)

# PART SETS
set_girder = bridge.parts['Girder'].Set(name = 'Girder set', edges = girder.edges)

# SECTION ASSIGNMENT
bridge.GeneralizedProfile(name='profileGirder', area=1.82, i11=5.76, i12=0.0, i22=1.512e+2,
    j=20.33, gammaO=0.0,  gammaW=0.0)
Emat3= Ematgirder
Gmat3=Emat3/(2*(1+0.3))
mySection=bridge.BeamSection(name='sectionGirder', profile='profileGirder',
    poissonRatio=0.3, integration=BEFORE_ANALYSIS,
        table=((Emat3, Gmat3), ), alphaDamping=0.0, beamShape=CONSTANT,
    betaDamping=0.0, centroid=(0.0, 0.0), compositeDamping=0.0,
        consistentMassMatrix=False, dependencies=0, shearCenter=(0.0, 0.0),
        temperatureDependency=OFF, thermalExpansion=OFF, density = Densitygirder)

bridge.parts['Girder'].SectionAssignment(region = set_girder, sectionName = 'sectionGirder')
bridge.parts['Girder'].assignBeamSectionOrientation(region = set_girder, method = N1_COSINES,
    n1 = (0.0, 1.0, 0.0))

# ASSEMBLY
inst_girder = myAssembly.Instance(name='Girder instance', part=girder, dependent=ON)

# BCs
vert_girder_ends = inst_girder.vertices.findAt(((deck[0]), ), ((deck[-1]), ),)
set_vert_girder_ends = myAssembly.Set(name = 'setVertices Girder_Ends',
    vertices = vert_girder_ends)
# bridge.DisplacementBC(createStepName = 'Initial', localCsys = None, name = 'girderBCs',
    region = set_vert_girder_ends, u1 = 0, u2 = 0, u3 = 0, ur1 = 0)

# MESH
girder.seedPart(size=sz_girder, deviationFactor=0.01, minSizeFactor=0.01)
elemType1=mesh.ElemType(elemCode=B32)
girder.setElementType(regions=set_girder, elemTypes=(elemType1,))
girder.generateMesh()

# ------------------------------------------------------------------------------------------
# ------------------------------------ HANGERS ---------------------------------------------
# ------------------------------------------------------------------------------------------
# PART CREATION
bridge.Part(name='Hangers', dimensionality=THREE_D, type=DEFORMABLE_BODY)
```

```
hangers = bridge.parts['Hangers']

startindex = np.argmax(z1_main)+1
sluttindex = len(z1_main)-1-np.argmax(z1_main)
for i in range(startindex, sluttindex):
        punkt2_cable1 = list(cable1.vertices[i].pointOn[0])
        punkt2_cable1[-1]= list(girder.vertices[-75+6*i].pointOn[0])[-1]+1.5
        punkt2_cable1[1]=h/2
        hangers.WirePolyLine(points = (list(cable1.vertices[i].pointOn[0]), punkt2_cable1),
            mergeType = IMPRINT, meshable = ON)

        punkt2_cable2 = list(cable2.vertices[i].pointOn[0])
        punkt2_cable2[-1]= list(girder.vertices[-75+6*i].pointOn[0])[-1]+1.5
        punkt2_cable2[1]=-h/2
        hangers.WirePolyLine(points = (list(cable2.vertices[i].pointOn[0]), punkt2_cable2),
            mergeType = IMPRINT, meshable = ON)

# PART SETS
set_hangers = bridge.parts['Hangers'].Set(name = 'Hangers set', edges = hangers.edges)

# SECTION ASSIGNMENT
bridge.GeneralizedProfile(name='profileHangers', area=0.0079, i11=9.82e-6, i12=0.0, i22=9.82e-6,
    j=1.96e-5, gammaO=0.0,  gammaW=0.0)
Emat=200e9
Gmat=Emat/(2*(1+0.3))
mySection=bridge.BeamSection(name='sectionHangers', profile='profileHangers',
    poissonRatio=0.3, integration=BEFORE_ANALYSIS,
        table=((Emat, Gmat), ), alphaDamping=0.0, beamShape=CONSTANT,
    betaDamping=0.0, centroid=(0.0, 0.0), compositeDamping=0.0,
        consistentMassMatrix=False, dependencies=0, shearCenter=(0.0, 0.0),
        temperatureDependency=OFF, thermalExpansion=OFF, density = 7850.0)

bridge.parts['Hangers'].SectionAssignment(region = set_hangers, sectionName = 'sectionHangers')
bridge.parts['Hangers'].assignBeamSectionOrientation(region = set_hangers, method = N1_COSINES,
    n1 = (0.0, 1.0, 0.0))

