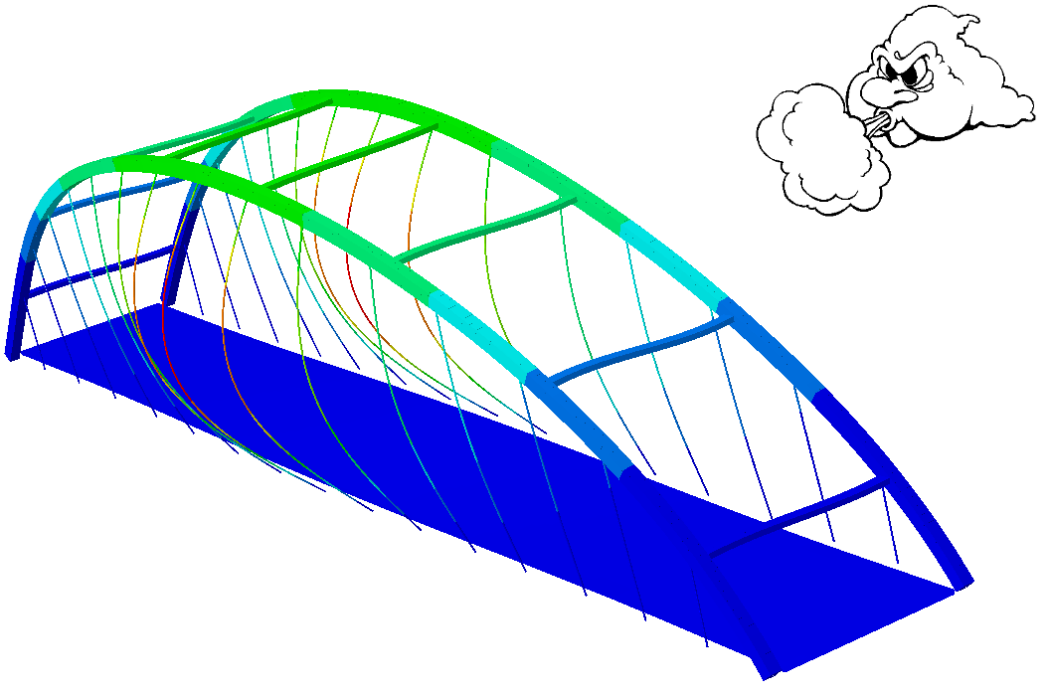
CHALMERS



Dynamic response of arch bridges exposed to wind load

Master of Science Thesis in the Master's Program Structural Engineering and Building Technology

JONATHAN JOHANSSON
DANIEL JOSEFSSON

Department of Applied Mechanics
Division of Dynamics
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2014
Master's Thesis 2014:35

MASTER'S THESIS 2014:35

Dynamic response of arch bridges exposed to wind load

Master of Science Thesis in the Master's Program Structural Engineering and Building Technology

JONATHAN JOHANSSON

DANIEL JOSEFSSON

Department of Applied Mechanics
Division of Dynamics
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2014

Dynamic response of arch bridges exposed to wind load

Master of Science Thesis in the Master's Program Structural Engineering and Building Technology

JONATHAN JOHANSSON
DANIEL JOSEFSSON

Master's Thesis 2014:35
ISSN 1652-8557
Department of Applied Mechanics
Division of Dynamics
Chalmers University of Technology
SE-412 96 Göteborg
Sweden
Telephone: + 46 (0)31-772 1000

Cover:
Arch bridge subjected to wind load.

Chalmers reproservice Göteborg, Sweden 2014

Dynamic response of arch bridges exposed to wind load

Master of Science Thesis in the Master's Program Structural Engineering and Building Technology

JONATHAN JOHANSSON
DANIEL JOSEFSSON
Department of Applied Mechanics
Division of Dynamics
Chalmers University of Technology

ABSTRACT

According to the Swedish design code, wind loads on bridges with constant cross section and span length less than 50m can be assessed in a simplified method presented in Eurocode. However, this simplified method is not applicable for arch bridges together with several other types of bridges. The dynamic response for these bridges needs to be assessed, which results in time consuming calculations.

To simplify the design calculations when investigating the dynamic response for a bridge it is of big importance to prove the first mode in wind direction is dominating. By proving this one can use a method known as the singe-mode method.

This Master's Thesis consists of a parametric study which displays the structural demand of arch bridges that are subjected to wind loads. Selected parameters are chosen and their influence on the decision of the dominating mode is investigated. The parametric study is divided into three main groups depending on the material of the arches.

The result is presented in written where characteristic examples for the associated parametric study are displayed. In addition to this a regression analysis is performed to estimate the relationship between the different parameters and the associated structural factor.

Keywords: Dynamic response, arch bridges, wind load, single-mode method, structural factor, dominating mode

Dynamisk respons av bågbroar med avseende på vindlast

Examensarbete inom Structural Engineering and Building Technology

JONATHAN JOHANSSON

DANIEL JOSEFSSON

Institutionen för Tillämpad Mekanik

Avdelningen för Dynamik

Chalmers tekniska högskola

SAMMANFATTNING

Enligt Trafikverkets dimensioneringsprinciper kan vindlaster för broar med konstant tvärsnitt och spännvidd under 50m utvärderas med en förenklad metod beskriven i Eurocode. För bl.a. bågbroar krävs däremot en mer detaljerad dynamisk analys där den dynamiska responsen för bron utreds. Detta resulterar i en mer komplicerad dimensioneringsprocess vilken ofta är tidskrävande.

För att förenkla dimensioneringsberäkningarna när den dynamiska responsen utreds är det en stor fördel om den första moden för bågen är dominerande. Genom att påvisa detta kan en enkel-mods metod användas vid beräkningen.

Detta mastersarbete består av en parameterstudie vilken visar det strukturella beteendet för bågbroar exponerade för vindlaster. Parametrar vilka anses ha stor inverkan på det dynamiska beteendet är utvalda. Vidare är dessa parametrars inverkan vid bestämmandet av den dominarande moden i vindriktningen analyserad. Parameterstudien är indelad i tre huvudgrupper beroende av bågarnas material.

Resultatet är presenterat i text där karakteristiska exempel, för den givna huvudgruppen, är illustrerade. Förutom detta har en regressionsanalys utförts för att påvisa sambandet mellan de utvalda parametrarna som varierats och den strukturella faktorn.

Nyckelord: Dynamisk respons, bågbroar, vindlast, enkel-mods metod, strukturell faktor, dominerande mod

Contents

Preface

This Master Thesis completes the Master's Program Structural Engineering and Building Technology at Chalmers University of Technology. It was written in the spring of 2014 and was carried out in cooperation between Chalmers University of Technology and WSP Bridge and Hydraulic Design in Gothenburg.

The completion of the Master's Thesis would never have been possible without the support from people in our surroundings.

Gratitude is given to Roland Olsson and his co-workers at WSP Bridge and Hydraulic design for giving us the opportunity to fulfill this Master's Thesis.

We are grateful for the input we have gotten from our opponent group Ismail Koc & Endrit Ndoi.

We would like to thank our Examiner Thomas Abrahamsson for the help he has given us throughout the Thesis project.

The biggest appreciation is given to our Supervisors Peter Nilsson and Kristoffer Ekholm for all the help they given us with both practical and theoretical issues. With great dedication and commitment they have inspired us to perform at the top of our ability.

Jonathan Johansson & Daniel Josefsson
May 2014

Notations

Abbreviations

FE	Finite element
IC	Initial conditions
PSD	Power spectral density
RMS	Root mean square
SDOF	Single degree of freedom
MDOF	Multi degree of freedom

Roman upper case letters

A_1, A_2	Amplitude constants	[-]
B	Background response factor	[-]
C_1, C_2	Integration constants	[-]
C_d	Local drag coefficient	[-]
$\mathbf{C}$	Damping matrix	[Ns/m]
$\boldsymbol{C}$	Modal damping matrix	[Ns/m]
E	Young´s modulus	[Pa]
F	Non-dimensional PSD	[-]
$\mathbf{F}$	Load matrix	[N]
H	Frequency Response factor	[-]
I	Impulse load	[Ns]
I_u	Turbulence intensity	[-]
$\mathbf{K}$	Stiffness matrix	[N/m]
$\boldsymbol{K}$	Modal stiffness matrix	[N/m]
M	Modal mass	[kg]
$\mathbf{M}$	Mass matrix	[kg]
$\boldsymbol{M}$	Modal mass matrix	[kg]
N	Arbitrary degrees of freedom	[-]
R	Resonant response factor	[-]
S	Power spectral density	[-]
T	Period time	[s]
U	Along-wind velocity	[m/s]
V	Mean wind velocity	[m/s]
X	Stochastic process	[-]

Roman lower case letters

b	Local width	[m]
c	Damping coefficient	[Ns/m]
c_r	Roughness factor	[-]
c_o	Orography factor	[-]
f_1	First eigenfrequency	[Hz]
$\mathbf{f}$	Load vector	[N]
$\boldsymbol{f}$	Modal load vector	[N]
k	Spring stiffness	[N/m]
k_b	Background factor	[-]
k_r	Response factor	[-]
k_p	Peak factor	[-]
k_t	Terrain factor	[-]
m	Mass	[kg]
p	Applied load	[N]
t	Time	[s]
u	Displacement	[m]
$\boldsymbol{u}$	Displacement vector	[m]
$\dot{u}$	Velocity	[m/s]
$\dot{\boldsymbol{u}}$	Velocity vector	[m/s]
$\ddot{u}$	Acceleration	[m/s^2]
$\ddot{\boldsymbol{u}}$	Acceleration vector	[m/s^2]
$\bar{u}$	Average displacement	[m]
u_x	Zero mean turbulence	[m/s]
v	Average frequency	[Hz]
v_b	Basic mean wind velocity	[m/s]
v_m	Mean wind velocity	[m/s]
w	Along wind load	[kN/m]
w_s	Load due to mean wind	[kN/m]
w_t	Load due to turbulence	[kN/m]
z_0	Roughness length	[m]

Greek upper case letters

Φ	Eigenvector	[-]
Ω	Frequency of the applied load	[rad/s]

Greek lower case letters

α	Phase angle	[rad]
ζ	Damping ratio	[-]
ζ_a	Aero dynamic damping	[-]
ζ_s	Structural damping	[-]
μ_x	Mean value of stochastic process	[-]
ν	Poisson's ratio	[-]
ρ	Density	[kg/m^3]
σ_x	Standard deviation	[-]
φ_X	Gust factor	[-]
ϕ	Mode shape	[-]
$\mathbf{\Phi}$	Modal matrix	[-]
ω	Circular frequency of load	[rad/s]
ω_d	Damped natural frequency	[rad/s]
ω_n	Natural frequency	[rad/s]
ω^*	Damped frequency	[rad/s]

1 Introduction

1.1 Background

According to the current regulations in Eurocode 1 (SS-EN 1991-1-4, 2005), bridges exposed to wind loads can be assessed by a simplified method where dynamic response calculations are not necessary. The structural factor, $c_s c_d$, is in the simplified method set equal to 1.0. However the Swedish guidelines in TRVK Bro 11 (Trafikverket, 2011), states that the dynamic response has to be evaluated for arch bridges, suspension bridges, cable-stay bridges, bridges with roofs, bridges with high slender columns and bridges with spans over 50 meters.

The dynamic response calculations are time consuming but may be simplified by a single-mode method. By using this method the number of parameters would be reduced and thereby shorten the calculation process. The single-mode method is acceptable to use under the condition that the structural response of the bridge is dominated by the first in-wind directional mode. A parametric study where the parameters that has the highest impact in the decision whether the in-wind directional mode is dominating or not would therefore be of interest. If these parameters are set into relation with the corresponding structural factor it might also be possible to assess the dynamic response without detailed calculations.

1.2 Aim and objective

The aim of the thesis is to produce a parametric study that will investigate the structural demands of arch bridges, with regards to wind load response, in order to use a simplified single-mode method. The structural factor, according to Eurocode 1 (SS-EN 1991-1-4, 2005), will be evaluated for those bridges that fulfill the requirements for the single mode method.

1.3 Method

This master thesis will be carried out in two parts; a literature study and a parametric study.

Initially a literature study is performed which is used to evaluate what parameters have the highest impact in the decision whether the in-wind directional mode is dominating or not. The literature study includes theory of structural dynamics, evaluation of different wind spectrum as well as the process to evaluate the structural factor.

The purpose of the parametric study is to evaluate different arch bridge designs and their structural response. The condition to use the single-mode method is that the structural response is dominated by the first in-wind directional mode. This condition is evaluated by analyzing the eigenfrequencies and the RMS (Root Mean Square) values of the displacements. In order to obtain the eigenfrequencies and the RMS values a FE-software will be used. The FE-software chosen for this study is BRIGADE/PLUS which is a FE-software customized for bridge engineering. In case

the results are satisfying the conditions for using the single-mode method, the structural factor for the corresponding bridge type is evaluated.

By setting the parameters that are used in the parametric study in relation to the corresponding structural factor it might be possible to predict the dynamic response for other, not yet assessed bridges.

1.4 Limitations

The behavior of the structure is assumed to be linear elastic. Only dynamic response due to wind load is considered. The parametric study will only evaluate through arch bridges with zero hinges.

1.4.1 Wind assumptions

In the theory part concerning wind load the following assumptions apply:

- The terrain is assumed to be horizontal and its roughness is constant
- At a sufficient height the wind flow is assumed to be horizontally homogeneous
- Any thermal contributions to the turbulence is neglected
- The wind direction don't change with the height above the terrain
- The main flow direction of the wind is perpendicular to the bridge span axis.

2 Theory of structural dynamics

The reader is expected to have elementary knowledge in structural dynamics, some basic equations and concepts are presented in this chapter. The equations in this section are derived according to (Craig Jr & Kurdila, 2006).

2.1 Undamped single degree of freedom system

The simple single degree of freedom (SDOF) model presented in Figure 2.1 below is considered. The mass is excited by an arbitrary force $p(t)$, the base is assumed to be fixed and the system starts from rest.

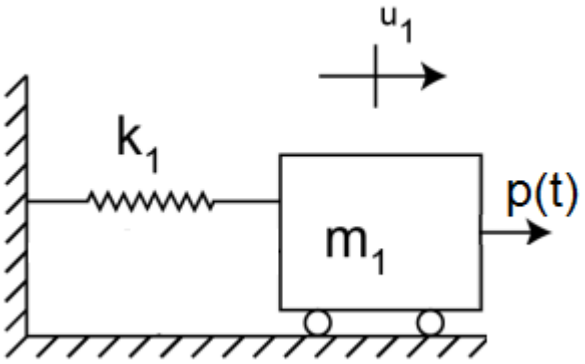


Figure 2.1 - SDOF Spring-mass system

The equation of motion for this system is

$$m\ddot{u}(t) + ku(t) = p(t) \tag{2.1}$$

Where

m	Mass	[kg]
k	Spring stiffness	[N/m]
p	Applied load	[N]
u	Displacement	[m]

Introducing the *natural frequency* ω_n

$$\omega_n = \sqrt{\frac{k}{m}} \tag{2.2}$$

Then equation (2.1) can be rewritten as

$$\ddot{u}(t) + \omega_n^2 u(t) = \frac{p(t)}{m} \tag{2.3}$$

This is a linear ordinary differential equation with constant coefficients. This equation has a solution that consists of two parts, a complementary solution u_c and a particular solution. u_p The total solution is then

$$u(t) = u_c(t) + u_p(t) \tag{2.4}$$

The complementary solution is obtained by consider the free vibration case of the system i.e. when $p = 0$. Then the complementary solution has the general solution as

$$u_c(t) = A_1 \cos \omega_n t + A_2 \sin \omega_n t \tag{2.5}$$

The particular solution depends on the applied load. Assuming that the applied load is harmonic, then the particular solution will be

$$u_p(t) = \frac{p_0 \cos(\Omega t)}{m(\omega_n^2 - \Omega^2)} \tag{2.6}$$

The total solution is then solved by inserting (2.6) and (2.5) into (2.4) and then apply the initial conditions from which the constants A_1 and A_2 can be solved.

2.1.1 Short-duration impulse response

If the load duration is much less than the system period time the load can be expressed as an impulse load. Assuming that the system is subjected to a load of duration t_d and that $t_d \ll T_n$ then the impulse can be defined as

$$I = \int_0^{t_d} p(t)\, dt \tag{2.7}$$

Where

I	Impulse load	[Ns]
p	Applied load	[N]
t_d	Impulse duration	[s]

The applied load $p(t)$ is an arbitrary function in time. Figure 2.2 below shows examples of some simple functions that can be used to model an impulse.

Figure 2.2 - Impulse load – Square, sinusoidal and triangular load

The equation of motion then becomes

$$m\ddot{u} + k\dot{u} = \begin{cases} p(t), & 0 < t < t_d \\ 0, & t_d < t \end{cases} \tag{2.8}$$

Equation (2.8) is then integrated with respect to time and with the initial conditions incorporated.

$$m\dot{u}(t_d) + k\bar{u}t_d = I \tag{2.9}$$

Here $\bar{u}$ is the average displacement in the time interval. If it can be assumed that the impulse is short, then $t_d \to 0$. Then equation (2.9) will become

$$m\dot{u}(0^+) = I \tag{2.10}$$

A short impulse will then give the mass an initial velocity of

$$\dot{u}(0^+) = \frac{I}{m} \tag{2.11}$$

The initial displacement will then become

$$u(0^+) = 0 \tag{2.12}$$

By using equation (2.11) and (2.12) as IC, the impulse response can be solved using equation (2.4) with the particular solution $u_p = 0$.

2.2 Damped single degree of freedom system

The motions in real systems cannot continue indefinitely due to damping that will dissipate energy from the system. Therefore a damped single degree of freedom (SDOF) system illustrated in Figure 2.3 is considered.

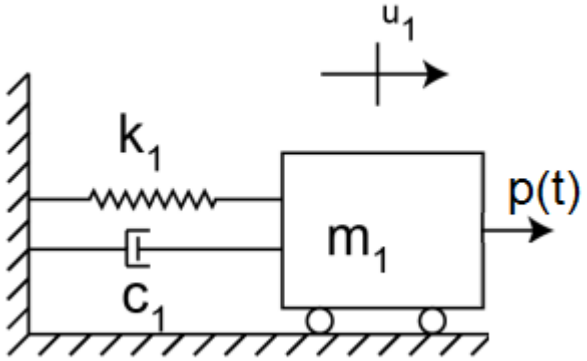


Figure 2.3 - SDOF Spring-mass-damper system

The equation of motion for this system is

$$m\ddot{u}(t) + c\dot{u}(t) + ku(t) = p(t) \tag{2.13}$$

Where

m	Mass	[kg]
c	Damping coefficient	[Ns/m]
k	Spring stiffness	[N/m]
p	Applied load	[N]
u	Displacement	[m]
$\dot{u}$	Velocity	[m/s]
$\ddot{u}$	Acceleration	[m/s^2]

The damping ratio is defined as

$$\zeta = \frac{c}{2\sqrt{km}} \tag{2.14}$$

Then the equation of motion then can be written as

$$\ddot{u}(t) + 2\zeta\omega_n\dot{u}(t) + \omega_n^2 u(t) = \frac{p(t)}{m} \tag{2.15}$$

As in the undamped case the solution for the differential equation consists of two parts the complementary solution and the particular solution, se equation (2.4). Equation (2.15) can be solved by making the ansatz

$$u(t) = Ce^{St} \tag{2.16}$$

This then gives

$$s^2 + 2\zeta\omega_n s + \omega_n^2 = 0 \tag{2.17}$$

This *characteristic equation* will have the roots given by

$$\left.\begin{matrix} s_1 \\ s_2 \end{matrix}\right\} = -\zeta\omega_n \pm \omega_n\sqrt{\zeta^2 - 1} \tag{2.18}$$

Three cases can be identified *underdamped* $(0 < \zeta < 1)$, *critically damped* $(\zeta = 1)$, and *overdamped* $(\zeta > 1)$. These three cases are illustrated in Figure 2.4.

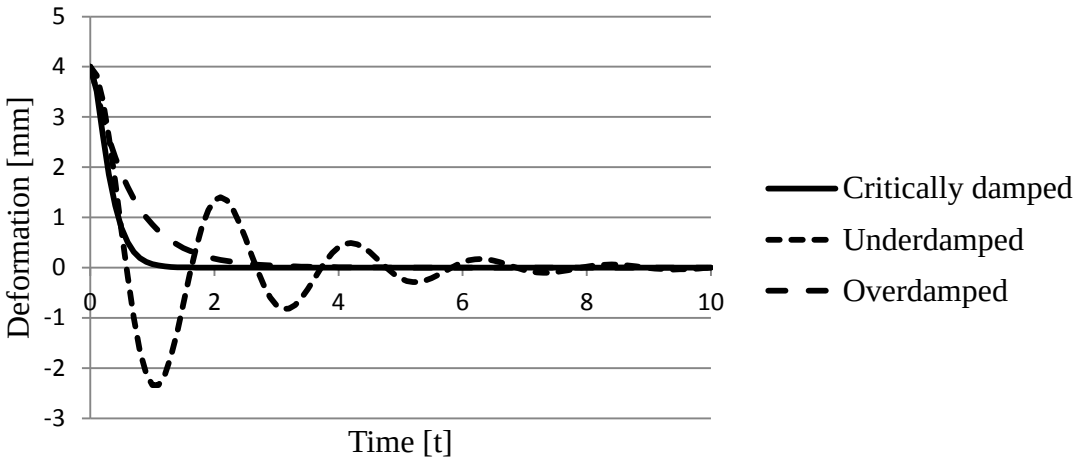


Figure 2.4 - Underdamped, Overdamped & Critically damped

2.2.1 Underdamped ($\zeta < 1$)

By introducing the *damped natural frequency* as

$$\omega_d = \omega_n\sqrt{1-\zeta^2} \tag{2.19}$$

Equation (2.18) can be rewritten in the form

$$\left.\begin{matrix} s_1 \\ s_2 \end{matrix}\right\} = -\zeta\omega_n \pm i\omega_d \tag{2.20}$$

Then the complementary solution can be written as

$$u_c(t) = e^{-\zeta\omega_n t}(A_1\cos\omega_d t + A_2\sin\omega_d t) \tag{2.21}$$

2.2.2 Critically damped ($\zeta = 1$)

For this case equation (2.18) will only have one solution, which is

$$s = -\omega_n \tag{2.22}$$

The complementary solution will be

$$u_c(t) = (C_1 + C_2 t)e^{-\omega_n t} \tag{2.23}$$

2.2.3 Overdamped ($\zeta > 1$)

Now equation (2.18) gives two distinct negative real roots. Introducing

$$\omega^* = \omega_n\sqrt{\zeta^2-1} \tag{2.24}$$

The complementary solution then can be written as

$$u(t) = e^{-\zeta\omega_n t}(C_1\cosh\omega^* t + C_2\sinh\omega^* t) \tag{2.25}$$

2.2.4 Particular solution

If the applied load is assumed to be harmonic the particular solution will be

$$u_p(t) = U\cos(\Omega t - \alpha) \tag{2.26}$$

Here U is the *steady-state amplitude* and α is the *phase angle* defined as

$$U = \frac{p_0}{k\sqrt{(1-r^2)^2 + (2\zeta r)^2}} \tag{2.27}$$

$$\tan\alpha = \frac{2\zeta r}{1-r^2} \tag{2.28}$$

Where

$$r = \frac{\Omega}{\omega_n} \tag{2.29}$$

The total solution can then be obtained, similar to the undamped system, by applying the IC and solve the constants from the complementary solution.

2.3 Multi degree of freedom system

Many structures like for example bridges are so complex that they cannot be represented by a SDOF system. For convenience a 2-degree of freedom system illustrated in Figure 2.5 will be considered. The method described below is analogue with an arbitrary N degree of freedom system.

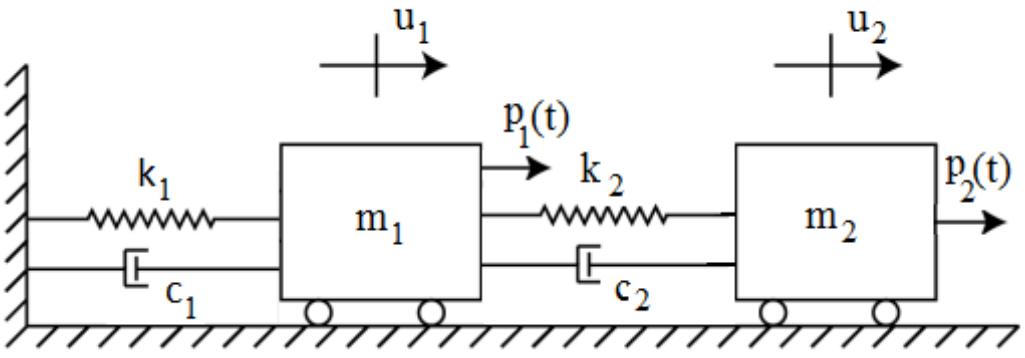


Figure 2.5 - MDOF Spring-Mass-Damper system

In order to obtain the equation of motion for the system a free-body diagram for each of the masses should be established. Then by applying Newton's second law of motion on each mass the following equation will be obtained

$$\begin{bmatrix} m_1 & 0 \\ 0 & m_2 \end{bmatrix}\begin{bmatrix} \ddot{u}_1 \\ \ddot{u}_2 \end{bmatrix} + \begin{bmatrix} c_1 + c_2 & -c_2 \\ -c_2 & c_2 \end{bmatrix}\begin{bmatrix} \dot{u}_1 \\ \dot{u}_2 \end{bmatrix} + \begin{bmatrix} k_1 + k_2 & -k_2 \\ -k_2 & k_2 \end{bmatrix}\begin{bmatrix} u_1 \\ u_2 \end{bmatrix} = \begin{bmatrix} p_1(t) \\ p_2(t) \end{bmatrix} \tag{2.30}$$

Introducing the notation

$\mathbf{M}$	Mass matrix
$\mathbf{C}$	Damping matrix
$\mathbf{K}$	Stiffness matrix
$\mathbf{f}$	Load vector
$\mathbf{u}$	Displacement vector
$\dot{\mathbf{u}}$	Velocity vector
$\ddot{\mathbf{u}}$	Acceleration vector

Then equation (2.30) can be written as

$$\mathbf{M}\ddot{\mathbf{u}} + \mathbf{C}\dot{\mathbf{u}} + \mathbf{K}\mathbf{u} = \mathbf{f} \tag{2.31}$$

Assuming that the motion is harmonic such that

$$\mathbf{u}(t) = \mathbf{U}\cos(\omega t - \alpha) \tag{2.32}$$

Here $\boldsymbol{U}$ is a vector with the constants that determines the amplitudes. By inserting (2.32) into (2.31) and disregarding the effect of damping

$$(\mathbf{K} - \omega^2\mathbf{M})\mathbf{U} = 0 \tag{2.33}$$

In order for equation (2.33) to have a nontrivial solution the values of ω^2 must satisfy the following equation

$$\left|\mathbf{K} - \omega_i^2\mathbf{M}\right| = 0 \tag{2.34}$$

By solving equation (2.34) the eigenfrequencies ω_i can be obtained. Once the eigenfrequencies has been determined the mode shapes can be calculated. The mode shapes are defined as

$$\boldsymbol{\phi}_r = \begin{bmatrix}\phi_1\\ \phi_2\end{bmatrix}_r = A_r\begin{bmatrix}1\\ \beta_r\end{bmatrix}, \qquad r = 1, 2 \tag{2.35}$$

Using $\boldsymbol{U} = \boldsymbol{\phi}_1 + \boldsymbol{\phi}_2$ and insert it into equation (2.32)

$$u_1(t) = A_1\cos(\omega_1 t - \alpha_1) + A_2\cos(\omega_2 t - \alpha_2) \tag{2.36}$$

$$u_2(t) = \beta_1 A_1\cos(\omega_1 t - \alpha_1) + \beta_2 A_2\cos(\omega_2 t - \alpha_2) \tag{2.37}$$

Here A_r and α_r can be calculated by applying IC.

2.4 Mode superposition and modal damping

The equation of motion for an N-DOF system has in general coupled equations. This requires solutions of N equations in N unknowns. The mode-superposition method is a method that transforms the set of coupled equations to a set of uncoupled equations. The method uses the normalized modes extracted from the undamped system.

Assuming that we have N modes and that $r = 1,2, \dots N$ the modal masses and the modal stiffnesses can be calculated using

$$M_r = \boldsymbol{\phi}_r^T \mathbf{M} \boldsymbol{\phi}_r \,, \qquad K_r = \boldsymbol{\phi}_r^T \mathbf{K} \boldsymbol{\phi}_r \tag{2.38}$$

Then due to orthogonality

$$\boldsymbol{\phi}_r^T \mathbf{M} \boldsymbol{\phi}_s = \boldsymbol{\phi}_r^T \mathbf{K} \boldsymbol{\phi}_s = 0 \tag{2.39}$$

This holds for all $r \neq s$. Collecting the mode shapes gives the modal matrix as

$$\boldsymbol{\Phi} = [\phi_1 \quad \phi_2 \quad \dots \quad \phi_N] \tag{2.40}$$

Then by using equation (2.38) and (2.39) the modal damping matrix and the modal stiffness matrix can be written as

$$\boldsymbol{M} = \boldsymbol{\phi}^T \mathbf{M} \boldsymbol{\phi} = diag(M_r), \qquad \boldsymbol{K} = \boldsymbol{\phi}^T \mathbf{K} \boldsymbol{\phi} = diag(K_r) \tag{2.41}$$

By introducing the principal coordinates

$$\boldsymbol{u}(t) = \boldsymbol{\Phi \eta}(t) = \sum_{r=1}^{N} \boldsymbol{\phi}_r \eta_r(t) \tag{2.42}$$

The equation of motion in principal coordinates can then be written as

$$\boldsymbol{M\ddot{\eta} + C\dot{\eta} + K\eta} = \boldsymbol{f}(t) \tag{2.43}$$

Where

$\boldsymbol{M}$	Modal mass matrix
$\boldsymbol{C}$	Modal damping matrix
$\boldsymbol{K}$	Modal stiffness matrix
$\boldsymbol{f}$	Modal load vector

Since the matrices $\boldsymbol{M}$ and $\boldsymbol{K}$ are diagonal equation (2.43) are coupled only through the damping matrix $\boldsymbol{C}$. One common damping technique used in structural dynamic problems is modal damping. The modal damping matrix is then assumed to satisfy orthogonality such that the damping matrix becomes

$$\boldsymbol{C} = \boldsymbol{\phi}^T \mathbf{C} \boldsymbol{\phi} = diag(C_r) = diag(2\zeta_r \omega_r M_r) \tag{2.44}$$

Equation (2.43) now consists of uncoupled equations of motion in principal coordinates.

2.5 Fourier analysis

When considering structures subjected to random loading it is convenient to transform the load response from the time domain to the frequency domain. This can be obtained by using Fourier analysis (Handa, 1982).

2.5.1 Fourier series

A periodic function can be separated into several harmonic components using a Fourier series expansion (Craig Jr & Kurdila, 2006).

Consider the periodic function

$$p(t) = \begin{cases} -p_0 & -\frac{\pi}{2} \leq t < 0 \\ p_0 & 0 \leq t < \frac{\pi}{2} \end{cases} \tag{2.45}$$

The function is illustrated in Figure 2.6.

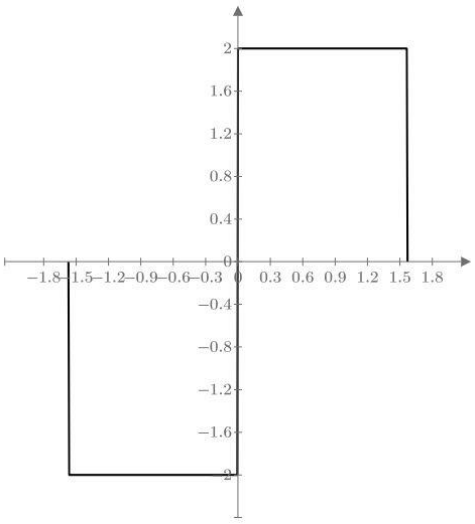


Figure 2.6 - Periodic function p(t)

By definition the *real Fourier series expansion* of $p(t)$ is

$$p(t) = a_0 + \sum_{n=1}^{\infty} a_n \cos n\Omega_1 t + \sum_{n=1}^{\infty} b_n \sin n\Omega_1 t \tag{2.46}$$

Where

$$\Omega_1 = \frac{2\pi}{T_1} \tag{2.47}$$

$$a_0 = \frac{1}{T_1} \int_{\tau}^{\tau+T_1} p(t)\, dt \tag{2.48}$$

$$a_n = \frac{2}{T_1} \int_{\tau}^{\tau+T_1} p(t) \cos n\Omega_1 t \; dt\,, \;\; n = 1,2 \ldots \tag{2.49}$$

$$b_n = \frac{2}{T_1}\int_{\tau}^{\tau+T_1} p(t)\sin n\Omega_1 t \; dt\,, \quad n = 1{,}2\ldots \tag{2.50}$$

Here τ is an arbitrary time and T_1 is the period of $p(t)$. In theory one may need an infinite number of terms n for $p(t)$. In this example it will be shown that one may get a sufficient approximation by using a relatively small number of terms.

By substituting equation (2.45) into equations (2.48), (2.49) and (2.50) it can be shown that the Fourier series representation according to equation (2.46) will become

$$p(t) = \frac{4p_0}{\pi}\sum_{n}^{N}\frac{1}{(2n-1)}\sin\big((2n-1)\Omega_1 t\big) \tag{2.51}$$

Equation (2.51) is plotted in Figure 2.7 using $N = 1,\ N = 3,\ N = 10$ and $N = 100$. It is clear that around 100 terms gives a good approximation of $p(t)$.

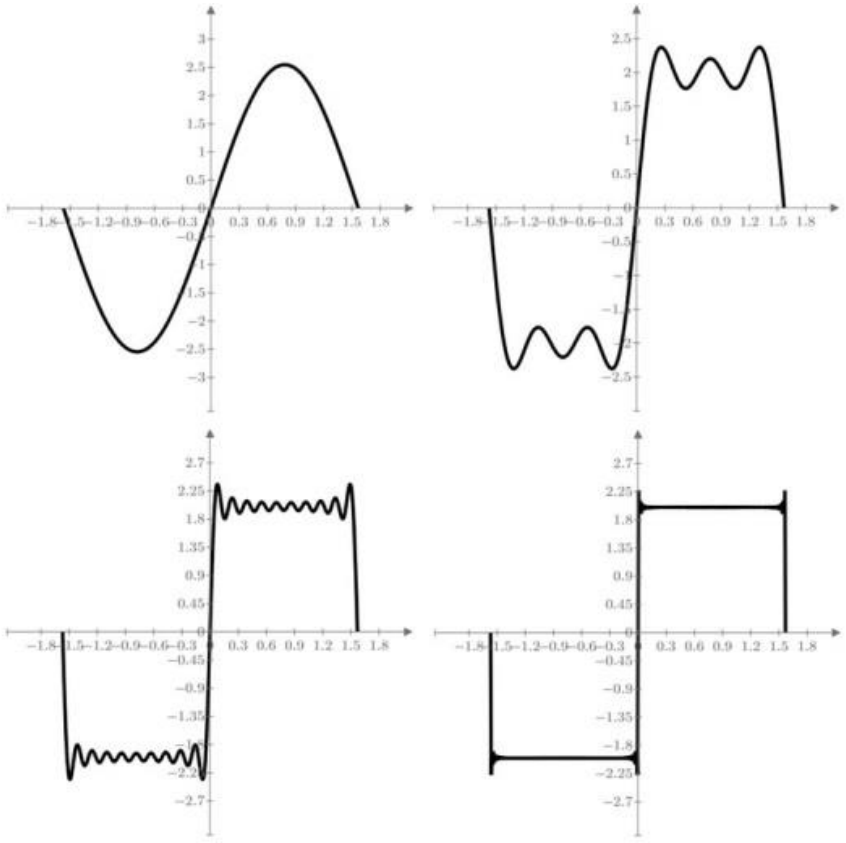


Figure 2.7 - Plots of equation (2.51) using N=1, 3, 10 & 100

In cases where the system is viscous-damped the complex Fourier series expansion can be useful instead of equation (2.46).

$$p(t) = \sum_{n=-\infty}^{\infty}\bar{P}_n(\Omega)e^{i(n\Omega_1 t)} \tag{2.52}$$

Where

$$\bar{P}_n = \frac{1}{T_1}\int_{\tau}^{\tau+T_1} p(t)e^{-i(n\Omega_1 t)}\,dt, \quad n = 0, \pm 1, \ldots \tag{2.53}$$

2.5.2 Fourier Integral

As shown in chapter 2.5.1 a periodic function can be written in terms of harmonic function with the Fourier series. However if a function is not periodic, such as for example the wind speed registration over a time period, we can still write it in terms of harmonic functions. This is done with the Fourier integral (Craig Jr & Kurdila, 2006).

Recalling equation (2.52) and (2.53), by letting $T_1 \to \infty$ and introducing the following notation

$$\Omega_1 = \Delta\Omega, \quad \Omega_n = n\Omega_1 \tag{2.54}$$

$$P(\Omega_n) = T_1\bar{P}_n = \frac{2\pi}{\Delta\Omega}\bar{P}_n \tag{2.55}$$

Then equation (2.52) can be written as

$$p(t) = \frac{1}{2\pi}\sum_{n=-\infty}^{\infty} \bar{P}(\Omega_n)e^{i(\Omega_n t)}\Delta\Omega \tag{2.56}$$

And equation (2.53) together with equation (2.55) will become

$$\bar{P}(\Omega_n) = \int_{-T_1/2}^{T_1/2} p(t)e^{i(\Omega_n t)}dt \tag{2.57}$$

And since $T_1 \to \infty$ then Ω_n becomes a continuous variable Ω and also $\Delta\Omega$ becomes the differential $d\Omega$. Hence equation (2.56) and (2.57) becomes

$$p(t) = \frac{1}{2\pi}\int_{-\infty}^{\infty} \bar{P}(\Omega)e^{i\Omega t}\,d\Omega \tag{2.58}$$

$$\bar{P}(\Omega) = \int_{-\infty}^{\infty} p(t)e^{i\Omega t}\,dt \tag{2.59}$$

Equation (2.58) is the inverse *Fourier transform* of $\bar{P}(\Omega)$ and equation (2.59) is called the *Fourier transform* of $p(t)$.