# ASSEMBLY
inst_hangers = myAssembly.Instance(name='Hangers instance', part=hangers, dependent=ON)

# MESH
hangers.seedPart(size=sz_hangers, deviationFactor=0.01, minSizeFactor=0.01)
elemType1=mesh.ElemType(elemCode=B33)
hangers.setElementType(regions=set_hangers, elemTypes=(elemType1,))
hangers.generateMesh()

myAssembly.regenerate()

# ------------------------------------------------------------------------------------------
# ------------------------------ CONNECTION ELEMENTS TOWERTOPS -----------------------------
# ------------------------------------------------------------------------------------------
# Indexes
indeks_tower1_top = np.argmax(pylon1_z)
indeks_tower2_top = np.argmax(pylon2_z)
indeks_tower1_cable1_top = len(z1_main)-1-np.argmax(z1_main)
indeks_tower1_cable2_top = len(z2_main)-1-np.argmax(z2_main)
indeks_tower2_cable1_top = np.argmax(z1_main)
```

```
indeks_tower2_cable1_top = np.argmax(z2_main)

# PART CREATION
bridge.Part(name='ConelTowers', dimensionality=THREE_D, type=DEFORMABLE_BODY)
coneltowers = bridge.parts['ConelTowers']
coneltowers.WirePolyLine(points = (cable1.vertices[indeks_tower1_cable1_top].pointOn[0],
    tower1.vertices[indeks_tower1_top].pointOn[0],
    cable2.vertices[indeks_tower1_cable2_top].pointOn[0]), mergeType = IMPRINT, meshable = ON)
coneltowers.WirePolyLine(points = (cable1.vertices[indeks_tower2_cable1_top].pointOn[0],
    tower2.vertices[indeks_tower2_top].pointOn[0],
    cable2.vertices[indeks_tower2_cable1_top].pointOn[0]), mergeType = IMPRINT, meshable = ON)

# PART SETS
set_coneltowers = bridge.parts['ConelTowers'].Set(name = 'ConelTowers set',
    edges = coneltowers.edges)

# SECTION ASSIGNMENT
bridge.GeneralizedProfile(name='profileConelTowers', area=0.22, i11=1000.0, i12=0.0, i22=1000.0,
    j=1000.0, gammaO=0.0,  gammaW=0.0)
Emat3=2.00000e+11
Gmat3=Emat3/(2*(1+0.3))
mySection=bridge.BeamSection(name='sectionConelTowers', profile='profileConelTowers',
    poissonRatio=0.3, integration=BEFORE_ANALYSIS,
        table=((Emat3, Gmat3), ), alphaDamping=0.0, beamShape=CONSTANT,
    betaDamping=0.0, centroid=(0.0, 0.0), compositeDamping=0.0,
        consistentMassMatrix=False, dependencies=0, shearCenter=(0.0, 0.0),
        temperatureDependency=OFF, thermalExpansion=OFF, density = 0.0)

bridge.parts['ConelTowers'].SectionAssignment(region = set_coneltowers,
    sectionName = 'sectionConelTowers')
bridge.parts['ConelTowers'].assignBeamSectionOrientation(region = set_coneltowers,
    method = N1_COSINES, n1 = (0.0, 0.0, -1.0))

# ASSEMBLY
inst_coneltowers = myAssembly.Instance(name='ConelTowers instance', part=coneltowers,
    dependent=ON)

# MESH
coneltowers.seedPart(size=sz_coneltowers, deviationFactor=0.01, minSizeFactor=0.01)
elemType1=mesh.ElemType(elemCode=B33)
coneltowers.setElementType(regions=set_coneltowers, elemTypes=(elemType1,))
coneltowers.generateMesh()

# ------------------------------------------------------------------------------------------
# ----------------------------- CONNECTION ELEMENTS GIRDER -----------------------------------
# ------------------------------------------------------------------------------------------
# PART CREATION
bridge.Part(name='ConelGirder', dimensionality=THREE_D, type=DEFORMABLE_BODY)
conelgirder = bridge.parts['ConelGirder']

sluttindex = len(z1_main)-1-2*np.argmax(z1_main)
for i in range(1, sluttindex):
        conelgirder_node1 = list(hangers.vertices[-2*i+103].pointOn[0])
        conelgirder_node2 = list(girder.vertices[6*i-3].pointOn[0])
        conelgirder_node3 = list(hangers.vertices[2*i+101].pointOn[0])
        conelgirder.WirePolyLine(points = (conelgirder_node1, conelgirder_node2,
```