2.6 Random response analysis

The wind load due to turbulence is in structural engineering treated as a stochastic and ergodic process. In order to predict the response of a structure excited by such loading, one can in an FE-software perform a *random response analysis*. In this type of analysis the structure is excited by a random load. This load is characterized in the frequency domain by a cross-spectral density matrix $\boldsymbol{S}(f)$. From a previous conducted eigenvalue analysis the eigenmodes are used to calculate the power spectral densities (PSD) of the response variables considered. The obtained deformations will then give the frequencies at which the system is most excited (Dassault Systèmes, 2012).

3 Theory of wind engineering

Due to irregular heating of the atmosphere there will be temperature differences between different geographic areas of the earth. Hence, air pressure differences will occur which induces movement of the air to reach equilibrium. This movement provides the air with kinetic energy and the mass and velocity of the wind has to be taken into account when designing a structure (Handa, 1982).

The fluctuating wind field is assumed to be stationary and homogenous within the considered time and space. It may be approximated as a combination between the long-term variation of the mean wind velocity, the short term single spatial distribution of the turbulence components and the short term single point time domain variation of the turbulence components. An illustration over a fluctuating wind field is shown in Figure 3.1 (Strømmen, 2010).

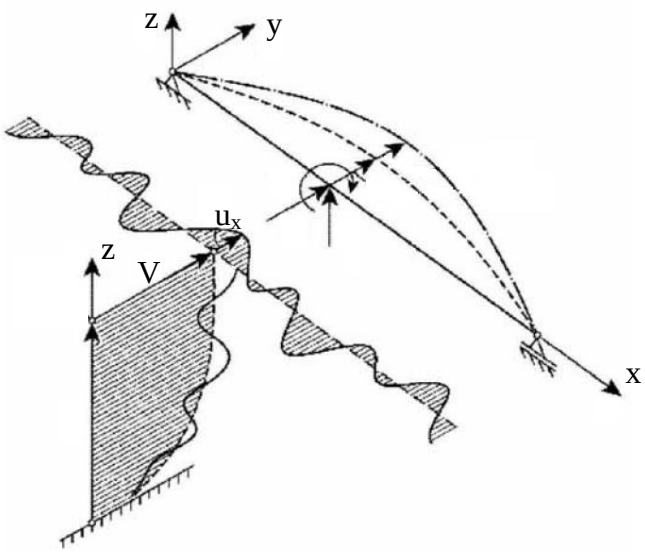


Figure 3.1 - Simple bridge structural system subjected to fluctuating wind field varying in time and space (Strømmen, 2010).

The along-wind load $w(z,t)$ per unit height can be defined according to Holmes (2001)

$$w(z,t) = \frac{1}{2} C_d(z) b(z) \rho_a U^2(z,t) \tag{3.1}$$

Where

$C_d(z)$	Local drag coefficient	[-]
$b(z)$	Local width	[m]
ρ_a	Air density	[kg/m^3]
$U(z,t)$	Wind velocity in along-wind direction	[m/s]

The wind velocity in along-wind direction $U(z,t)$ in equation (3.2) can according to Strømmen (2010) be defined as the mean wind velocity $V(z)$ added to the zero mean turbulence component $u_x(z,t)$.

$$U(z,t) = V(z) + u_x(z,t) \tag{3.2}$$

By inserting equation (3.2) into equation (3.1) and neglecting the term ${u_x}^2(z,t)$ since $V(z) \gg u_x(z,t)$, the following expression is obtained

$$w(z,t) = \frac{1}{2} C_d(z) b(z) \rho_a \left(V^2(z) + 2V(z) u_x(z,t) \right) \tag{3.3}$$

The wind load can be divided into the following contributions

$$w(z,t) = w_s(z) + w_t(z,t) \tag{3.4}$$

Where

$w_s(z) = \frac{1}{2} C_d(z) b(z) \rho_a V^2(z)$	Quasi-static load due to $V(z)$	[kN/m]
$w_t(z,t) = C_d(z) b(z) \rho_a V(z) u_x(z,t)$	Load due to wind turbulence	[kN/m]

3.1 Load due to mean wind velocity

The mean wind velocity at height z is depending on the terrain roughness, the orography and the basic wind velocity (Dyrbye, 1997). Eurocode 1 (SS-EN 1991-1-4, 2005) presents a way to calculate the mean wind velocity which is given by:

$$V(z) = c_r(z) c_o(z) v_b \tag{3.5}$$

Where

v_b	Basic mean velocity	[m/s]
$c_r(z)$	Roughness factor	[-]
$c_o(z)$	Orography factor	[-]

The basic mean velocity v_b varies between different geographic regions in Sweden and can be decided using the Swedish Annex to Eurocode (TRVFS, 2011:12).

According to the Swedish Annex to Eurocode (TRVFS, 2011:12) the orography factor $c_o(z)$ is taken as 1.0 since its influence is already accounted for in the basic mean velocity.

The roughness factor $c_r(z)$ depends on the height above the ground level and the roughness of the ground at the wind direction of the structure. It is given by Eurocode 1 (SS-EN 1991-1-4, 2005):

$$\begin{aligned} c_r(z) &= k_t \ln\left(\frac{z}{z_0}\right) \quad for \quad z_{min} \le z \le z_{max} \\ c_r(z) &= c_r(z_{min}) \quad for \quad z \le z_{min} \end{aligned} \tag{3.6}$$

Where

z_0	Roughness length given in Table 3.1	[m]
z_{min}	Minimum height given in Table 3.1	[m]
z_{max}	Maximum height taken as 200m	[m]
k_t	Terrain factor depending on roughness length	[-]

$$k_t = 0.19\left(\frac{z_0}{z_{0-IV}}\right)^{0.07} \tag{3.7}$$

Where

z_{0-IV}	Roughness length depending on terrain type 0-IV	[m]

Table 3.1 - Terrain categories and terrain parameters (SS-EN 1991-1-4, 2005).

Terrain category		**z_0 [m]**	**z_{min} [m]**
0	Sea or coastal area exposed to the open sea	*0.003*	*1*
I	Lakes or flat and horizontal area with negligible vegetation and without obstacles	*0.01*	*1*
II	Area with low vegetation such as grass and isolate obstacles (tree, buildings) with separations of at least 20 obstacles heights	*0.05*	*2*
III	Area with regular cover of vegetation or buildings or with isolated obstacles with separations of maximum 20 obstacle heights (such as villages, suburban terrain, permanent forrest)	*0.3*	*5*
IV	Area in which at least 15 % of the surface is covered with buildings and their average height exceeds 15 m	1.0	10

3.2 Load due to wind turbulence

As mentioned, the turbulence load varies in both time and space. Therefore the load cannot be predicted deterministically and statistical measures are needed (Dyrbye, 1997). The wind load and its turbulent component can be described as a stationary and ergodic stochastic process (Handa, 1982).

3.2.1 Power spectral density for along wind

A structure exposed to a turbulent wind load experience both high and low frequency loading. The part of the wind spectra that is of interest when designing a structure, with respect to wind loads, is the one above 0.1Hz (Handa, 1982). To describe the whole range of frequencies the power spectral density (PSD) for along wind $S_{u_x}(z,\omega)$ is used (Handa, 1982). Some of the most common theories to describe the interaction between the different frequency spans are Davenport's, Harris, Solari's and von

Karman's spectra. Davenport's and Harris spectra are independent on the height z while von Karman's and Solari's spectra are varying with different heights.

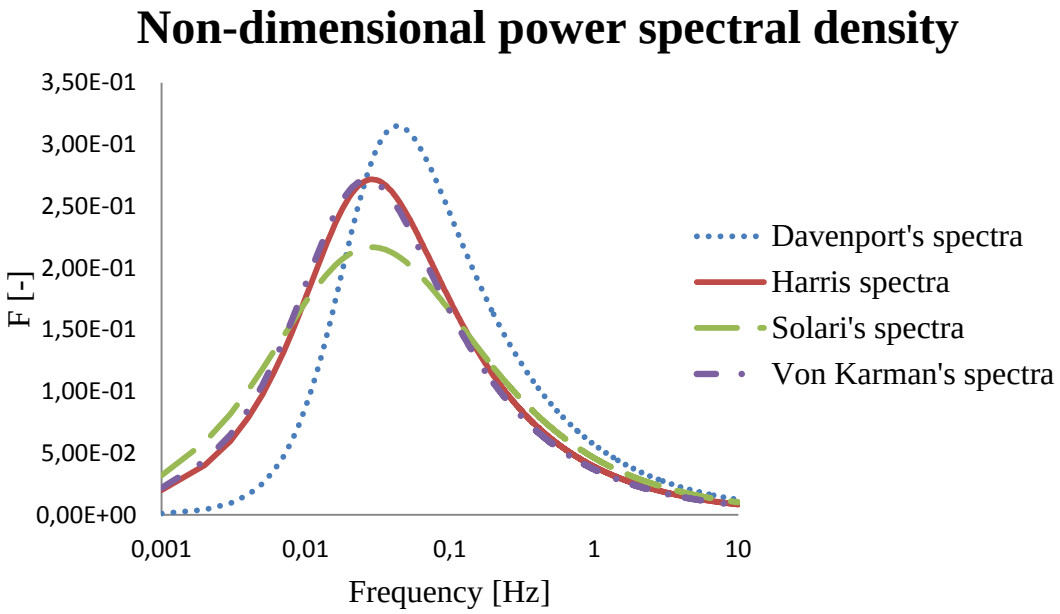


Figure 3.2 - Non-dimensional power spectral density given by von Karman, Solari, Davenport and Harris.

The non-dimensional PSD used in the Swedish Annex to Eurocode (TRVFS, 2011:12) origins from Von Karman's wind spectra and is given by

$$F = \frac{4\frac{150 f_1}{v_m}}{\left(1 + 70.8\left(\frac{150 f_1}{v_m}\right)^2\right)^{\frac{5}{6}}} \tag{3.8}$$

3.2.2 Stochastic processes and general definitions

A stochastic process is considered stationary under the conditions that the values are time-independent and that the correlations between values at different times only depend on time differences (Dyrbye, 1997). If every group of reading is considered statistically equivalent with every other group of readings and the fact that one of these groups would be representative for many groups of readings, than the process would be considered an stationary and ergodic stochastic process (Handa, 1982).

According to the gust factor method the gust factor, φ_X can be expressed as the maximum expected value of a stochastic process, $E[X_{max}]$ divided by μ_X, the expected value of the stochastic process. Another way to describe $E[X_{max}]$ is by adding the expected value of the stochastic process to the peak factor, k_p multiplied with σ_X, the standard deviation of the stochastic process (Mørk, et al., 1999)

$$\varphi_X = \frac{E[X_{max}]}{\mu_X} = 1 + \frac{k_p \sigma_X}{\mu_X} \tag{3.9}$$

The peak factor can be expressed as

$$k_p = \sqrt{2\ln\left(\frac{1}{2\pi}\frac{\sigma_{\dot{X}}}{\sigma_X}T\right)} + \frac{\gamma}{\sqrt{2\ln\left(\frac{1}{2\pi}\frac{\sigma_{\dot{X}}}{\sigma_X}T\right)}} \tag{3.10}$$

Where

$\sigma_{\dot{X}}$	Standard deviation of $\dot{X}$	[-]
T	Period time	[s]
γ	Eulers constant $= 0.5772$	[-]

The expected value μ_X of a stationary and stochastic process $X(t)$ is defined as (Strømmen, 2010)

$$\mu_X = E\{X(t)\} = \lim_{T\to\infty}\frac{1}{T}\int_{t_1}^{t_1+T} X(t)dt \tag{3.11}$$

By definition, the variance σ_x^2 of the stochastic process is defined as (Strømmen, 2010)

$$\sigma_x^2 = E\{(X(t)-\mu_X)^2\} \tag{3.12}$$

The cross covariance is used to determine the relation between two or more stochastic processes. If we consider $X(t)$ and $Y(t)$ as two stationary stochastic processes, the covariance Cov_{XY} would be expressed as (Dyrbye, 1997)

$$Cov_{XY}(\tau) = E\{(X(t)-\mu_X)(Y(t+\tau)-\mu_Y)\} \tag{3.13}$$

If $Y(t) = X(t)$ then equation (3.13) gives

$$Cov_X(\tau) = E\{(X(t)-\mu_X)(X(t+\tau)-\mu_x)\} \tag{3.14}$$

The cross-spectrum can for the processes $Y(t)$and X(t)be defined as (Mørk, et al., 1999)

$$S_{XY}(\omega) = 2\int_{-\infty}^{\infty} Cov_{XY}(\tau)e^{-i\omega\tau}d\tau \tag{3.15}$$

$$Cov_{XY}(\tau) = \frac{1}{4\pi}\int_{-\infty}^{\infty} S_{XY}(\omega)e^{i\omega\tau}d\omega \tag{3.16}$$

If $\tau = 0$ and using equation (3.14) and equation (3.15) the variance from the stochastic process $X(t)$ and $\dot{X}(t)$ are found (Mørk, et al., 1999)

$$\sigma_x^2 = \frac{1}{2\pi}\int_0^\infty S_{XX}(\omega)d\omega \tag{3.17}$$

$$\sigma_{\dot{x}}^2 = \frac{1}{2\pi}\int_0^\infty \omega^2 S_{XX}(\omega)d\omega \tag{3.18}$$

The cross-spectrum density $S_{XY}(\omega)$ is generally of a complex matter while the one-sided auto-spectrum density $S_{XX}(\omega)$ is always real for positive cyclic frequencies. The cross-spectrum is defined by using the cross-amplitude spectrum $|S_{XY}(\omega)|$ and the phase spectrum $\Phi_{XY}(\omega)$ and can be defined as (Mørk, et al., 1999)

$$S_{XY}(\omega) = |S_{XY}(\omega)|e^{\Phi_{XY}(\omega)} \tag{3.19}$$

To measure the statistical dependence between stochastic processes at a given frequency, the coherence spectrum $Coh_{XY}(\omega)$ is used (Dyrbye, 1997)

$$Coh_{XY}(\omega) = \frac{|S_{XY}(\omega)|^2}{S_{XX}(\omega)S_{YY}(\omega)} \tag{3.20}$$

3.2.3 Load contribution due to wind turbulence

According to equation (3.4) the load due to wind turbulence can be described as

$$w_t(z,t) = g(z)u_x(z,t) \tag{3.21}$$

The turbulence load can be written on spectral form by using the covariance function $Cov_{XX}(\tau)$ shown in equation (3.14)

$$Cov_{w_t w_t}(z_1, z_2, \tau) = E\{[w_t(z_1,t_1) - \mu_{w_t}(z_1,t_1)][w_t(z_2,t_2) - \mu_{w_t}(z_2,t_2)]\} \tag{3.22}$$

As $u(z,t)$ is defined as the zero mean turbulent component it can be shown that $\mu_{u_x} = 0 \rightarrow \mu_{w_t} = 0$. By applying this into equation (3.22) the following relation is obtained (Mørk, et al., 1999)

$$Cov_{w_t w_t}(\tau) = E\{g(z_1)u_x(z_1,t_1)g(z_2)u_x(z_2,t_2)\} \tag{3.23}$$

Since $g(z)$ is not time-dependent, equation (3.23) can be rewritten (Mørk, et al., 1999)

$$Cov_{w_t w_t}(\tau) = g(z_1)g(z_2)Cov_{u_x u_x}(\tau) \tag{3.24}$$

Applying the definition given in equation (3.15) into equation (3.24) gives

$$S_{\omega_t\omega_t}(z_1, z_2, \omega) = g(z_1)g(z_2)S_{u_x u_x}(z_1, z_2, \omega) \tag{3.25}$$

The auto-spectral density $S_{XX}(z, \omega)$ is determined using the cross-spectral density for the modal coordinate process (Mørk, et al., 1999)

$$S_{XX}(z, \omega) = \sum_{m=1}^{\infty}\sum_{n=1}^{\infty} \Phi_m(z)\Phi_n(z)S_{q_m q_n}(\omega) \tag{3.26}$$

The cross-spectral density $S_{q_m q_n}$is obtained using the frequency response function $H_i(\omega)$ (Strømmen, 2010)

$$S_{q_m q_n}(\omega) = H_m^*(\omega)H_n(\omega)S_{P_m P_n}(\omega) \tag{3.27}$$

The frequency response factor $H_i(\omega)$(*complex conjugate) is used to split the response calculation into a background and resonant part, shown in Figure 3.3.The frequency response factor is defined as (Mørk, et al., 1999)

$$H_m(\omega) = \frac{1}{(\omega_m^2 - \omega^2 + 2\zeta_m\omega_m\omega i)M_m} \tag{3.28}$$

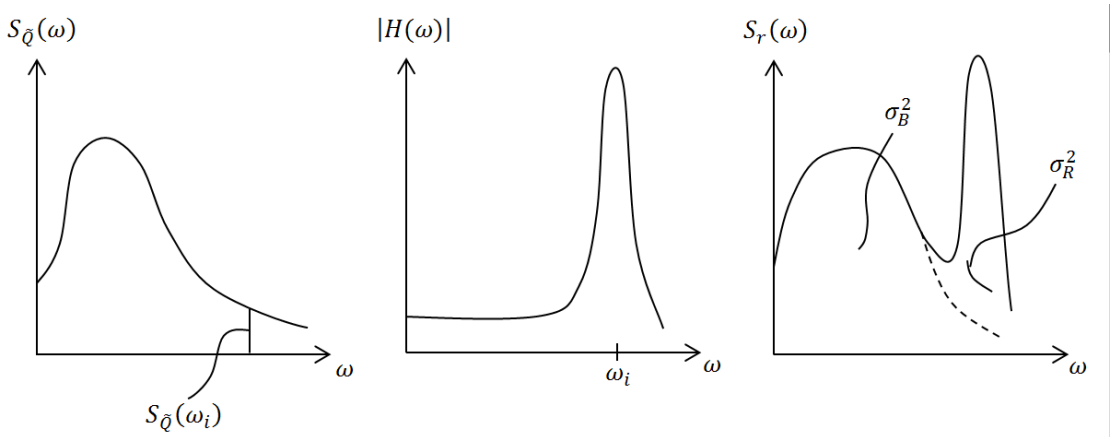


Figure 3.3 - Illustration demonstrating the split into background and resonant part using the frequency response factor.

The cross-spectral density for the modal load $S_{P_m P_n}(\omega)$ is determined using an orthogonally condition and given by (Mørk, et al., 1999)

$$S_{P_m P_n}(\omega) = \int_0^h \int_0^h \Phi_m(\mathrm{z}_1)\Phi_n(\mathrm{z}_2)\, S_{\omega_t\omega_t}(z_1, z_2, \omega)\, dz_1 dz_2 \tag{3.29}$$

If only the first mode shape $(m, n = 1)$is considered and in addition to this combining equation (3.17) and (3.26) the following expression will be obtained (Mørk, et al., 1999)

$$\sigma_x^2(z) = \frac{1}{2\pi}\Phi_1^2(z)\int_0^{\infty} |H_1(\omega)|^2 S_{P_1 P_1}(\omega)\, d\omega \tag{3.30}$$

The expression in equation (3.30) may be simplified by using equation (3.29) and (3.25) and by integrating with regard to the cyclic frequency ω (Mørk, et al., 1999)

$$\sigma_x^2(z) \approx (\mu_x(z) 2 I_u(z_{ref}))^2 (k_b + k_r) \tag{3.31}$$

The turbulence intensity $I_u$is given by

$$I_u(z_{ref}) = \frac{\sigma_u(z_{ref})}{V(z_{ref})} \tag{3.32}$$

The background factor k_b and the response factor k_r from equation (3.31) are given by

$$k_b = \frac{1}{2\pi} \frac{\int_0^\infty \int_0^h \int_0^h \Phi_1(z_1)\Phi_1(z_2) g(z_1) g(z_2) \frac{S_{u_x u_x}(z_1, z_2, \omega_2)}{\sigma_u^2} dz_1 dz_2 d\omega}{\left(\int_0^h \Phi_1(z) g(z) dz\right)^2} \tag{3.33}$$

$$k_r = \frac{1}{2} \frac{\pi^2}{\delta_s + \delta_a} \frac{n_1 S_{u_x}(z_{ref}, n_1)}{\sigma_u^2} K_s(n_1) \tag{3.34}$$

Where

$n_1 = \frac{\omega_1}{2\pi}$	Lowest eigenfrequency	[Hz]
$\delta_s = 2\pi\zeta_{1S}$	Structural damping coefficient for lowest eigenfrequency	[-]
$\delta_a = 2\pi\zeta_{1a}$	Aerodynamic damping coefficient for lowest eigenfrequency	[-]
$K_s(n_1)$	Size reduction factor given in equation (3.35)	

$$K_s(n_1) = \frac{\int_0^h \int_0^h \Phi_1(z_1)\Phi_1(z_2) g(z_1) g(z_2) \sqrt{Coh_{u_x u_x}(z_1, z_2, n_1)} dz_1 dz_2}{\left(\int_0^h \Phi_1(z) g(z) dz\right)^2} \tag{3.35}$$

For a plan perpendicular to the wind direction full-scale experiment have shown that the vertical coherence spectrum can be stated as (Mørk, et al., 1999)

$$Coh_{u_x u_x}(z_1, z_2, n_1) = e^{-2Cn_1 \frac{|z_1 - z_2|}{v(\bar{z})}} \tag{3.36}$$

Where

C	Decay constant	[-]
$v(\bar{z})$	$\frac{1}{2}(v(z_1) + v(z_2))$	[m/s]

Using the background factor, resonant factor and size reduction factor one can calculate the structural factor $c_s c_d$.

3.2.4 Wind turbulence according to the Swedish Annex to Eurocode

In the Swedish Annex to Eurocode (TRVFS, 2011:12) some basic equations are presented which can be combined with the theory given in chapter 3.2.1 - 3.2.3. The factor comprising the resonant response of the structure is given as

$$R^2 = \frac{2\pi F \emptyset_b \emptyset_h}{\delta_s + \delta_a} \tag{3.37}$$

Where F is Von Karman's wind spectra given in equation (3.8). The variables $\emptyset_b$ and $\emptyset_h$ are equal to $K_s(n_1)$ given in equation (3.35) where $\emptyset_b$ covers the horizontal correlation and $\emptyset_h$ the vertical correlation. The factor comprising the background response of the structure is given by

$$B^2 = e^{-0.05\frac{z}{z_{ref}} + \left(1-\frac{b}{z}\right)\left(0.04+0.01\frac{z}{z_{ref}}\right)} \tag{3.38}$$

The background response factor will reduce the structural factor where b is the width of the structure. However a cconservative approximation can be made by setting B^2 equal to 1.0. The average frequency is given by

$$v = f_1 \frac{R}{\sqrt{B^2 + R^2}} \tag{3.39}$$

The peak factor is given by

$$k_p = \sqrt{2\ln(vT)} + \frac{0.6}{\sqrt{2\ln(vT)}} \tag{3.40}$$

Where the time interval T can be set to 600 s which is the time interval belonging to the wind spectra. The turbulence intensity comprises the impact of the surrounding environment and height of the structure and is given by

$$I_v = \frac{k_l}{c_0 \ln\left(\frac{z}{z_0}\right)} \tag{3.41}$$

Where

k_l	Turbulence factor	[-]
c_0	Topographic factor	[-]

Finally the structural factor is calculated which can be compared with equation (3.9).

$$c_s c_d = \frac{1 + 2k_p I_v \sqrt{B^2 + R^2}}{1 + 7I_v} \tag{3.42}$$

The structural factor is the percentage increase of the static wind load.

4 Arch bridges

An arch can be defined according to Birnstiel (2014) as

> *"A structural member that spans horizontally between supports that develops inwardly directed horizontal reactions when the member is subjected to a vertical load"*

In theory a perfect arch is only subjected to compressive forces which act at the centroid of each arch element. Bridges are subjected to several types of loadings, such as dead load, temperature load, wind load, moving loads etc. This creates bending moments in addition to the compressive forces in the arch (Chen & Duan, 2000). In Figure 4.1 some arch terms are presented.

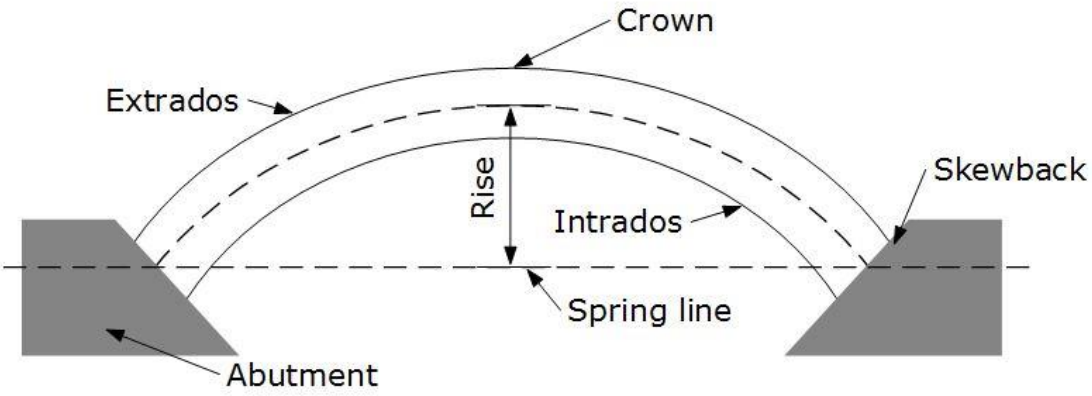


Figure 4.1 - Arch terminology

The choice of material nowadays for arch bridges is normally steel, concrete or timber. There are older bridges that have been built in masonry and stone but these materials are no longer used (Birnstiel, 2014).

4.1 Types of Arch bridges

One fundamental difference between different types of arch bridges is the location of the bridge deck. If the bridge deck lies above the arch it is called *deck arch*. In the case where the deck lies in the spring line of the arch it is called *through arch*. The third case is called *half-trough arch* and in this case the deck is elevated and placed between the spring line and the arch crown. In Figure 4.2 below sketches of these three types of arches are presented (Chen & Duan, 2000).

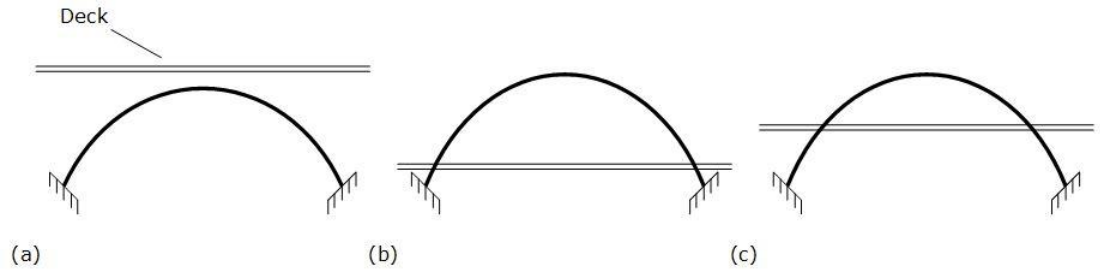


Figure 4.2 - (a) Deck arch. (b) Through arch. (c) Half-through arch

The arch itself can be built with zero, one, two or three hinges. In Figure 4.3a-b two example sketches of hinge placements are shown. An advantage in using hinges at the arch ends is the absence of moment at the skewback, and therefore it makes the foundation design easier compared to the fixed arch. An arch bridge can be constructed with a structural tie that balances the horizontal forces created by the arch and reliving the abutments from these forces (Birnstiel, 2014). This bridge type is called *tied-arch bridge* and is sketched in Figure 4.3c (Chen & Duan, 2000).

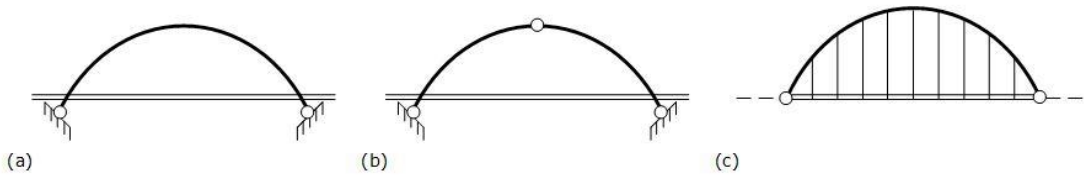


Figure 4.3 - (a) 2-hinges. (b) 3-hinges. (c) Tied-arch bridge

5 Parametric study

5.1 Workflow

The parametric study was carried out using the software's BRIGADE/PLUS, Matlab and Microsoft Excel. To be able to run multiple analyses in Brigade, Python scripts were used. The results from Brigade was analyzed by a Matlab program and thereafter presented in Microsoft Excel.

The parametric study comprised two processes. The first process was the modelling and generation of results, see Figure 5.1. The second process consisted of the analysis of results, see Figure 5.2. The script *BRIDGE_DATA.py* contained all necessary geometric and material data to generate the different Brigade models while *ARCH_BRIDGE.py* consisted of the code that generated the model in Brigade. The main script *PARAMETRIC_STUDY.py* combined the information from the two scripts and generated .inp-files, one for each parametric combination. The .inp-files were manually sent to the Brigade solver that generated results from the Frequency and Random Response analysis. The output from the analyses is gathered in .odb files. These can be viewed visually in the Brigade visualizer module.

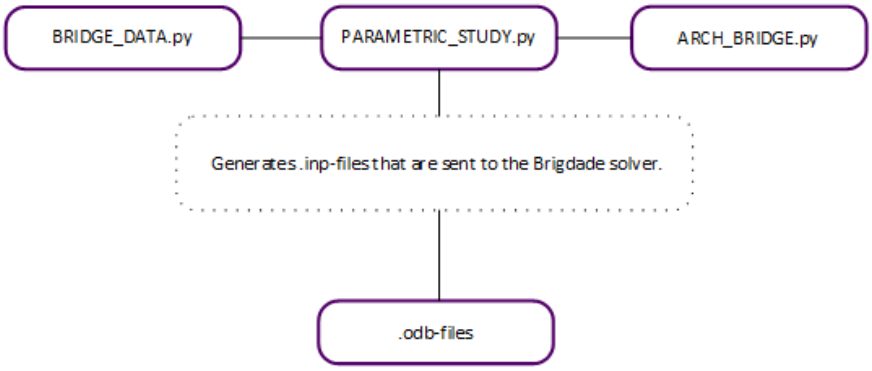


Figure 5.1 – Flow chart displaying the modelling and the generation of results.

To be able to run as many analyses as possible it was necessary to be able to manage the result in the odb-files in a non-visual manner. This was performed by *OUTPUT.py* that collected the information from each .odb-file and converted it into .txt-files. The .txt-files contained RMS-values in wind direction for selected nodes and also the eigenfrequencies with their corresponding mode shape. The Matlab program *ANALYSIS_OF_U_RMS.m* extracts necessary parameters from the .txt files and in the end this information was sent to the *FREQUENCY_RMS_RESULT.xlsx* that gave a better visual overview of the results.

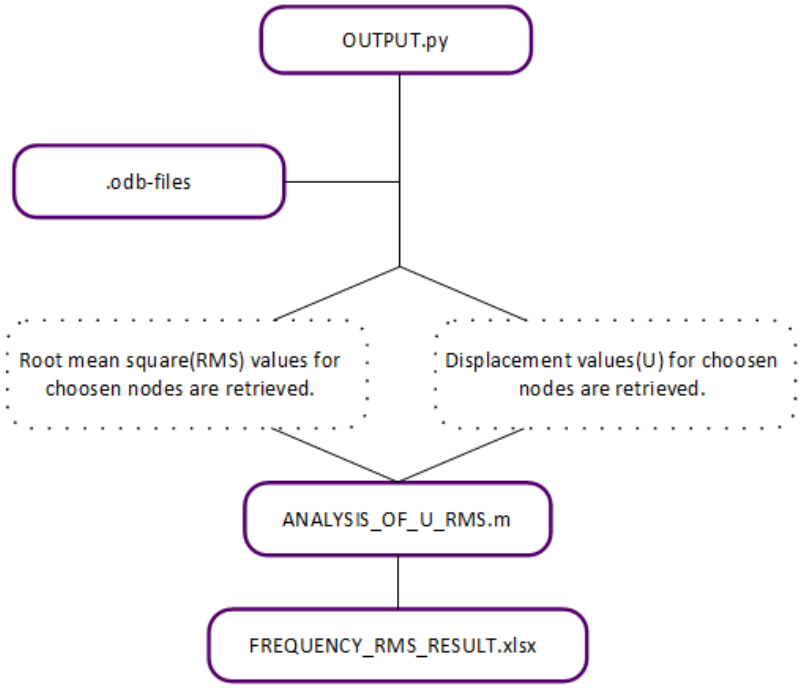


Figure 5.2 – Flow chart displaying the analysis of the results from the .odb-files.

The result presented in *FREQUENCY_RMS_RESULT.xlsx* showed whether the first in-wind directional mode was dominating or not. With this information it was possible to evaluate the structural factor.

5.2 Selection of input data

When selecting the input parameters for the arch bridges the Swedish Management system of Bridges and Tunnels, BaTMan, was used. Drawings and necessary structural parameters have been gathered from BaTMan and is presented in Appendix A.

Depending on the structure the parameters were chosen to resemble the parameters of a constructed arch bridge. To begin with, the different arch bridges are divided into material of the arch. Therefore three tables are presented, one for each type of material.

5.2.1 Parameters for concrete arch bridges

Initially a study of input data regarding concrete arch bridges was performed. The characteristics for these bridges are quite long spans with rectangular cross sections. The bridges do not normally have any cross bracing in addition to the transversal bracing. In Table 5.1 the chosen parameters for concrete arch bridges are shown.

Table 5.1 - Parameters that were chosen for concrete arch bridges.

Arch	
Number of hinges	0
Free height	4.7m
Arch height	10, 15, 20, 25, 30m
Shape of arch	Circular
Span length	60, 70, 80, 90, 100, 110, 120, 130, 140m
Distance between arches	8, 12, 16m
Material	Concrete
Shape of cross section	Rectangular
Width of cross section	0.5, 0,7, 0.9m
Height of cross section	0.5, 0.8, 1.1m
Bridge deck	
Material	Concrete
Height of bridge deck	0.25m
Hangers	
Material	Steel
Shape of cross section	Circular
Diameter of cross section	50mm
Quantity	15
Transverse bracing	
Material	Concrete
Shape of cross section	Rectangular
Height of cross section	0.7m
Width of cross section	0.3m
Quantity	4, 6, 8

5.2.2 Parameters for steel arch bridges

Steel arch bridges may be built as long span bridges but also as shorter spanned walking bridges. The arches are usually made as box profiles with transversal bracing and additional cross bracing. In Table 5.2 the chosen parameters for steel arch bridges are shown.

Table 5.2 - Parameters that were chosen for steel arch bridges.

Arch	
Number of hinges	0
Free height	4.7m
Arch height	6, 9, 12, 15, 18m
Shape of arch	Circular
Span length	30, 40, 50, 60, 70, 80, 90m
Distance between arches	4, 6, 8, 10m
Material	Steel
Shape of cross section	Box
Width of cross section	0.25, 0.50, 0.75m
Height of cross section	0.25, 0.50, 0.75m
Thickness of steel	15mm
Bridge deck	
Material	Concrete
Height of bridge deck	0.25m
Hangers	
Material	Steel
Shape of cross section	Circular
Diameter of cross section	50mm
Quantity	10
Transverse bracing	
Material	Steel
Shape of cross section	Box
Height of cross section	0.25x0.25x0.01m
Quantity	3, 6, 9, 12
Cross bracing	
Material	Steel
Shape of cross section	Circular
Diameter of cross section	40mm

5.2.3 Parameters for timber arch bridges

Timber arch bridges normally have shorter span than concrete and steel arch bridges. Generally the arches are made of glulam beams with a rectangular cross section and in addition they are braced with transverse and cross bracing. In Table 5.3 the chosen parameters for timber arch bridges are shown.

Table 5.3 – Parameters that were chosen for timber arch bridges.

Arch	
Number of hinges	0
Free height	4.7m
Arch height	6, 8, 10m
Shape of arch	Circular
Span length	20, 25, 30, 35, 40, 45, 50m
Distance between arches	3, 5, 7, 9m
Material	Glulam timber
Shape of cross section	Rectangular
Width of cross section	0.2, 0.4, 0.6m
Height of cross section	0.6, 0.8, 1.0m
Bridge deck	
Material	Glulam timber
Height of bridge deck	0.30m
Hangers	
Material	Steel
Shape of cross section	Circular
Diameter of cross section	25mm
Quantity	5
Transverse bracing	
Material	Glulam timber
Shape of cross section	Rectangular
Width of cross section	0.2m
Height of cross section	0.2m
Quantity	3, 5, 7
Cross bracing	
Material	Steel
Shape of cross section	Circular
Diameter of cross section	25mm

5.3 FE-model

The software used in this study is Brigade/Plus which is a modified version of the commercial software Abaqus. The difference between Abaqus and Brigade is that Brigade is stripped from certain functions and keywords that are normally not used in bridge design. Brigade has also support for live loads in order to simulate vehicles driving over the bridge.

The global coordinate system was set according to Figure 5.3, with the x-axis along the bridge length, y-axis in the bridge vertical direction and z-axis in the lateral direction (in wind load direction).

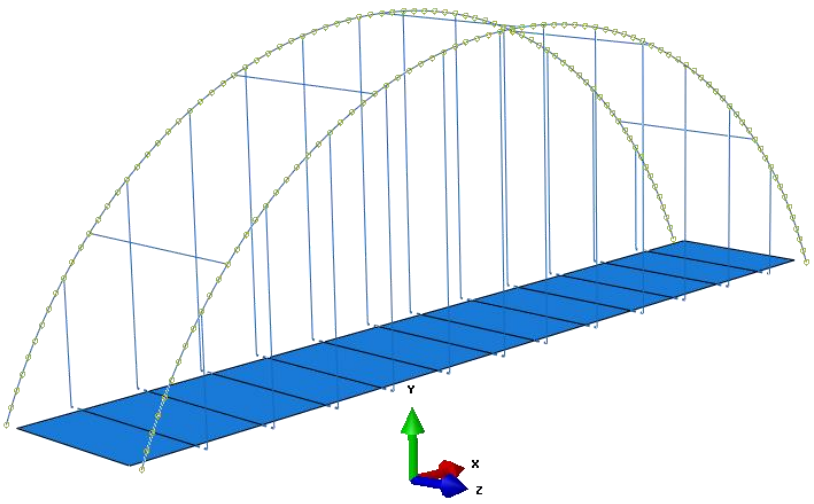


Figure 5.3 - Global coordinate system

5.3.1 Element types

The arch bridges were modelled using beam and shell elements. All parts of the bridge except the deck were modelled with beam type B31, which is a three-dimensional beam element that uses Timoshenko beam theory. The bridge deck was modelled with element type S4R, which is a quadrilateral shell element. These shell elements are so called general-purpose shell elements, which mean that they use both Kirchhoff and Mindlin shell theory in order to provide good solutions to both thin and thick shells (Dassault Systèmes, 2012).