```
            conelgirder_node3), mergeType = IMPRINT, meshable = ON)

# PART SETS
set_conelgirder = bridge.parts['ConelGirder'].Set(name = 'ConelGirder set',
    edges = conelgirder.edges)

# SECTION ASSIGNMENT
bridge.GeneralizedProfile(name='profileConelGirder', area=0.22, i11=1000.0, i12=0.0, i22=1000.0,
    j=1000.0, gammaO=0.0,  gammaW=0.0)
Emat3=2.00000e+11
Gmat3=Emat3/(2*(1+0.3))
mySection=bridge.BeamSection(name='sectionConelGirder', profile='profileConelGirder',
    poissonRatio=0.3, integration=BEFORE_ANALYSIS,
        table=((Emat3, Gmat3), ), alphaDamping=0.0, beamShape=CONSTANT,
    betaDamping=0.0, centroid=(0.0, 0.0), compositeDamping=0.0,
        consistentMassMatrix=False, dependencies=0, shearCenter=(0.0, 0.0),
        temperatureDependency=OFF, thermalExpansion=OFF, density = 0.0)

bridge.parts['ConelGirder'].SectionAssignment(region = set_conelgirder,
    sectionName = 'sectionConelGirder')
bridge.parts['ConelGirder'].assignBeamSectionOrientation(region = set_conelgirder,
    method = N1_COSINES, n1 = (0.0, 0.0, -1.0))

# ASSEMBLY
inst_conelgirder = myAssembly.Instance(name='ConelGirders instance', part=conelgirder,
    dependent=ON)

# MESH
conelgirder.seedPart(size=sz_conelgirder, deviationFactor=0.01, minSizeFactor=0.01)
elemType1=mesh.ElemType(elemCode=B33)
conelgirder.setElementType(regions=set_conelgirder, elemTypes=(elemType1,))
conelgirder.generateMesh()

# ----------------------------------------------------------------------------------------------
# ------------------------------ TIES AT THE TOWER TOPS ----------------------------------------
# ----------------------------------------------------------------------------------------------
# Indexes
indeks_tower1_top = np.argmax(pylon1_z)
indeks_tower2_top = np.argmax(pylon2_z)
indeks_tower1_cable1_top = len(z1_main)-1-np.argmax(z1_main)
indeks_tower1_cable2_top = len(z2_main)-1-np.argmax(z2_main)
indeks_tower2_cable1_top = np.argmax(z1_main)
indeks_tower2_cable2_top = np.argmax(z2_main)

# CONNECTION ELEMENT AT TOWER 1
# Top of tower to con.el
verticesMaster = inst_tower1.vertices.findAt((
    (inst_tower1.vertices[indeks_tower1_top].pointOn[0]), ) ,)
verticesSlave = inst_coneltowers.vertices.findAt((
    (inst_tower1.vertices[indeks_tower1_top].pointOn[0]), ) ,)
setMaster = myAssembly.Set(name = 'Master set 1', vertices = verticesMaster)
setSlave = myAssembly.Set(name = 'Slave set 1', vertices = verticesSlave)
bridge.Tie(adjust=ON, master= setMaster, name='Constraint-1', positionToleranceMethod=COMPUTED,
    slave= setSlave, thickness=ON, tieRotations=ON)

# Con.el to cable1
```

```
verticesMaster = inst_coneltowers.vertices.findAt((
    (inst_cable1.vertices[indeks_tower1_cable1_top].pointOn[0]), ), )
verticesSlave = inst_cable1.vertices.findAt((
    (inst_cable1.vertices[indeks_tower1_cable1_top].pointOn[0]), ) ,)
setMaster = myAssembly.Set(name = 'Master set 2', vertices = verticesMaster)
setSlave = myAssembly.Set(name = 'Slave set 2', vertices = verticesSlave)
bridge.Tie(adjust=ON, master= setMaster, name='Constraint-2', positionToleranceMethod=COMPUTED,
    slave= setSlave, thickness=ON, tieRotations=ON)

# Con.el to cable2
verticesMaster = inst_coneltowers.vertices.findAt((
    (inst_cable2.vertices[indeks_tower1_cable2_top].pointOn[0]), ), )
verticesSlave = inst_cable2.vertices.findAt((
    (inst_cable2.vertices[indeks_tower1_cable2_top].pointOn[0]), ) ,)
setMaster = myAssembly.Set(name = 'Master set 3', vertices = verticesMaster)
setSlave = myAssembly.Set(name = 'Slave set 3', vertices = verticesSlave)
bridge.Tie(adjust=ON, master= setMaster, name='Constraint-3', positionToleranceMethod=COMPUTED,
    slave= setSlave, thickness=ON, tieRotations=ON)

# CONNECTION ELEMENT AT TOWER 2
# Top of tower to con.el
verticesMaster = inst_tower2.vertices.findAt((
    (inst_tower2.vertices[indeks_tower2_top].pointOn[0]), ) ,)
verticesSlave = inst_coneltowers.vertices.findAt((
    (inst_tower2.vertices[indeks_tower2_top].pointOn[0]), ) ,)
setMaster = myAssembly.Set(name = 'Master set 4', vertices = verticesMaster)
setSlave = myAssembly.Set(name = 'Slave set 4', vertices = verticesSlave)
bridge.Tie(adjust=ON, master= setMaster, name='Constraint-4', positionToleranceMethod=COMPUTED,
    slave= setSlave, thickness=ON, tieRotations=ON)

# Con.el to cable1
verticesMaster = inst_coneltowers.vertices.findAt((
    (inst_cable1.vertices[indeks_tower2_cable1_top].pointOn[0]), ), )
verticesSlave = inst_cable1.vertices.findAt((
    (inst_cable1.vertices[indeks_tower2_cable1_top].pointOn[0]), ) ,)
setMaster = myAssembly.Set(name = 'Master set 5', vertices = verticesMaster)
setSlave = myAssembly.Set(name = 'Slave set 5', vertices = verticesSlave)
bridge.Tie(adjust=ON, master= setMaster, name='Constraint-5', positionToleranceMethod=COMPUTED,
    slave= setSlave, thickness=ON, tieRotations=ON)

# Con.el to cable2
verticesMaster = inst_coneltowers.vertices.findAt((
    (inst_cable2.vertices[indeks_tower2_cable2_top].pointOn[0]), ), )
verticesSlave = inst_cable2.vertices.findAt((
    (inst_cable2.vertices[indeks_tower2_cable2_top].pointOn[0]), ) ,)
setMaster = myAssembly.Set(name = 'Master set 6', vertices = verticesMaster)
setSlave = myAssembly.Set(name = 'Slave set 6', vertices = verticesSlave)
bridge.Tie(adjust=ON, master= setMaster, name='Constraint-6', positionToleranceMethod=COMPUTED,
    slave= setSlave, thickness=ON, tieRotations=ON)

# ------------------------------------------------------------------------------------------
# ------------------------------ TIES AT THE RIGELS ----------------------------------------
# ------------------------------------------------------------------------------------------
# Rigel 1: positive y-value
verticesMaster = inst_tower1.vertices.findAt(((punkt1_tower1), ), )
verticesSlave = inst_rigel1.vertices.findAt(((punkt1_tower1), ), )
```

```
setMaster = myAssembly.Set(name = 'Master set 7', vertices = verticesMaster)
setSlave = myAssembly.Set(name = 'Slave set 7', vertices = verticesSlave)
bridge.Tie(adjust=ON, master= setMaster, name='Constraint-7', positionToleranceMethod=COMPUTED,
    slave= setSlave, thickness=ON, tieRotations=ON)

# Rigel 1: negative y-value
verticesMaster = inst_tower1.vertices.findAt(((punkt2_tower1), ), )
verticesSlave = inst_rigel1.vertices.findAt(((punkt2_tower1), ), )
setMaster = myAssembly.Set(name = 'Master set 8', vertices = verticesMaster)
setSlave = myAssembly.Set(name = 'Slave set 8', vertices = verticesSlave)
bridge.Tie(adjust=ON, master= setMaster, name='Constraint-8', positionToleranceMethod=COMPUTED,
    slave= setSlave, thickness=ON, tieRotations=ON)

# Rigel 2: positive y-value
verticesMaster = inst_tower2.vertices.findAt(((punkt1_tower2), ), )
verticesSlave = inst_rigel2.vertices.findAt(((punkt1_tower2), ), )
setMaster = myAssembly.Set(name = 'Master set 9', vertices = verticesMaster)
setSlave = myAssembly.Set(name = 'Slave set 9', vertices = verticesSlave)
bridge.Tie(adjust=ON, master= setMaster, name='Constraint-9', positionToleranceMethod=COMPUTED,
    slave= setSlave, thickness=ON, tieRotations=ON)