5.3.2 Material data

Both concrete and steel were modelled as an elastic isotropic material. The material parameters defined were Young´s modulus E, Poisson´s ratio ν and the density ρ. The concrete strength class for all concrete bridges in the study was chosen to be C30/37. For concrete the material parameters were chosen according to Eurocode 2 (SS-EN

1992-1-1, 2005) and for steel according to Eurocode 3 (SS-EN 1993-1-1, 2005). Values are presented in Table 5.4.

Table 5.4 - Material parameters for concrete & steel

Material	E [GPa]	ν	ρ [kg/m^3]
Concrete [C30/37]	33	0.1	2500
Steel	200	0.3	7850

A big difference between Timber and the other two materials is that it is an anisotropic material with different material parameters in each direction. Therefore the glulam timber had to be defined with more detailed material parameters. In Brigade the material can be modelled by defining the engineering constants in each direction. The beam directions were set according to Figure 5.4.

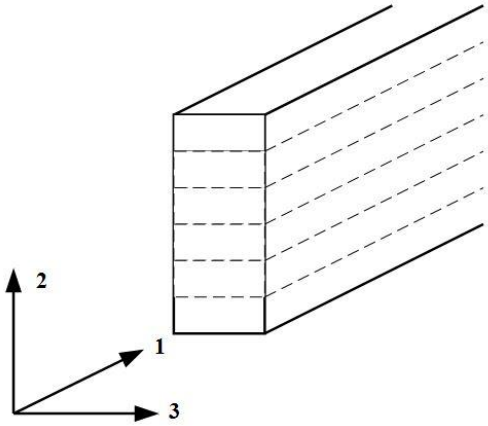


Figure 5.4 - Glulam beam cross-section direction

The strength class for the glulam timber was chosen to be GL28c and the material parameters were chosen according to the Swedish Standard Timber structures (SS-EN 14080, 2013) and are presented in Table 5.5.

Table 5.5 - Material parameters for glulam timber

Glulam timber GL28c							
E_1 [MPa]	E_2 [MPa]	E_3 [MPa]	ν_1, ν_2, ν_3	G_{12}	G_{13}	G_{23}	ρ [kg/m3]
12 500	300	300	0	650	650	65	420

5.3.3 Analysis type

Two different analyses were performed on each bridge model in the parametric study. First a frequency analysis was performed in order to obtain the eigenfrequencies and their corresponding mode shape. Then a random response analysis was performed with a rectangular power spectral density with magnitude 1 and with a line load applied on the arch in the wind direction. From this step the root mean square values of the arch displacement in the wind direction were obtained.

In the random response analysis modal damping was applied to all modes. The damping coefficient was calculated using equation (5.1) according to Craig Jr & Kurdila (2006) where δ is the logarithmic decrement taken according to Table F.2 in Eurocode 1 (SS-EN 1991-1-4, 2005).

$$\zeta = \frac{1}{\sqrt{1 + \left(\frac{2\pi}{\delta}\right)^2}} \tag{5.1}$$

In both types of analyses the frequency ranges of interest were set to 0.1-10 Hz. The lower limit was set according to Handa (1982). The higher limit was set due to fact that the wind spectrums, described in chapter 3.2.1, has very low values above 10 Hz. Therefore it is reasonable to assume that the wind cannot excite the structure at frequencies above this limit.

5.3.4 Assembly

The models of the arch bridges consist of five parts. These parts were all put together in an assembly step. In order to be able to control the coupling between the different parts, these were modeled so that there was a small distance between the connecting parts. The coupling between the parts were then set with an interaction connection in Brigade, see Figure 5.5.

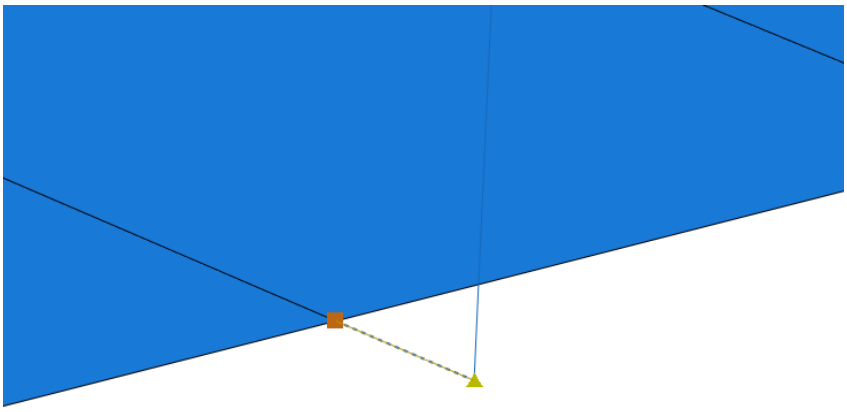


Figure 5.5 - Interaction connection between hanger and deck

Hinged connections were used for the hangers, connecting to the arch and the deck, and for the cross-bracings between the arches. The connection between the horizontal bracing members and the arches were modeled as a fixed connection.

5.3.5 Boundary conditions

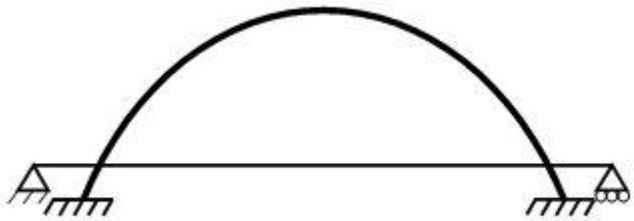


Figure 5.6 - Boundary conditions sketch

The boundary conditions for the bridge models were applied at the arch ends and at the bridge deck ends. A principal sketch of the boundary conditions is given in Figure 5.6.

The arch ends were prevented from rotation and translation in all directions. The bridge deck was in one end prevented from translation in all directions and in the other end the translation in global y- and z-axis was prevented.

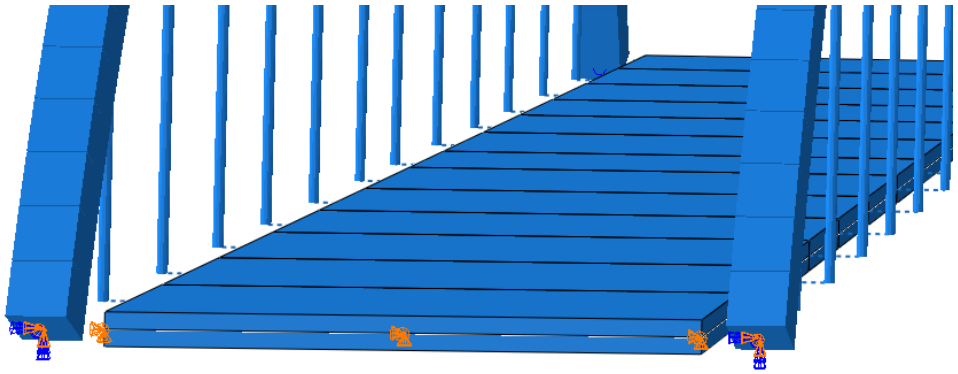


Figure 5.7 - Boundary conditions

5.3.6 Convergence study

When meshing an FE-model it is important that it is sufficiently detailed in order for the results to approach the analytical solution. However using an overly detailed mesh increases computation recourse and time (Zienkiewicz, et al., 2013).

A convergence study of the obtained eigenfrequencies was conducted in order to find an optimal mesh size. The results of the convergence study presented are from a concrete bridge with parameters set according to Table 5.6. Equivalent results were obtained from other types of bridges.

Table 5.6 - Convergence study bridge parameters

Span length [m]	30
Arch length [m]	10
Cross section dimension of arch [mm]	500 x 600
Distance between arches [m]	6
Thickness of deck [mm]	300
Number of transversal bracings	4
Cross section dimension of transversal bracing [mm]	500 x 600
Number of hangers	8
Diameter of hangers [mm]	50
Cross bracing	No

The arches were constructed in Brigade using straight lines, and in order to get a good shape on the arch these lines were set to have a length of 500 millimeters. This then limits the largest element size to the same value. The bridge was meshed using element size of 500mm, 250mm and 125mm. This corresponds to 84, 162 and 318 elements on a single arch. Then a frequency analysis was performed in order to obtain the eigenfrequencies. The first two eigenfrequencies that belonged to the arch is plotted in Figure 5.8.

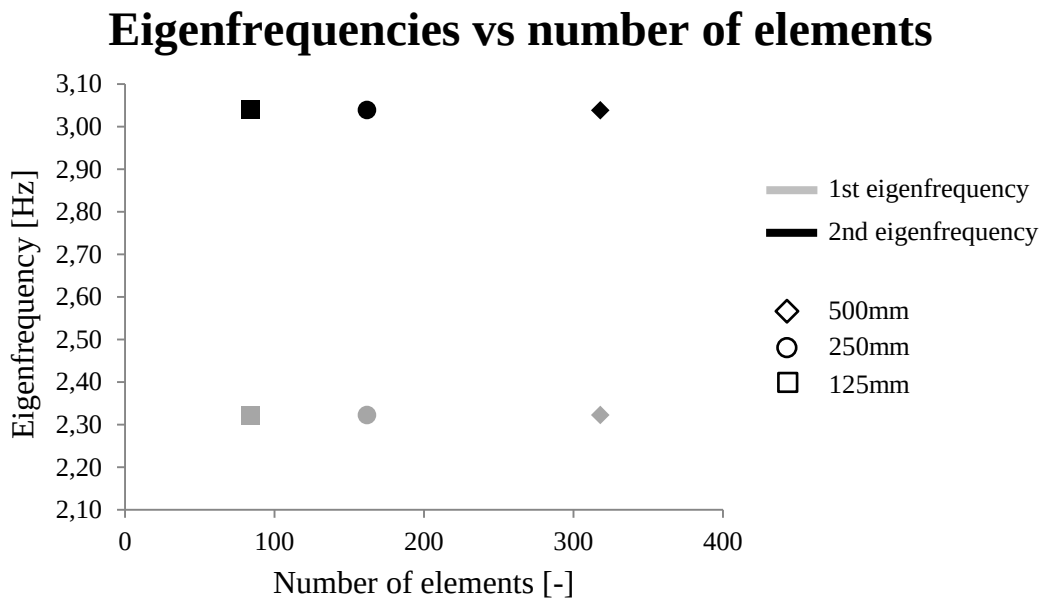


Figure 5.8 - Eigenfrequencies vs number of elements

It is clear that the model had already reached convergence at an element size of 500 mm and therefore this was the element size used in the parametric study for the whole model.

5.4 Analysis of result

The analysis of result consisted of determine whether the first in-wind directional mode for the arch was dominating. By studying the RMS values at the eigenfrequencies of the arches it was possible to determine how much each eigenmode contributed to the total deformation. If it turned out that the first eigenmode had the majority of the contribution it could be stated that the first in-wind directional mode is dominating.

Since the bridge consisted of many structural members the frequency analysis comprised eigenfrequencies for these members as well. Therefore the first step was to identify the eigenfrequencies belonging to the arches. This was performed by studying the deformation in wind direction on the arches. In Figure 5.9 respectively Figure 5.10 the mode shape for eigenmode 1 and 2 is shown.

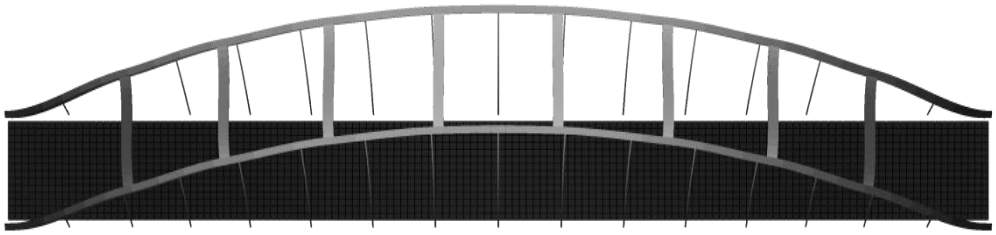


Figure 5.9 – Eigenmode 1 for arch bridge.

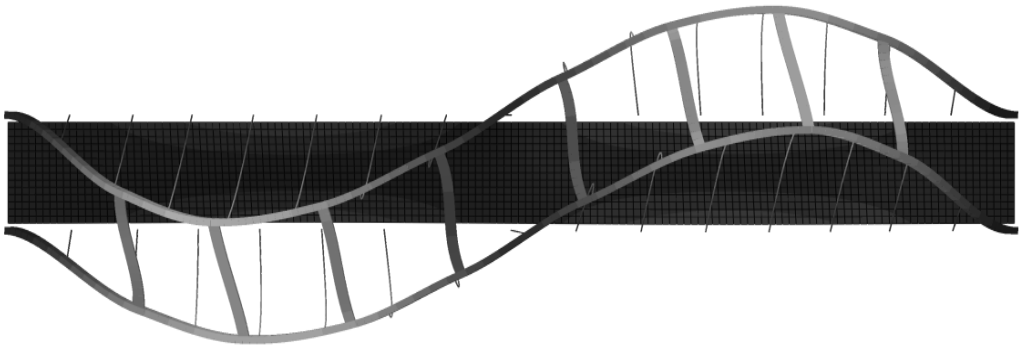


Figure 5.10 – Eigenmode 2 for arch bridge

The identification of the first two eigenmodes for the arches was based on these two figures. The largest deformation in eigenmode 1 will occur in the center of the arch and besides this, the direction of the deformation will be the same for all other nodes. For eigenmode 2 the maximum deformation will occur about 25% respectively 75% into the length of the arch. These maximum values will have different direction and in addition the deformation in the center of the arch will be close to zero. Therefore the deformation values at the nodes placed at 25%, 50% and 75% of the length of the arch were extracted from the .odb-files and gathered in .txt-files.

The RMS plot for a typical arch bridge is shown in Figure 5.11. In this case the deformation is dominated by a frequency at about 0.5Hz. The difference between the

first and the second eigenfrequency, F_1 respectively F_2, is set to ΔF. As seen in the figure the RMS value for F_1 is increasing at a bit higher frequency. Therefore the RMS value of interest is set to the corresponding frequency at $F_1 + 0.1 * \Delta F$.

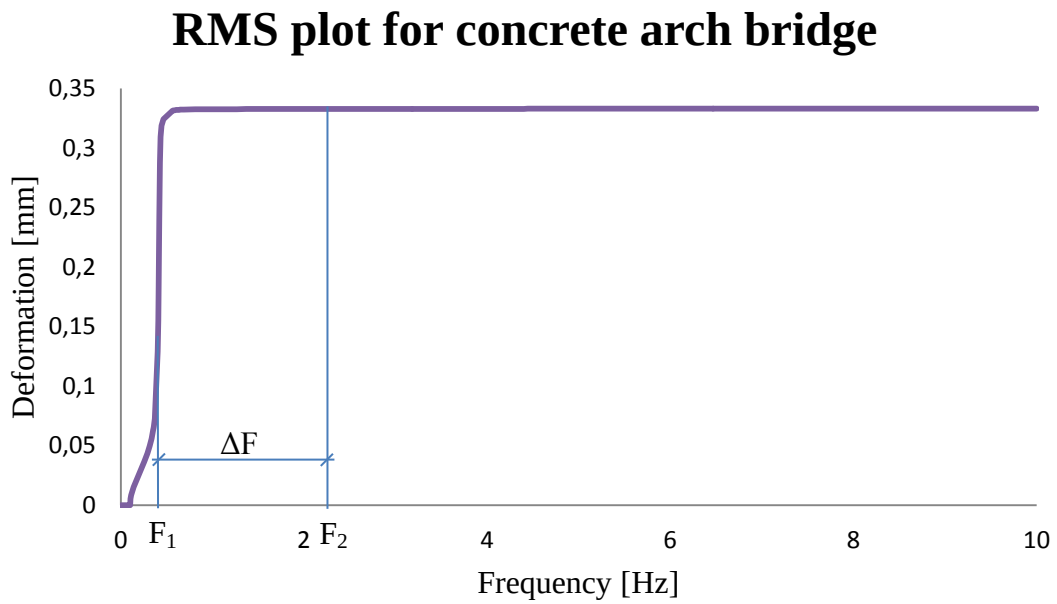


Figure 5.11 – RMS values plotted against a frequency span from 0.1 to 10.0 Hz

The program *ANALYSIS_OF_U_RMS.m* uses the arches deformation to identify the first two eigenfrequencies of the arches. In addition the program will add the corresponding RMS value and evaluate it in comparison to the maximum RMS value.

In a case where the RMS value for $F_1 + 0.1 * \Delta F$ is bigger than 95% of the maximum RMS value, the arch bridge was considered to be dominated by the first in-wind directional mode.

If the parametric study could show that the deformations were dominated by the first in-wind directional mode then the structural factor according to equation (3.42) where to be calculated. Since equation (3.42) is based on a single-mode method it is not valid if several modes contributes to the deformations in the wind direction. Therefore this step where excluded when this condition was not met.

It would be interesting to evaluate the correlation between structural factor and the parameters that is varying in the parametric study. Therefore a multi linear regression analysis where performed in the cases the structural factor where calculated.

6 Results

In this chapter results the parametric study is presented. For all the concrete arch bridges, the dominating mode where explicitly characterized by the first in-wind directional mode. Hence, the structural factor was derived for each combination of parameters. These structural factors have been statistically analyzed through a regression analysis to detect each parameters impact. Regarding the timber and steel arch bridges the dynamic response was more complicated. There were no explicit mode behaviors among these bridges. Hence, timber and steel bridges are treated under the same chapter.

6.1 Concrete arch bridges

The analysis consisted of a parametric study including 3645 concrete arch bridge realizations. The results of the analysis were explicitly indicating that the first in-wind directional mode was dominating in all cases, see *Figure 6.1*. The first eigenfrequency for the arches are varying between 0.10-1.12Hz.