# Rigel 2: negative y-value
verticesMaster = inst_tower2.vertices.findAt(((punkt2_tower2), ), )
verticesSlave = inst_rigel2.vertices.findAt(((punkt2_tower2), ), )
setMaster = myAssembly.Set(name = 'Master set 10', vertices = verticesMaster)
setSlave = myAssembly.Set(name = 'Slave set 10', vertices = verticesSlave)
bridge.Tie(adjust=ON, master= setMaster, name='Constraint-10', positionToleranceMethod=COMPUTED,
    slave= setSlave, thickness=ON, tieRotations=ON)

# ------------------------------------------------------------------------------------------------
# ---------------------- EQUATION AND SPRING STIFFNESS AT GIRDER ENDS ----------------------------
# ------------------------------------------------------------------------------------------------
# Rigel 1: center to girder
verticesMaster = inst_rigel1.vertices.findAt(((deck[0]), ), )
verticesSlave = inst_girder.vertices.findAt(((deck[0]), ), )
setMaster = myAssembly.Set(name = 'Master set 11', vertices = verticesMaster)
setSlave = myAssembly.Set(name = 'Slave set 11', vertices = verticesSlave)
#bridge.Tie(adjust=ON, master= setMaster, name='Constraint-11', positionToleranceMethod=COMPUTED,
    slave= setSlave, thickness=ON, tieRotations=ON)
bridge.Equation(name = 'eq1', terms = ((1.0, 'Master set 11', 3,), (-1.0, 'Slave set 11', 3,)))
bridge.Equation(name = 'eq2', terms = ((1.0, 'Master set 11', 2,), (-1.0, 'Slave set 11', 2,)))
bridge.Equation(name = 'eq3', terms = ((1.0, 'Master set 11', 4,), (-1.0, 'Slave set 11', 4,)))

#Horizontal translational spring at rigel1
region=((setMaster, setSlave), )
myAssembly.engineeringFeatures.TwoPointSpringDashpot(
    name='Springs/Dashpots-1', regionPairs=region, axis=FIXED_DOF, dof1 = 1, dof2 = 1,
    springBehavior=ON, springStiffness=1291200000.0, dashpotBehavior=OFF,
    dashpotCoefficient=0.0)

# Rigel 2: center to girder
verticesMaster = inst_rigel2.vertices.findAt(((deck[-1]), ), )
verticesSlave = inst_girder.vertices.findAt(((deck[-1]), ), )
setMaster = myAssembly.Set(name = 'Master set 12', vertices = verticesMaster)
setSlave = myAssembly.Set(name = 'Slave set 12', vertices = verticesSlave)
#bridge.Tie(adjust=ON, master= setMaster, name='Constraint-12', positionToleranceMethod=COMPUTED,
```

```
    slave= setSlave, thickness=ON, tieRotations=ON)
bridge.Equation(name = 'eq4', terms = ((1.0, 'Master set 12', 3,), (-1.0, 'Slave set 12', 3,)))
bridge.Equation(name = 'eq5', terms = ((1.0, 'Master set 12', 2,), (-1.0, 'Slave set 12', 2,)))
bridge.Equation(name = 'eq6', terms = ((1.0, 'Master set 12', 4,), (-1.0, 'Slave set 12', 4,)))

#Horizontal translational spring at rigel2
region=((setMaster, setSlave), )
myAssembly.engineeringFeatures.TwoPointSpringDashpot(
    name='Springs/Dashpots-2', regionPairs=region, axis=FIXED_DOF, dof1 = 1, dof2 = 1,
    springBehavior=ON, springStiffness=1291200000.0, dashpotBehavior=OFF,
    dashpotCoefficient=0.0)

# ------------------------------------------------------------------------------------------------
# ------------------------------ TIES ALONG THE GIRDER -------------------------------------------
# ------------------------------------------------------------------------------------------------
sluttindex = len(z1_main)-1-2*np.argmax(z1_main)
for i in range(1, sluttindex):
        verticesMaster_girder = inst_girder.vertices.findAt((girder.vertices[6*i-3].pointOn[0], ) ,)
        verticesSlave_conelgirder = inst_conelgirder.vertices.findAt(
            (girder.vertices[6*i-3].pointOn[0], ) ,)
        setMaster_girder = myAssembly.Set(name='Master set 7-'+str(i),
            vertices=(verticesMaster_girder))
        setSlave_conelgirder = myAssembly.Set(name='Slave set 7-'+str(i),
            vertices=(verticesSlave_conelgirder))
        bridge.Tie(adjust=ON, master= setMaster_girder, name='Constraint-'+str(i+99),
            positionToleranceMethod=SPECIFIED,
                constraintEnforcement = NODE_TO_SURFACE, positionTolerance = 100.0,
                slave= setSlave_conelgirder, thickness=ON, tieRotations=ON)

# ------------------------------------------------------------------------------------------------
# ------------------------------ TIES AT THE BOTTOM OF HANGERS -----------------------------------
# ------------------------------------------------------------------------------------------------
sluttindex = len(z1_main)-1-2*np.argmax(z1_main)
# Underneath cable 1
for i in range(1, sluttindex):
        verticesMaster_conelgirder = inst_conelgirder.vertices.findAt(
            (hangers.vertices[-2*i+103].pointOn[0], ) ,)
        verticesSlave_hanger = inst_hangers.vertices.findAt(
            (hangers.vertices[-2*i+103].pointOn[0], ) ,)
        setMaster_conelgirder = myAssembly.Set(name='Master set 8-'+str(i),
            vertices=(verticesMaster_conelgirder))
        setSlave_hanger = myAssembly.Set(name='Slave set 8-'+str(i), vertices=(verticesSlave_hanger))
        bridge.Tie(adjust=ON, master= setMaster_conelgirder, name='Constraint-'+str(i+199),
            positionToleranceMethod=SPECIFIED,
                constraintEnforcement = NODE_TO_SURFACE, positionTolerance = 100.0,
                slave= setSlave_hanger, thickness=ON, tieRotations=ON)

# Underneath cable 2
for i in range(1, sluttindex):
        verticesMaster_conelgirder = inst_conelgirder.vertices.findAt(
            (hangers.vertices[2*i+101].pointOn[0], ) ,)
        verticesSlave_hanger = inst_hangers.vertices.findAt(
            (hangers.vertices[2*i+101].pointOn[0], ) ,)
        setMaster_conelgirder = myAssembly.Set(name='Master set 9-'+str(i),
            vertices=(verticesMaster_conelgirder))
        setSlave_hanger = myAssembly.Set(name='Slave set 9-'+str(i), vertices=(verticesSlave_hanger))
```

```
        bridge.Tie(adjust=ON, master= setMaster_conelgirder, name='Constraint-'+str(i+299),
            positionToleranceMethod=SPECIFIED,
                constraintEnforcement = NODE_TO_SURFACE, positionTolerance = 100.0,
                slave= setSlave_hanger, thickness=ON, tieRotations=ON)

# ------------------------------------------------------------------------------------------
# ------------------------------ TIES ON TOP OF HANGERS  -----------------------------------
# ------------------------------------------------------------------------------------------
startindex = np.argmax(z1_main)
sluttindex = 63
# Along cable 1
for i in range(startindex, sluttindex):
        verticesMaster_cable1 = inst_cable1.vertices.findAt(
            (cable1.vertices[i+1].pointOn[0], ) ,)
        verticesSlave_hanger = inst_hangers.vertices.findAt(
            (hangers.vertices[-2*i+124].pointOn[0], ) ,)
        setMaster_cable1 = myAssembly.Set(name='Master set 10-'+str(i-11),
            vertices=(verticesMaster_cable1))
        setSlave_hanger = myAssembly.Set(name= 'Slave set 10-'+str(i-11),
            vertices=(verticesSlave_hanger))
        bridge.Tie(adjust=ON, master= setMaster_cable1, name='Constraint-'+str(i+388),
            positionToleranceMethod=SPECIFIED,
                constraintEnforcement = NODE_TO_SURFACE, positionTolerance = 100.0,
                slave= setSlave_hanger, thickness=ON, tieRotations=ON)

# Along cable 2
for i in range(startindex, sluttindex):
        verticesMaster_cable2 = inst_cable2.vertices.findAt((cable2.vertices[i+1].pointOn[0], ) ,)
        verticesSlave_hanger = inst_hangers.vertices.findAt(
            (hangers.vertices[2*i+78].pointOn[0], ) ,)
        setMaster_cable2 = myAssembly.Set(name='Master set 11-'+str(i-11),
            vertices=(verticesMaster_cable2))
        setSlave_hanger = myAssembly.Set(name= 'Slave set 11-'+str(i-11),
            vertices=(verticesSlave_hanger))
        bridge.Tie(adjust=ON, master= setMaster_cable2, name='Constraint-'+str(i+488),
            positionToleranceMethod=SPECIFIED, constraintEnforcement = NODE_TO_SURFACE,
            positionTolerance = 100.0, slave= setSlave_hanger, thickness=ON, tieRotations=ON)