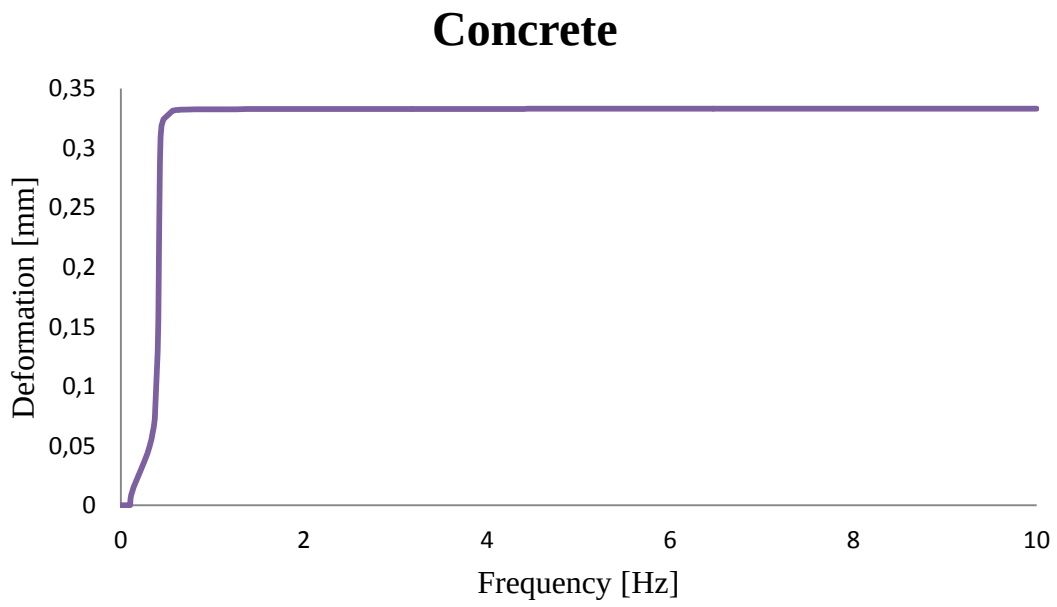


Figure 6.1 – Concrete bridge, RMS-plot

As a consequence of the fact that all concrete arch bridges was dominated by the first in-wind directional mode the structural factor was evaluated for all combinations. These factors were varying from 1.09 to 1.64 depending on the combination of parameters.

The regression analysis gave each parameters impact on the structural factor which is presented in Table 6.1.

Table 6.1 – Presentation of each parameters impact on the structural factor.

	Coefficients	Standard Error	t-stat
Intercept	1.0953	3.79E-03	288.95
Span length. l_s [m]	0.0026	1.64E-05	156.70
Arch width. d_a [m]	0.0065	1.30E-04	50.26
Arch height. h_a [m]	0.0060	5.99E-05	100.83
Number of horizontal bracing. n_{hb} [-]	-0.0249	2.59E-04	-95.82
Width of cross section (Arch). w_{cs} [m]	-0.1909	2.59E-03	-73.58
Height of cross section (Arch). h_{cs} [m]	0.0477	1.73E-03	27.58

The formula derived from the regression analysis is given in (6.1)

$$c_s c_d = \frac{1095.3 + 2.6l_s + 6.5d_a + 6.0h_a - 24.9n_{hb} - 190.9w_{cs} + 47.7h_{cs}}{1000} + residuals \qquad (6.1)$$

The coefficient of determination and standard error are presented in Table 6.2 and the residuals are presented in Figure 6.2.

Table 6.2 – Regression statistics

Regression R^2	0.9352
Standard Error	0.0256

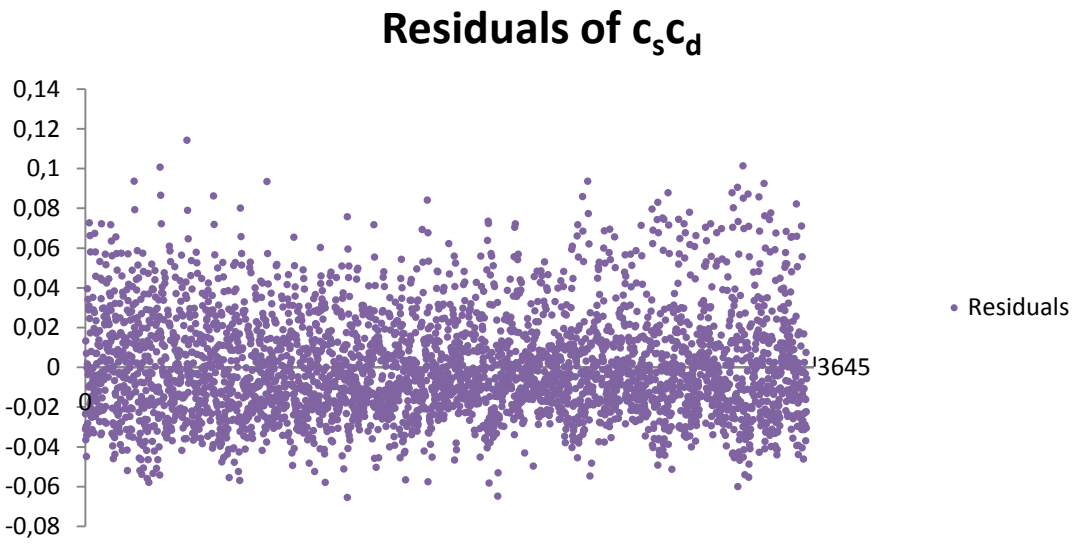


Figure 6.2 – Residuals of $c_s c_d$ with respect to the regression analysis made on concrete arch bridges.

6.2 Timber and steel arch bridges

The analysis of both the steel arches and the timber arches showed a more complicated behavior. Several bridges for both materials did not have deformations

that where dominated by the first in-wind directional mode. These instead had several modes that contributed to the deformations. In Figure 6.3 and Figure 6.4 RMS-plots of one timber and one steel bridge that showed these behaviors are presented.

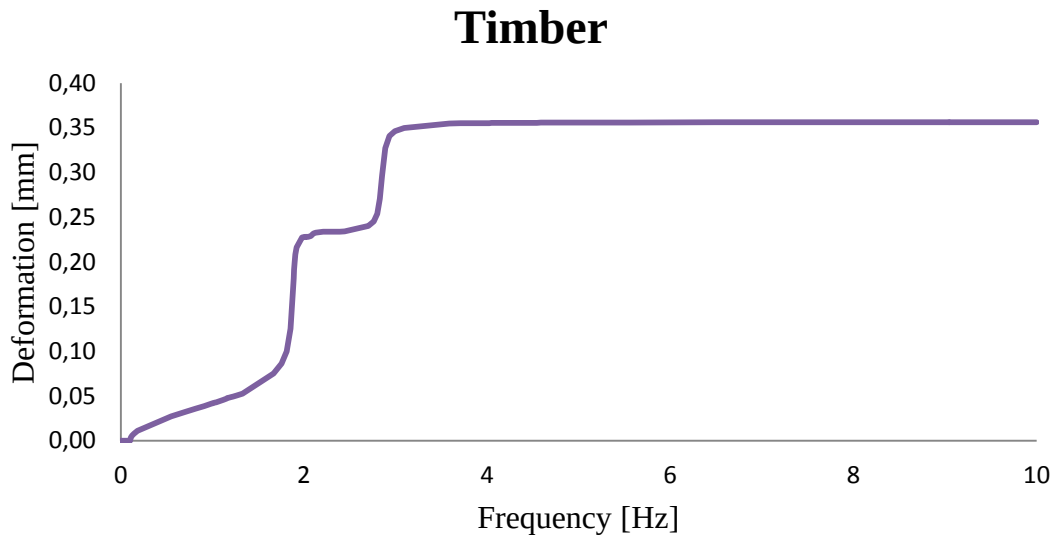


Figure 6.3 - Timber bridge, RMS-plot

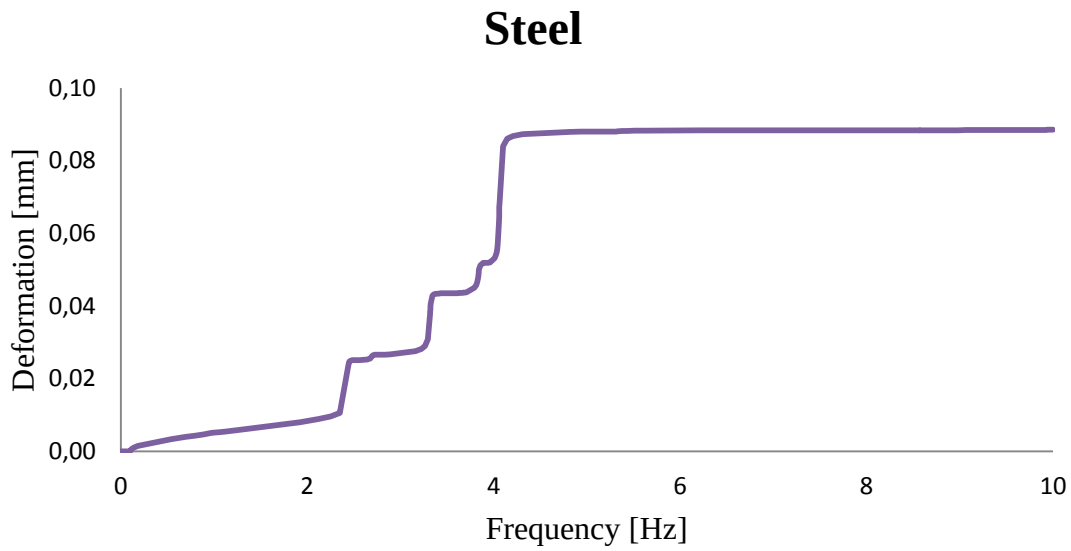


Figure 6.4 - Steel bridge, RMS-plot

Following this results it was investigated if a correlation between the varying parameters, that could predict whether or not the bridge deformations were dominated by the first in-wind directional mode, could be found. However, no such correlation was to be found.

7 Discussion

As presented in the results the dominating mode was only explicit for the concrete arch bridges. In these cases the first in-wind directional mode was dominating for all parametric combinations. However, for timber and steel arch bridges the dynamic response was more complicated. This might be able to be explained by the fact that concrete is considered being a more heavy construction material in comparison with timber and steel.

The structural factor is calculated for the concrete arch bridges. The method used is the one presented in the Swedish Annex (TRVFS, 2011:12), where some of the factors origins from the theory presented in chapter 3. The theory given in the Swedish Annex to Eurocode is a more simplified method than the one given in chapter 3.2.2-3.2.3. However, since the chosen method is accepted by Trafikverket, the Swedish Transport Administration, we consider the method as acceptable.

The regression analysis showed that all chosen parameters had impact on the structural factor. This was an expected results since these parameters where chosen as parameters with strong relation to the dynamic behavior of the structure. To set the values in relation it would have been interesting if a parameter was chosen that was believed not having a big impact on the structural factor.

The derived equation for the structural factor regarding concrete arch bridges presents the impact from each parameter. With increasing span length and arch width the stiffness decreases which could explain why the structural factor is increasing when these parameters are increasing. With increasing arch height the structural factor increases. This is an interesting result since the analyses shows that the overall stiffness in wind direction increases with increasing height. However, it may be explained by the impact of the arch height has on the calculations of structural factor. For example, the higher the arch gets the bigger the wind load gets. The amount of horizontal bracings is increasing the stiffness of bridge which is in line with the structural factor that decreases with increasing number of horizontal bracings. The cross section of arch is also of high importance since these parameters have big impact on the stiffness. An increasing width and decreasing height of the structure will result in a decreasing structural factor. These results can be explained by the fact that an increasing width will have a positive effect on the stiffness to mass ratio comparing an increase of height. The analysis shows that a larger width of the arch increases the value of the first eigenfrequency while a larger height of the arch's cross section will decrease the first eigenfrequency of the arch. This theory is supported by equation (2.34) that displays the importance of stiffness to mass ratio when calculating the eigenfrequencies.

The derived equation from the regression analysis can be seen as an effective way of calculating the structural factor. However, this equation should be treated with carefulness. With the assumptions made for boundary conditions and other structural parameters the equation is not general for all concrete arch bridges. However, it could be used to derive guideline values to compare with the actual value for each case.

For timber and steel arch bridges no obvious correlation could be found between the varying parameters that could explain the dynamic behavior observed in chapter 6.2. It is however quite clear theoretically that the density, elastic modulus and stiffness in wind direction of the arch affect the dynamic behavior significantly and that the combination of at least these may be used to predict this dynamic behavior.

The evaluation method chosen, were based on root mean square values of the arch deformations. This due to the fact that deformations reach convergence faster than for example stresses. The method used, is also a method traditionally used in these types of problems and is approved by the Swedish Road Administration.

8 Conclusions

The parametric study indicates that the dynamic response for arch bridges varies depending on what parameters that are altered. Generally the first eigenfrequency was lower for the concrete arch bridges than for steel and timber arch bridges which can be explained by the stiffness to mass ratio for the different materials and the fact that timber and steel arch bridges were modelled with cross bracing.

For concrete arch bridges the dynamic response in all cases was dominated by the first in-wind directional mode. Hence, the single-mode method could be applied when calculating the structural factor. A regression analysis was made to compare the altered parameters with the associated structural factor. When designing a concrete arch bridge the derived equation from the regression analysis can be used as guideline to compare with the true structural factor.

Regarding timber and steel arch bridges there were no obvious dominating mode. This concludes the fact that the single-mode method is no safe method to use without determining the dominating mode for each case.

8.1 Further studies within the field

Within the field there are several topics that would be interesting for further evaluations.

- This thesis has covered the global dynamic response of arch bridges. In addition to this there are local vibrations that need to be designed for. To extend the study it would be interesting to examine the dynamic behavior of the bridge deck and hangers.

- An interesting topic would be to extend the existing parametric study where more parameters are altered. For example by changing the boundary conditions where the end supports are modelled as hinged connections instead of fixed etc. Further it would be particularly interesting to investigate the influence of stiffness to mass ratio when determining the dominating mode.

- Since the Swedish design code states that the dynamic response should not only be evaluated for arch bridges, it would be of interest to perform similar parametric studies for the other relevant bridge types.

9 References and sources

Birnstiel, C., 2014. *Arch in AccessScience.* McGraw Hill Education ed. Retrived from: http://www.accessscience.com-proxy.lib.chalmers.se/content/arch/047300.

Chen, W.-F. & Duan, L., 2000. *Bridge engineering handbook.* 1st ed. Boca Raton, FL: CRC Press.

Craig Jr, R. & Kurdila, A., 2006. *Fundamentals of Structural Dynamics.* Hoboken NJ: John Wiley & Sons Inc.

Dassault Systèmes, 2012. *Abaqus Theory Manual: version 6.12.* Providence: Dassault Systèmes Simulia Corp.

Dyrbye, C., 1997. *Wind load on Structures.* 1st ed. Chichester: Wiley.

Handa, K., 1982. *Kompendium i byggnadsaerodynamik,* Göteborg: Chalmers Tekniska Högskola.

Holmes, J. D., 2001. *Wind Loading of Structures.* New York: Spon Press.

Mørk, K., Kirkegaard, P. H. & Sørensen, J. D., 1999. *Wind loads on dynamic sensitive structures,* Aalborg: Aalborg University.

SS-EN 14080, 2013. *Timber structures – Glued laminated timber and glued solid timber – Requirements,* Stockholm: Swedish Standards Institute.

SS-EN 1991-1-4, 2005. *Eurocode 1: Actions on structures – Part 1-4: General actions – Wind actions,* Stockholm: Swedish Standards Institue.

SS-EN 1992-1-1, 2005. *Eurocode 2: Design of concrete structures - Part 1-1: General rules and rules for buildings,* Stockholm: Swedish Standards Institue.

SS-EN 1993-1-1, 2005. *Eurocode 3: Design of steel structures – Part 1-1: General rules and rules for buildings,* Stockholm: Swedish Standards Institute.

Strømmen, E., 2010. *Theory of bridge aerodynamics.* 2nd ed. Berlin: Springer.

Trafikverket, 2011. *TRVK Bro 11,* Borlänge: Trafikverket.

TRVFS, 2011:12. *Trafikverkets föreskrifter om ändring i Vägverkets föreskrifter om tillämpningen av europeiska beräkningsstandarder.* Borlänge: Trafikverket.

Zienkiewicz, O. C., Taylor , R. L. & Zhu, J., 2013. *The Finite Element Method: its Basis and Fundamentals.* 7th ed. Oxford: Butterworth-Heinemann.

Appendix A

Concrete arch bridges

Bridge No	Arch material	Span length [m]	Arch height [m]	Cross section of arch	Dim. of arch, wxh [mm]	Dist. Betw. arches [m]	Material of bridge deck	Thickness of deck [mm]	No. transv. bracing	Profile transv. Bracing [mm]	No hangers	Diameter of hanger [mm]	Cross bracing
2-545-1	Concrete	180	33.5	Box	1600x2300	7.5	Concrete	240	4	800x1300	14	63	-
3500-4926-1	Concrete	126	39.0	Rectangular	-	14.6	Concrete	-	6	Rectangular	6	61x16mm	-
17-688-1	Concrete	100	18.8	Rectangular	800x850	7.0	Concrete	270	10	300x1250	18	35	-
20-553-1	Concrete	93	15.5	Rectangular	800x950	6.0	Concrete	300	12	300x1100	24	56	-
24-1191-1	Concrete	75	15.8	Rectangular	650x1200	4.0	Concrete	250	7	400x900	14	50	-
24-617-1	Concrete	75	15.0	Rectangular	600x1400	4.0	Concrete	380	8	300x800	26	40	-
23-144-1	Concrete	70	13.5	Rectangular	700x750	4.0	Concrete	250	8	300x1000	15	45	-
22-814-1	Concrete	70	10.0	Rectangular	700x1050	4.0	Concrete	260	5	400x900	15	45	-
15-99-1	Concrete	48.3	9.0	Rectangular	700x900	6.0	Concrete	140	3	400x1000	13	2x38	-
3500-2780-1	Concrete	47	10.0	Rectangular	-	5.0	Concrete	-	4	-	10	-	-
1783-9-1	Concrete	44.5	11.5	Rectangular	900x1100	5.0	Concrete	400	3	1000x350	10	60	-
24-448-1	Concrete	40	8.0	Rectangular	600x600	6.0	Concrete	340	3	1000x300	16	70	-
24-1163-1	Concrete	40	8.0	Rectangular	675x850	4.0	Concrete	230	4	350x700	12	48	-
25-825-1	Concrete	29	6.0	Rectangular	500x500	4.0	Concrete	200	3	400x400	9	55	-

Steel arch bridges

Bridge No	Arch material	Span length [m]	Arch height [m]	Cross section of arch	Dim. of arch, wxh [mm]	Dist. Betw. arches [m]	Material of bridge deck	Thickness of deck [mm]	No. transv. bracing	Profile transv. Bracing [mm]	No hangers	Diameter of hanger [mm]	Cross bracing
25-573-1	Steel	100	20.0	Box	600x600	6.7	Concrete	274	14	I-profil	21	52	Yes
25-16-1	Steel	70	13.5	I-profil	-	6.0	Concrete+Steel	200	8	U-profil	12	62	Yes
24-558-1	Steel	70	13.0	Box	680x520	7.0	Concrete	250	7	390x610	17	50	-
23-623-1	Steel	70	12.6	Box	620x450	9.0	Concrete	240	6	I-profil	13	64	Yes
17-1238-1	Steel	67.6	13.5	Box	600x950	12.9	Concrete	320	8	KKR 200x200	18	KKR 200x200	Yes
24-1222-1	Steel	60	11.5	Box	600x550	7.0	Concrete	300	9	2st L-vinklar	13	62	Yes
2-1906-1	Steel	54.35	10.5	Quadratic (45deg)	630x630	7.7	Concrete	320	-	-	10	64	-
3500-1587-1	Steel	45	12.5	Box	-	4.9	Steel	-	4	-	11	-	-
1780-1215-1	Steel	42	6.0	Quadratic (45deg)	250x250x16	4.5	Concrete	380	4	230x140x6,3	13	VKR 260x140x6,3	-
4-821-1	Steel	38.4	3.8	VKR	250x250x8	3.9	Steel	-	3	VKR 140x140x8	9	VKR 180x180x10	-
16-270-1	Steel	36.5	6.1	Box	350x210	6.0	Concrete+Steel	280	2	210x350	10	45	-
180-12566-1	Steel	30	3.8	Circular	d = 290	4.1	Concrete+Steel	360	-	-	7	Steel pipes 120	-
25-1265-1	Steel	9.245	16.0	Box	700x600	9.8	Concrete+Steel	230	16	Box	18	62	-