# ------------------------------------------------------------------------------------------
# ---------------------------------------- STEP 1 ------------------------------------------
# ------------------------------------------------------------------------------------------
# Create step
bridge.StaticStep(name='step1', previous='Initial', nlgeom=ON, initialInc = 0.001,
    timePeriod = 1, minInc = 1e-7)
bridge.FieldOutputRequest(name='F-Output-2', createStepName='step1', variables=('SF', 'U'),
    region = MODEL, frequency = 1)

# TOWERS LOADING
edges1 = inst_tower1.edges
edges2 = inst_tower2.edges
set_edges_towers = myAssembly.Set(edges=edges1+edges2, name='SetEdges towers')
last1 = bridge.Gravity(name='Gravity load towers', createStepName='step1', comp3=-9.81,
    distributionType=UNIFORM, field='', region = set_edges_towers)

# RIGELS LOADING
edges1 = inst_rigel1.edges
```

```
edges2 = inst_rigel2.edges
set_edges_rigels = myAssembly.Set(edges=edges1+edges2, name='SetEdges rigels')
last1 = bridge.Gravity(name='Gravity load rigels', createStepName='step1', comp3=-9.81,
    distributionType=UNIFORM, field='', region = set_edges_rigels)

#model change, remove: cables, girder, hangers, conelgirder
edges1 = inst_cable1.edges
edges2 = inst_cable2.edges
edges3 = inst_girder.edges
edges4 = inst_hangers.edges
edges5 = inst_coneltowers.edges
edges6 = inst_conelgirder.edges

set_modelchange1 = myAssembly.Set(edges=edges1+edges2+edges3+edges4+edges5+edges6,
    name='Set-modelchange1')
mdb.models['Model-1'].ModelChange(name = 'modelchange step 1', createStepName = 'step1',
    region = set_modelchange1)

# ------------------------------------------------------------------------------------------
# ---------------------------------------- STEP 2 ------------------------------------------
# ------------------------------------------------------------------------------------------
# Create step
bridge.StaticStep(name='step2', previous='step1', nlgeom=ON, initialInc = 0.001, timePeriod = 1,
    minInc = 1e-7, maxNumInc = 500, matrixSolver = DIRECT)
bridge.FieldOutputRequest(name='F-Output-2', createStepName='step2', variables=('SF', 'U'),
    region = MODEL) #, frequency = 1)

#model change, add: cables
edges1 = inst_cable1.edges
edges2 = inst_cable2.edges
edges3 = inst_coneltowers.edges
set_modelchange2 = myAssembly.Set(edges=edges1+edges2+edges3, name='Set-modelchange2')
mdb.models['Model-1'].ModelChange(name = 'modelchange step 2', createStepName = 'step2',
    region = set_modelchange2, activeInStep = ON)

# CABLE LOADING
edges1 = inst_cable1.edges
edges2 = inst_cable2.edges
set_edges_cables = myAssembly.Set(edges=edges1+edges2, name='SetEdges cables')
last2 = bridge.Gravity(name='Gravity load cables', createStepName='step2', comp3=-9.81,
    distributionType=UNIFORM, field='', region = set_edges_cables)

# ------------------------------------------------------------------------------------------
# ---------------------------------------- STEP 3 ------------------------------------------
# ------------------------------------------------------------------------------------------
# Create step
bridge.StaticStep(name='step3', previous='step2', nlgeom=ON, initialInc = 0.001, timePeriod = 1,
    minInc = 1e-7, maxNumInc = 500, matrixSolver = DIRECT)
bridge.FieldOutputRequest(name='F-Output-2', createStepName='step3', variables=('SF', 'U'),
    region = MODEL) #, frequency = 1)

#model change, add: girder, con.el.girder and hangers
edges1 = inst_girder.edges
edges2 = inst_conelgirder.edges
edges3 = inst_hangers.edges
set_modelchange3 = myAssembly.Set(edges=edges1+edges2+edges3, name='Set-modelchange3')
```

```
mdb.models['Model-1'].ModelChange(name = 'modelchange step 3', createStepName = 'step3',
    region = set_modelchange3, activeInStep = ON)

# GIRDER LOADING
edges = inst_girder.edges
set_edges_girder = myAssembly.Set(edges=edges, name='SetEdges girder')
last3 = bridge.Gravity(name='Gravity load girder', createStepName='step3', comp3=-9.81,
    distributionType=UNIFORM, field='', region = set_edges_girder)