Timber arch bridges

Bridge No	20-1225-1	22-1532-1	2-2124-1	3501-5441-1	22-1538-1
Arch material	Glulam	Glulam	Glulam	Glulam	Glulam
Span length [m]	35	35	34	31.5	29.2
Arch height [m]	8.5	7.0	8.6	6.5	7.9
Cross section of arch	Rectangular	Rectangular	Rectangular	Rectangular	Rectangular
Dim. of arch, wxh [mm]	645x875	215x630	380x900	380x1033	215x1033
Dist. Betw. arches [m]	4.0	2.5	4.9	5.0	3.8
Material of bridge deck	Glulam timber	Glulam timber	Glulam timber	Glulam timber	Glulam timber
Thickness of deck [mm]	280	120	-	325	275
No. transv. bracing	2	8	-	4	6
Profile transv. Bracing [mm]	190x190	90x80	-	115x400	115x400
No hangers	6	8	2	5	6
Diameter of hanger [mm]	165x115 Glulam	20	HEA200	31	24
Cross bracing	Yes	Yes	-	Yes	Yes

Appendix B

Python source code

Appendix B comprises the Python source code of the following scripts:

- PARAMETRIC_STUDY.py
- ARCH_BRIDGE.py
- BRIDGE_DATA.py
- OUTPUT.py

PARAMETRIC_STUDY.py

```
#-----------------------------------------------------------------------
# PARAMETRIC_STUDY.PY
# By: Jonathan Johansson, Daniel Josefsson
#-----------------------------------------------------------------------
from abaqus import*
from abaqusConstants import*
from math import*
from string import*
import __main__
backwardCompatibility.setValues(includeDeprecated=True,reportDeprecat
ed=False)
import sketch
import part
import numpy
import bpCustomData
import regionToolset
import section
import step
import material
import os
import mesh
import job
import time
session.journalOptions.setValues(replayGeometry=COORDINATE)
#RUN INPUT FILE
execfile('BRIDGE_DATA.py')

#SET AN INPUT DIRECTORY AS CWD
cwd=os.getcwd()
os.chdir(cwd+'\\Input')

#Start report file
outputFile = open('Inp_file_report.txt','w')
outputFile.write(time.strftime("%Y-%m-%d  %H:%M:%S",
time.localtime())+'\n')
outputFile.write('Filename structure:\nsp_len - arch_width -
      arch_height - n_hang - phi_hang - n_Hbrac - t_deck - R_a -
      R_b - R_ahb - R_bhb - use_Cbrac - phi_Cbrac\n\n')
for ii in range(len(sp_len)):
    for jj in range(len(arch_width)):
      for kk in range(len(n_hang)):
          for ll in range(len(R_a)):
            for mm in range(len(R_b)):
                for nn in range(len(arch_height)):
                  for pp in range(len(t_deck)):
                      for qq in range(len(phi_hang)):
                        for rr in range(len(n_Hbrac)):
                            for ss in range(len(use_Cbrac)):
                              for tt in range(len(phi_Cbrac)):
                                  for uu in range(len(R_ahb)):
                                    for vv in range(len(R_bhb)):

      name=str(int(sp_len[ii]))+'-'+str(int(arch_width[jj]))+'-
            '+str(int(arch_height[nn]))+'-'+str(int(n_hang[kk]))+'-
            '+str(int(phi_hang[qq]*1000))+'-
            '+str(int(n_Hbrac[rr]))+'-'+str(int(t_deck[pp]*1000))+'-
            '+str(int(R_a[ll]*1000))+'-'+str(int(R_b[mm]*1000))+'-
```

```
                '+str(int(R_ahb[uu]*1000))+'-
                '+str(int(R_bhb[vv]*1000))+'-'+str(int(use_Cbrac[ss]))+'-
                '+str(int(phi_Cbrac[tt]*1000))

                # CONDITIONS

                if (arch_height[nn]-(free_height+deck_height)) > 1.0:
                    if sp_len[ii]>=2*arch_height[nn]:
                       if R_bhb[vv]<=R_b[mm]:

                           path=os.path.dirname(os.getcwd())

                           execfile(path+'\\ARCH_BRIDGE.py')

                           mdb.models['ARCH BRIDGE'].keywordBlock.
                           synchVersions(storeNodesAndElements=False)

                           index=mdb.models['ARCH BRIDGE'].keywordBlock.
                           sieBlocks.index('*Modal Damping,
                           definition=FREQUENCY RANGE\n0.1, 0.016\n10.,
                           0.016')

                           mdb.models['ARCH BRIDGE'].keywordBlock.
                           insert(index, """
**


*Dload, load case=1
ArchL, PZ, 1.
**
*CORRELATION, TYPE=CORRELATED, PSD=rekt
1,1,0
**
                           """)

                           mdb.Job(name=name, model='ARCH BRIDGE')

                           mdb.jobs[name].setValues(description='',
                              memoryUnits=PERCENTAGE, memory=50,
                              getMemoryFromAnalysis=True)

                           mdb.customData.jobName = name

                           mdb.customData.jobModel = 'ARCH BRIDGE'

                           from func.modules.job.freeBodyCut.
                           createFreeBodyCutSurfaces import
                           CreateFreeBodyCutSurfaces

                           CreateFreeBodyCutSurfaces('ARCH BRIDGE','Job-1')

                           mdb.jobs[name].writeInput()

                           outputFile.write(name+' - Input file created\n')

                       else:
                       outputFile.write(name+' - The transverse bracing
                           beam is higher than the arch\n')
```

```
                else:
                outputFile.write(name+' - The arch heigth is to
                     high compared to the span length\n')

            else:
            outputFile.write(name+' - Deck height has to high value
                  compared to Arch height\n')

# Close report file
outputFile.close()

#Set work directory back to script directory
os.chdir(cwd)
```