# HANGERS LOADING
edges = inst_hangers.edges
set_edges_hangers = myAssembly.Set(edges=edges, name='SetEdges hangers')
last4 = bridge.Gravity(name='Gravity load hangers', createStepName='step3', comp3=-9.81,
    distributionType=UNIFORM, field='', region = set_edges_hangers)

# Remove the BC at the midspan of the bridge
constraint_midspan1.setValuesInStep(stepName = 'step3', u2 = FREED)
constraint_midspan2.setValuesInStep(stepName = 'step3', u2 = FREED)

# ------------------------------------------------------------------------------------------------
# ---------------------------------------- STEP 4 ------------------------------------------------
# ------------------------------------------------------------------------------------------------
bridge.FrequencyStep(name = 'stepModal', previous = 'step3', eigensolver = LANCZOS,
    numEigen = 100)
bridge.FieldOutputRequest(name = 'F-Output-frequency', createStepName = 'stepModal',
    variables = ('SF', 'U', 'COORD'), region = MODEL)
#bridge.HistoryOutputRequest(name = 'H-Output-frequency', createStepName = 'stepModal',
    variables = ('SF', 'U', 'UR'), region = MODEL)

# ------------------------------------------------------------------------------------------------
# ------------------------------- NODE NUMBERING AND PATH ----------------------------------------
# ------------------------------------------------------------------------------------------------
a = mdb.models['Model-1'].rootAssembly

# GIRDER
a.makeIndependent(instances=(a.instances['Girder instance'], ))
partInstances =(a.instances['Girder instance'], )
a.setMeshNumberingControl(instances=partInstances, startNodeLabel=10000,
    startElemLabel=10000)
session.Path(name='Path-girder', type=NODE_LIST,
    expression=(('Girder instance', ('10001:10306', )), ))

# CABLE 1 (positive y-values)
a.makeIndependent(instances=(a.instances['Cable1 instance'], ))
partInstances =(a.instances['Cable1 instance'], )
a.setMeshNumberingControl(instances=partInstances, startNodeLabel=20000,
    startElemLabel=20000)
session.Path(name='Path-cable1', type=NODE_LIST,
    expression=(('Cable1 instance', ('20001:20076', )), ))


#Node set
#m = mdb.model['Model-1']
#rA = m.rootAssembly
#rA.Set(name = 'all_nodes',
    nodes = reduce(lambda a,b : a+b,[rA.instances[key].nodes[:]for key in rA.instances.keys()]))
```

```
#girdernodes = myAssembly.nodes

myAssembly.Set(name = 'Girder nodes', nodes = inst_girder.nodes[:])

# ------------------------------------------------------------------------------------------
# ------------------------------- ADD BEAM GIRDER INERTIA ----------------------------------
# ------------------------------------------------------------------------------------------

mdb.models['Model-1'].keywordBlock.synchVersions(storeNodesAndElements=False)
mdb.models['Model-1'].keywordBlock.replace(111, """
*Beam General Section, elset=\"Girder set\", poisson = 0.3, density=0., section=GENERAL
1.82, 5.76, 0., 151.2, 20.33
0.,1.,0.
7e+10, 2.69231e+10
*BEAM ADDED INERTIA
4914,   0.,             0.,     0.,     15309,  408740,     0.
1977,   0.,             0.,     0.,     2019,   158358,     0.
6000,   0.,             2.0,    0.,     3,      450000,     0.
33,     -14.75,         2.0,    0.,     0.,     0.,         0.
33,     14.75,          2.0,    0.,     0.,     0.,         0.""")

"""
From above
1: Aluminium girder
2: Aluminium diaphragmas
3: Asphalt
4: Hanger heads left
5: Hanger heads right
"""

mdb.models['Model-1'].keywordBlock.insert(1861, """
*OUTPUT, HISTORY
*NODE PRINT, nset = "Girder nodes"
COORD, U, UR""")


# ------------------------------------------------------------------------------------------
# ---------------------------------------- CREATE JOB --------------------------------------
# ------------------------------------------------------------------------------------------
jobName='LangenuenJob'
myJob=mdb.Job(name=jobName, model='Model-1', multiprocessingMode=DEFAULT, numCpus=4,
    numDomains=4, numGPUs=0)
myJob.submit(consistencyChecking=OFF)

# ------------------------------------------------------------------------------------------
# ---------------------------------------- END ---------------------------------------------
# ------------------------------------------------------------------------------------------
```

Mari Voll Dombu, Markus Gjelstad

Parametric modelling of a suspension bridge with an aluminium girder

NTNU
Norwegian University of
Science and Technology