ARCH_BRIDGE.py

```
#-----------------------------------------------------------------------
# ARCH_BRIDGE.PY
# By: Jonathan Johansson, Daniel Josefsson
#-----------------------------------------------------------------------
# THIS SCRIPT CREATES DIFFERENT ARCH BRIDGES IN ABAQUS IN ORDER TO
# PERFORM A FRQUENCY ANALYSIS AND A RANDOM RESPONSE ANALYSIS. THE
# OUPUT ARE GATHERED IN THE WORK DIRECTORY AS .job-FILES. THE SCRIPT
# IS CONNECTED TO "PARAMETRIC_STUDY.PY" AND CANNOT BE RUN BY IT SELF.

# TABLE OF CONTENTS

# 1. GENERAL
#     1.1. IMPORTS PROGRAMMING COMMANDS FROM ABAQUS
#     1.2. CREATES MODEL
#
# 2. PARTS
#     2.1. ARCH
#     2.2. BRDIGE DECK
#     2.3. HANGERS
#     2.4. HORISONTAL BRACING
#     2.5. CROSS BRACING
# 3. PROPERTIES
#     3.1. MATERIALS
#     3.2. PROFILES AND SECTIONS
#     3.3. ASSIGNS PROPERTIES AND BEAM ORIENTATIONS
# 4. ASSEMBLIES MODEL
# 5. INTERACTION
# 6. MESH
# 7. STEPS
```

```
#<><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><>
# 1. GENERAL
#<><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><>

#----------------------------------------------------------------------
# 1.1. IMPORTS PROGRAMMING COMMANDS FROM ABAQUS
#----------------------------------------------------------------------
# This part of the script loads the neccessary programming commands
# into the script.

from abaqus import*
from abaqusConstants import*
from math import*
from string import*
import __main__
import sketch
import part
import numpy
import bpCustomData
import regionToolset
import section
import step
import material
import os
import mesh
import job
backwardCompatibility.setValues(includeDeprecated=True,
      reportDeprecated=False)
session.journalOptions.setValues(replayGeometry=COORDINATE)

#----------------------------------------------------------------------
# 1.2. CREATES MODEL
#----------------------------------------------------------------------
# The model is created and named "ARCH BRIDGE"

myModel = mdb.Model(name='ARCH BRIDGE')
mySketch = myModel.ConstrainedSketch(name='Sketch ARCH',
      sheetSize=200.0)
```

```
#<><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><>
# 2. PARTS
#<><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><>
free_height=free_height+deck_height
#-----------------------------------------------------------------------
# 2.1. ARCH
#-----------------------------------------------------------------------

myARCH = myModel.Part(name='ARCH', dimensionality=THREE_D,
      type=DEFORMABLE_BODY)

# GEOMETRY FOR ARCH
radius=((sp_len[ii]/2)**2+arch_height[nn]**2)/(2*arch_height[nn])
sec_ang=2*acos((radius-arch_height[nn])/radius)

arch_len=sec_ang*radius

# CALCULATES THE PRLEMINARY NUMBER OF ELEMENTS
n_el=int(ceil(arch_len/opt_el_len))

# CHOOSES AN EVEN NUMBER OF ELEMENTS
if  n_el % 2 == 0:
      n_el=n_el
else:
      n_el=n_el+1

# CREATES X- AND Y-COORDINATES FOR OUTER LINE OF ARCH
real_el_len=arch_len/n_el

Ys=range(n_el+1)
angle=0
for i in range(n_el+1):
      Ys[i]=radius*sin((((pi+sec_ang)/2)-angle))-
            (radius-arch_height[nn])
      angle=angle+real_el_len/radius

Xs=range(n_el+1)
angle=0
for i in range(n_el+1):
      Xs[i]=radius*cos((((pi+sec_ang)/2)-angle))+sp_len[ii]/2
      angle=angle+real_el_len/radius

Zs=[0]*len(Ys)

# CALCULATES DISTANCE dPs FOR EACH ELEMENT
dPs=range(n_el)
for i in range(n_el):
      dPs[i]=sqrt((Xs[i+1]-Xs[i])**2+(Ys[i+1]-Ys[i])**2+
            (Zs[i+1]-Zs[i])**2)

# CREATES CENTRE COORINDATES FOR EACH ELEMENT
Px=range(n_el)
Py=range(n_el)
Pz=[0]*len(range(n_el))
for i in range(n_el):
      Px[i]=(Xs[i+1]-Xs[i])/2+Xs[i]
      Py[i]=(Ys[i+1]-Ys[i])/2+Ys[i]
```

```
# CREATES AND SORTS THE DATUM POINTS
for i in range(len(Xs)):
      myARCH.DatumPointByCoordinate(coords=(Xs[i],Ys[i],Zs[i]))
myARCH_datums_keys = myARCH.datums.keys()
myARCH_datums_keys.sort()
DPs = myARCH.datums

# CREATES A WIRE BETWEEN THE DATUM POINTS
for i in range(n_el):
      myARCH.WirePolyLine(points=((DPs[myARCH_datums_keys[i]],
      DPs[myARCH_datums_keys[i+1]]), ), mergeWire=OFF, meshable=ON)

# CALCULATES SECTION WIDTH OF ARCH PROFILE
if type == 1:
      sec_width=max(I_b1,I_b2)
elif type == 2:
      sec_width=B_a
elif type == 3:
      sec_width=P_r
elif type ==4:
      sec_width=C_r
elif type == 5:
      sec_width=R_a[ll]
elif type == 6:
      sec_width=H_r
elif type == 7:
      sec_width=max(T_a,T_c)
elif type == 8:
      sec_width=L_a
elif type == 9:
      sec_width=T_b

#-------------------------------------------------------------------------
# 2.2. BRIDGE DECK
#-------------------------------------------------------------------------

# CREATES BRIDGE DECK AS PLANAR SHELL ELEMENT
myDECK = mdb.models['ARCH BRIDGE'].ConstrainedSketch(
      name='__profile__', sheetSize=200.0)
myDECK.setPrimaryObject(option=STANDALONE)
myDECK.rectangle(point1=(0, 0), point2=(sp_len[ii],
      (arch_width[jj]-2*sec_width)))
mdb.models['ARCH BRIDGE'].Part(name='Bridge Deck',
dimensionality=THREE_D, type=DEFORMABLE_BODY)
mdb.models['ARCH BRIDGE'].parts['Bridge Deck'].BaseShell(
      sketch=myDECK)
myDECK.unsetPrimaryObject()

#-------------------------------------------------------------------------
# 2.3. HANGERS
#-------------------------------------------------------------------------
myHANGER = myModel.Part(name='Hanger', dimensionality=THREE_D,
      type=DEFORMABLE_BODY)
dH=0.05

sec_ang_h=2*acos((radius-(arch_height[nn]-deck_height))/radius)

deck_len=2*radius*sin(0.5*sec_ang_h)
```

```
d_hang=deck_len/(n_hang[kk]+1)
X_start=(sp_len[ii]-deck_len)/2

# CREATES COORDINATES FOR HANGERS
Xh=range(n_hang[kk])
Yh=range(n_hang[kk])
d=d_hang
for i in range(n_hang[kk]):
      Xh[i]=X_start+d
      Yh[i]=abs(sqrt(radius**2-(Xh[i]-sp_len[ii]/2)**2))-
            (radius-arch_height[nn])
      d=d+d_hang
s1 = mdb.models['ARCH BRIDGE'].ConstrainedSketch(name='__profile__',
      sheetSize=200.0)
for i in range(n_hang[kk]):
      delta=0.01
      Xhf = mdb.models['ARCH
BRIDGE'].parts['ARCH'].vertices.getClosest(
            coordinates=((Xh[i],
      Yh[i],0.0),(Xh[i]+delta,Yh[i]+delta,0.0),))
      Xh[i]=Xhf[0][1][0]
      Yh[i]=(abs(sqrt(radius**2-(Xh[i]-sp_len[ii]/2)**2))-
            (radius-arch_height[nn]))
      s1.Line(point1=(Xh[i], deck_height), point2=(Xh[i], Yh[i]-dH))

p = mdb.models['ARCH BRIDGE'].parts['Hanger']
p.BaseWire(sketch=s1)
s1.unsetPrimaryObject()

#------------------------------------------------------------------------
# 2.4. HORISONTAL BRACING
#------------------------------------------------------------------------
dHb=0.05

if n_Hbrac[rr] > 0:
      #-------------------------------------------------
      # CALCULATES SECTION HEIGHT OF HORISONTAL BRACING
      #-------------------------------------------------
      if typehb == 1:
            h_Hbrac=I_hhb
      elif typehb == 2:
            h_Hbrac=B_bhb
      elif typehb == 3:
            h_Hbrac=2*P_rhb
      elif typehb ==4:
            h_Hbrac=2*C_rhb
      elif typehb == 5:
            h_Hbrac=R_bhb[vv]
      elif typehb == 6:
            h_Hbrac=H_rhb
      elif typehb == 7:
            h_Hbrac=T_bhb
      elif typehb == 8:
            h_Hbrac=L_bhb
      elif typehb == 9:
            h_Hbrac=T_hhb
```

```
      y_frih=free_height+h_Hbrac/2
      x_frih=-sqrt(radius**2-(y_frih+(radius-
arch_height[nn]))**2)+sp_len[ii]/2

      s = mdb.models['ARCH
BRIDGE'].ConstrainedSketch(name='__profile__',
            sheetSize=200.0)
      s.setPrimaryObject(option=STANDALONE)
      s.Line(point1=(0.0, 0.0), point2=(arch_width[jj]-dHb, 0.0))
      p = mdb.models['ARCH BRIDGE'].Part(name='Hbracing',
dimensionality=THREE_D,
            type=DEFORMABLE_BODY)
      p = mdb.models['ARCH BRIDGE'].parts['Hbracing']
      p.BaseWire(sketch=s)
      s.unsetPrimaryObject()
      del mdb.models['ARCH BRIDGE'].sketches['__profile__']

      x_dist=(sp_len[ii]-(2*x_frih))
      theta=acos((x_dist**2-2*radius**2)/(-2*radius**2))

      if n_Hbrac[rr] == 1:
            Yhb=range(n_Hbrac[rr])
            Xhb=range(n_Hbrac[rr])
            Xhbf = mdb.models['ARCH
BRIDGE'].parts['ARCH'].vertices.getClosest(coordinates=(

      (sp_len[ii]/2,arch_height[nn],0.0),(sp_len[ii]/2,arch_height[nn
],0.0),))
            Xhb[0]=Xhbf[0][1][0]
            Yhb[0]=abs(sqrt(radius**2-(Xhb[0]-sp_len[ii]/2)**2))-
(radius-arch_height[nn])
      else:
            dtheta=theta/(n_Hbrac[rr]-1)

            Yhb=range(n_Hbrac[rr])
            angle=0
            for i in range(n_Hbrac[rr]):
                  Yhb[i]=radius*sin((((pi+theta)/2)-angle))-(radius-
arch_height[nn])
                  angle=angle+dtheta

            Xhb=range(n_Hbrac[rr])
            angle=0
            for i in range(n_Hbrac[rr]):
                  Xhb[i]=radius*cos((((pi+theta)/2)-
angle))+sp_len[ii]/2
                  angle=angle+dtheta

            delta=0.01
            for i in range(n_Hbrac[rr]):
                  Xhbf = mdb.models['ARCH
BRIDGE'].parts['ARCH'].vertices.getClosest(coordinates=((Xhb[i],Yhb[i
],0.0),(Xhb[i]+delta,Yhb[i]+delta,0.0),))
                  Xhb[i]=Xhbf[0][1][0]
                  Yhb[i]=abs(sqrt(radius**2-(Xhb[i]-
sp_len[ii]/2)**2))-
                        (radius-arch_height[nn])
```

```
#-------------------------------------------------------------------
# 2.5. CROSS BRACING
#-------------------------------------------------------------------
if n_Hbrac[rr] < 3:
      use_Cbrac[ss]=0
if use_Cbrac[ss]==1:
      n_fack=n_Hbrac[rr]-1
      n_Cbrac=n_fack*2
      d_Cbrac=range(n_fack)

      dCb=0.05
      for i in range(n_fack):
            s = mdb.models['ARCH BRIDGE'].ConstrainedSketch(
            name='__profile__', sheetSize=200.0)
            s.setPrimaryObject(option=STANDALONE)
            d_Cbrac[i]=sqrt((Xhb[i+1]-Xhb[i])**2+(Yhb[i+1]-
Yhb[i])**2+
            arch_width[jj]**2)
            s.Line(point1=(0.0, 0.0), point2=(d_Cbrac[i]-dCb, 0.0))
            p = mdb.models['ARCH BRIDGE'].Part(name='Cbracing-
'+str(i+1),
            dimensionality=THREE_D,type=DEFORMABLE_BODY)
            p = mdb.models['ARCH BRIDGE'].parts['Cbracing-'+str(i+1)]
            p.BaseWire(sketch=s)
            s.unsetPrimaryObject()
            del mdb.models['ARCH BRIDGE'].sketches['__profile__']
```

```
#<><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><>
# 3. PROPERTIES
#<><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><>

#----------------------------------------------------------------------
# 3.1. CREATES MATERIAL
#----------------------------------------------------------------------

myArch=myModel.Material(name='Arch')
elasticProperties = (E_arch,v_arch)
myArch.Elastic(table=(elasticProperties,))
myArch.Density(table=((D_arch,), ))

myDeck=myModel.Material(name='Deck')
elasticProperties = (E_deck,v_deck)
myDeck.Elastic(table=(elasticProperties,))
myDeck.Density(table=((D_deck,), ))

myHang=myModel.Material(name='Hangers')
elasticProperties = (E_hang,v_hang)
myHang.Elastic(table=(elasticProperties,))
myHang.Density(table=((D_hang,), ))

myHbrac=myModel.Material(name='HBracing')
elasticProperties = (E_Hbrac,v_Hbrac)
myHbrac.Elastic(table=(elasticProperties,))
myHbrac.Density(table=((D_Hbrac,), ))

myCbrac=myModel.Material(name='CBracing')
elasticProperties = (E_Cbrac,v_Cbrac)
myCbrac.Elastic(table=(elasticProperties,))
myCbrac.Density(table=((D_Cbrac,), ))

#----------------------------------------------------------------------
# 3.2. PROFILES AND SECTIONS
#----------------------------------------------------------------------

# 3.2.1 CREATES PROFILES FOR ARCH
if type == 1:
      myModel.IProfile(name='PROF', l=I_l, h=I_h, b1=I_b1, b2=I_b2,
      t1=I_t1, t2=I_t2, t3=I_t3)
elif type == 2:
      myModel.BoxProfile(name='PROF', b=B_b, a=B_a,
      uniformThickness=OFF, t1=B_t1, t2=B_t2, t3=B_t3, t4=B_t4)

elif type == 3:
      myModel.PipeProfile(name='PROF', r=P_r, t=P_t)
elif type ==4:
      myModel.CircularProfile(name='PROF', r=C_r)
elif type == 5:
      myModel.RectangularProfile(name='PROF', a=R_a[ll], b=R_b[mm])
elif type == 6:
      myModel.HexagonalProfile(name='PROF', r=H_r, t=H_t)
elif type == 7:
      myModel.TrapezoidalProfile(name='PROF', a=T_a, b=T_b)
elif type == 8:
      myModel.LProfile(name='PROF', a=L_a, b=L_b, t1=L_t1, t2=L_t2)
elif type == 9:
```

```
      myModel.TProfile(name='PROF', b=T_b, h=T_h, l=T_l, tf=T_tf,
      tw=T_tw)

# CREATES SECTION NAMES FOR ARCH
SEKT=range(n_el)
for i in range(n_el):
      SEKT[i]='Section-'+str(i+1)

# CREATES SECTIONS FOR ARCH
for i in range(n_el):
      myModel.BeamSection(name=SEKT[i], profile='PROF',
      integration=DURING_ANALYSIS, poissonRatio=v_arch,
      material= 'Arch', temperatureVar=LINEAR)

# 3.2.2 CREATES PROFILE AND SECTION FOR DECK
mdb.models['ARCH BRIDGE'].HomogeneousShellSection(name='Section-
Deck',
      preIntegrate=OFF, material='Deck', thicknessType=UNIFORM,
      thickness=t_deck[pp], thicknessField='', i
      dealization=NO_IDEALIZATION, poissonDefinition=DEFAULT,
      thicknessModulus=None, temperature=GRADIENT, useDensity=OFF,
      integrationRule=SIMPSON, numIntPts=5)

# 3.2.3. CREATES PROFILES AND SECTIONS FOR HORISONTAL BRACING
if n_Hbrac[rr] > 0:

      if typehb == 1:
            myModel.IProfile(name='PROF-hb', l=I_lhb, h=I_hhb,
b1=I_b1hb,
            b2=I_b2hb, t1=I_t1hb, t2=I_t2hb, t3=I_t3hb)
      elif typehb == 2:
            myModel.BoxProfile(name='PROF-hb', b=B_bhb, a=B_ahb,
            uniformThickness=OFF, t1=B_t1hb, t2=B_t2hb, t3=B_t3hb,
            t4=B_t4hb)
      elif typehb == 3:
            myModel.PipeProfile(name='PROF-hb', r=P_rhb, t=P_thb)
      elif typehb ==4:
            myModel.CircularProfile(name='PROF-hb', r=C_rhb)
      elif typehb == 5:
            myModel.RectangularProfile(name='PROF-hb', a=R_ahb[uu],
            b=R_bhb[vv])
      elif typehb == 6:
            myModel.HexagonalProfile(name='PROF-hb', r=H_rhb,
t=H_thb)
      elif typehb == 7:
            myModel.TrapezoidalProfile(name='PROF-hb', a=T_ahb,
b=T_bhb)
      elif typehb == 8:
            myModel.LProfile(name='PROF-hb', a=L_ahb, b=L_bhb,
            t1=L_t1hb, t2=L_t2hb)
      elif typehb == 9:
            myModel.TProfile(name='PROF-hb', b=T_bhb, h=T_hhb,
            l=T_lhb, tf=T_tfhb, tw=T_twhb)

      myModel.BeamSection(name='Section-HB', profile='PROF-hb',
      integration=DURING_ANALYSIS, poissonRatio=v_Hbrac,
      material= 'HBracing', temperatureVar=LINEAR)
```

```
# 3.2.4. CREATES PROFILES AND SECTIONS FOR CROSS BRACING
if use_Cbrac[ss]==1:
      n_fack=n_Hbrac[rr]-1
      n_Cbrac=n_fack*2

      myModel.CircularProfile(name='PROF-Cbrac', r=phi_Cbrac[tt])
      myModel.BeamSection(name='Cbrac', profile='PROF-Cbrac',
      integration=DURING_ANALYSIS, poissonRatio=v_Hbrac,
      material= 'CBracing', temperatureVar=LINEAR)

#-------------------------------------------------------------------
# 3.3. ASSIGNS SECTIONS AND BEAM ORIENTATION
#-------------------------------------------------------------------

# 3.3.1. ASSIGNS SECTION AND BEAM ORIENTATION FOR ARCH

# CREATES SET NAMES     FOR ARCH
set=range(n_el)
for i in range(n_el):
      set[i]='set-'+str(i+1)
      edg = myARCH.edges.findAt(((Px[i],Py[i],Pz[i]),),)
      reg = myModel.parts['ARCH'].Set(edges=edg, name=set[i])
      myARCH.SectionAssignment(region=reg,sectionName=SEKT[i])
      p = mdb.models['ARCH BRIDGE'].parts['ARCH']
      region=p.sets[set[i]]
      p = mdb.models['ARCH BRIDGE'].parts['ARCH']
      p.assignBeamSectionOrientation(region=region,
method=N1_COSINES,
      n1=(0.0, 0.0, 1.0))

# 3.3.2. ASSIGNS SECTION AND BEAM ORIENTATION FOR DECK
p = mdb.models['ARCH BRIDGE'].parts['Bridge Deck']
f = p.faces
faces = f.getSequenceFromMask(mask=('[#1 ]', ), )
region = regionToolset.Region(faces=faces)
p = mdb.models['ARCH BRIDGE'].parts['Bridge Deck']
p.SectionAssignment(region=region, sectionName='Section-Deck',
      offset=0.0, offsetType=MIDDLE_SURFACE, offsetField='',
      thicknessAssignment=FROM_SECTION)

# 3.3.3. ASSIGNS SECTION AND BEAM ORIENTATION FOR HANGERS
myModel.CircularProfile(name='PROF-hang', r=phi_hang[qq])
myModel.BeamSection(name='Hanger', profile='PROF-hang',
      integration=DURING_ANALYSIS, poissonRatio=v_Hbrac,
      material= 'Hangers', temperatureVar=LINEAR)
p = mdb.models['ARCH BRIDGE'].parts['Hanger']
e = p.edges
# ASSIGNS SECTIONS AND BEAM ORIENTATION FOR HANGERS
for i in range(n_hang[kk]):
      edges = e.findAt(((Xh[i], Yh[i]-dH, 0.0), ),)
      region = regionToolset.Region(edges=edges)
      p = mdb.models['ARCH BRIDGE'].parts['Hanger']
      p.SectionAssignment(region=region, sectionName='Hanger',
            offset=0.0, offsetType=MIDDLE_SURFACE, offsetField='',
            thicknessAssignment=FROM_SECTION)
      p.assignBeamSectionOrientation(region=region,
            method=N1_COSINES, n1=(1.0, 0.0, 0.0))
```

```
# 3.3.4. ASSIGNS SECTION AND BEAM ORIENTATION FOR HORISONTAL BRACING
dHb=0.05
if n_Hbrac[rr] > 0:
      # ASSINGS SECTIONS FOR HORISONTAL BRACING
      p = mdb.models['ARCH BRIDGE'].parts['Hbracing']
      e = p.edges
      edges = e.findAt(((0.0, 0.0, 0.0), ))
      region = regionToolset.Region(edges=edges)
      p = mdb.models['ARCH BRIDGE'].parts['Hbracing']
      p.SectionAssignment(region=region, sectionName='Section-HB',
            offset=0.0, offsetType=MIDDLE_SURFACE, offsetField='',
            thicknessAssignment=FROM_SECTION)


      # ASSIGNS BEAM SECTION ORIENTATION FOR HORISONTAL BRACING
      for i in range(n_Hbrac[rr]):
            p = mdb.models['ARCH BRIDGE'].parts['Hbracing']
            e = p.edges
            edges = e.findAt(((0.0, 0.0, 0.0), ))
            region=regionToolset.Region(edges=edges)
            p.assignBeamSectionOrientation(region=region,
                  method=N1_COSINES, n1=(0.0, 0.0, 1.0))

# 3.3.5. ASSIGNS SECTION AND BEAM ORIENTATION FOR CROSS BRACING
if use_Cbrac[ss]==1:
      for i in range(n_fack):
            p = mdb.models['ARCH BRIDGE'].parts['Cbracing-'+str(i+1)]
            e = p.edges
            edges = e.findAt(((0.0, 0.0, 0.0), ))
            region = regionToolset.Region(edges=edges)
            p = mdb.models['ARCH BRIDGE'].parts['Cbracing-'+str(i+1)]
            p.SectionAssignment(region=region, sectionName='Cbrac',
                  offset=0.0, offsetType=MIDDLE_SURFACE,
offsetField='',
                  thicknessAssignment=FROM_SECTION)
            p.assignBeamSectionOrientation(region=region,
                  method=N1_COSINES, n1=(0.0, 0.0, 1.0))

            zz=range(len(d_Cbrac))
            xx=range(len(d_Cbrac))
            yy=range(len(d_Cbrac))
```

```
#<><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><>
# 4. ASSEMBLIES MODEL
#<><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><>
# ASSSEMBLIES ARCHES
a = mdb.models['ARCH BRIDGE'].rootAssembly
a.DatumCsysByDefault(CARTESIAN)
p = mdb.models['ARCH BRIDGE'].parts['ARCH']
a.Instance(name='ARCH-1', part=p, dependent=OFF)
a.LinearInstancePattern(instanceList=('ARCH-1', ),
      direction1=(1.0, 0.0, 0.0), direction2=(0.0, 1.0, 0.0),
      number1=1, number2=2, spacing1=0.0, spacing2=0.0)
a.translate(instanceList=('ARCH-1-lin-1-2', ), vector=(0.0, 0.0,
      arch_width[jj]))
p = mdb.models['ARCH BRIDGE'].parts['Bridge Deck']
a.Instance(name='Bridge Deck-1', part=p, dependent=OFF)
a.rotate(instanceList=('Bridge Deck-1', ),
      axisPoint=(0.0, 0.0, 0.0), axisDirection=(1.0, 0.0, 0.0),
      angle=90.0)
a.translate(instanceList=('Bridge Deck-1', ),
      vector=(0.0, 0.0, sec_width))
a.translate(instanceList=('Bridge Deck-1', ),
      vector=(0.0, deck_height, 0.0))

a = mdb.models['ARCH BRIDGE'].rootAssembly
e1 = a.instances['ARCH-1'].edges
e2 = a.instances['ARCH-1-lin-1-2'].edges
a.Set(edges=e1, name='ArchL')
a.Set(edges=e2, name='ArchR')

# ASSEMBLIES HANGERS
p = mdb.models['ARCH BRIDGE'].parts['Hanger']
a.Instance(name='Hanger-1', part=p, dependent=OFF)
a.LinearInstancePattern(instanceList=('Hanger-1', ),
      direction1=(1.0, 0.0, 0.0), direction2=(0.0, 1.0, 0.0),
      number1=1, number2=2, spacing1=0.0, spacing2=0.0)
a.translate(instanceList=('Hanger-1-lin-1-2', ), vector=(0.0, 0.0,
      arch_width[jj]))

# ASSEMBLIES HORISONTAL BRACING
if n_Hbrac[rr] > 0:
      a1 = mdb.models['ARCH BRIDGE'].rootAssembly
      p = mdb.models['ARCH BRIDGE'].parts['Hbracing']
      for i in range(n_Hbrac[rr]):
            a1.Instance(name='Hbracing-'+str(i+1), part=p,
dependent=OFF)
            a1.rotate(instanceList=('Hbracing-'+str(i+1), ),
            axisPoint=(0.0, 0.0, 0.0), axisDirection=(0.0, 1.0, 0.0),
            angle=270.0)
            a1.translate(instanceList=('Hbracing-'+str(i+1), ),
                  vector=(Xhb[i], Yhb[i], dHb/2))

# ASSEMBLIES CROSS BRACING
if n_Hbrac[rr] < 0:
      use_Cbrac[ss]=0
if use_Cbrac[ss]==1:
      zz=range(len(d_Cbrac))
      xx=range(len(d_Cbrac))
      yy=range(len(d_Cbrac))
```

```
for i in range(len(d_Cbrac)):
      alpha=asin((Yhb[i+1]-Yhb[i])/d_Cbrac[i])*180/pi
      beta=atan(arch_width[jj]/(Xhb[i+1]-Xhb[i]))*180/pi
      zeta=atan((Yhb[i+1]-Yhb[i])/arch_width[jj])*180/pi
      yy[i]=sin(alpha*pi/180)*dCb/2
      if zeta==0:
            zz[i]=sin(beta*pi/180)*dCb/2
      else:
            zz[i]=yy[i]/tan(zeta*pi/180)
      xx[i]=sqrt((dCb/2)**2-yy[i]**2-zz[i]**2)

      a1 = mdb.models['ARCH BRIDGE'].rootAssembly
      p = mdb.models['ARCH BRIDGE'].parts['Cbracing-'+str(i+1)]
      a1.Instance(name='Cbracing-'+str(i+1)+'-1', part=p,
            dependent=OFF)
      a1.translate(instanceList=('Cbracing-'+str(i+1)+'-1', ),
            vector=(xx[i]+Xhb[i], yy[i]+Yhb[i], zz[i]))
      a1.rotate(instanceList=('Cbracing-'+str(i+1)+'-1', ),
            axisPoint=(xx[i]+Xhb[i], yy[i]+Yhb[i], zz[i]),
            axisDirection=(0.0, 0.0, 1.0), angle=alpha)
      a1.rotate(instanceList=('Cbracing-'+str(i+1)+'-1', ),
            axisPoint=(xx[i]+Xhb[i], yy[i]+Yhb[i], zz[i]),
            axisDirection=(0.0, -1.0, 0.0), angle=beta)

Xdp=range(n_fack)
Ydp=range(n_fack)
Xnext=range(n_fack)
Ynext=range(n_fack)
theta=range(n_fack)
for i in range(len(d_Cbrac)):
      alpha=asin((Yhb[i+1]-Yhb[i])/d_Cbrac[i])*180/pi
      beta=atan(arch_width[jj]/(Xhb[i+1]-Xhb[i]))*180/pi
      zeta=atan((Yhb[i+1]-Yhb[i])/arch_width[jj])*180/pi

      yy[i]=sin(alpha*pi/180)*dCb/2
      if zeta==0:
            zz[i]=sin(beta*pi/180)*dCb/2
      else:
            zz[i]=yy[i]/tan(zeta*pi/180)
      xx[i]=sqrt((dCb/2)**2-yy[i]**2-zz[i]**2)


      a1 = mdb.models['ARCH BRIDGE'].rootAssembly
      p = mdb.models['ARCH BRIDGE'].parts['Cbracing-'+str(i+1)]
      a1.Instance(name='Cbracing-'+str(i+1)+'-2', part=p,
            dependent=OFF)
      a1.translate(instanceList=('Cbracing-'+str(i+1)+'-2', ),
            vector=(Xhb[i]+xx[i], Yhb[i]+yy[i],
            arch_width[jj]-zz[i]))
      a1.rotate(instanceList=('Cbracing-'+str(i+1)+'-2', ),
            axisPoint=(Xhb[i]+xx[i], Yhb[i]+yy[i],
            arch_width[jj]-zz[i]),
            axisDirection=(0.0, 0.0, 1.0), angle=alpha)
      a1.rotate(instanceList=('Cbracing-'+str(i+1)+'-2', ),
            axisPoint=(Xhb[i]+xx[i], Yhb[i]+yy[i],
            arch_width[jj]-zz[i]), axisDirection=(0.0, 1.0,
            0.0), angle=beta)
```

```
#<><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><>
# 5. INTERACTION
#<><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><>

mdb.models['ARCH BRIDGE'].ConnectorSection(name='Pin-fixed',
      assembledType=UJOINT)
mdb.models['ARCH BRIDGE'].ConnectorSection(name='Pin',
      translationalType=JOIN, rotationalType=ROTATION)
mdb.models['ARCH BRIDGE'].ConnectorSection(name='Fixed',
      translationalType=JOIN, rotationalType=ALIGN)
datum1 = mdb.models['ARCH BRIDGE'].rootAssembly.datums[1]

#-------------------------------------------------------------------
# ADDING COUPLING BETWEEN HANGERS AND BRIDGE DECK
#-------------------------------------------------------------------
# ONLY RUN IF HANGERS EXISTS
if n_hang[kk] > 0:
      #CREATES PARTITIONS IN THE BRIDGE DECK BETWEEN THE HANGERS
      a = mdb.models['ARCH BRIDGE'].rootAssembly
      for i in range(n_hang[kk]):
            v1 = a.instances['Hanger-1'].vertices
            v2 = a.instances['Hanger-1-lin-1-2'].vertices
            f1 = a.instances['Bridge Deck-1'].faces
            pickedFaces = f1.findAt(((Xh[i], deck_height,
                  arch_width[jj]/2), ))
            a.PartitionFaceByShortestPath(point1=v1.findAt(
                  coordinates=(Xh[i], deck_height, 0.0)),
                  point2=v2.findAt(coordinates=(Xh[i], deck_height,
                  arch_width[jj])), faces=pickedFaces)
      # DEFINE COUPLING BETWEEN THE HANGERS AND BRIDGE DECK

      v11 = a.instances['Bridge Deck-1'].vertices
      v12 = a.instances['Hanger-1'].vertices
      for i in range(n_hang[kk]):
            a = mdb.models['ARCH BRIDGE'].rootAssembly
            v1 = a.instances['Bridge Deck-1'].vertices
            v2 = a.instances['Hanger-1'].vertices
            a.WirePolyLine(points=((v1.findAt(coordinates=(Xh[i],
                  deck_height, sec_width)),
                  v2.findAt(coordinates=(Xh[i], deck_height, 0.0))),
),
                  mergeWire=OFF, meshable=OFF)
            e1 = a.edges
            edges1 = e1.findAt(((Xh[i], deck_height, sec_width/2), ))
            region=regionToolset.Region(edges=edges1)

            csa = a.SectionAssignment(sectionName='Pin-fixed',
                  region=region)
            a.ConnectorOrientation(region=csa.getSet(),
                  localCsys1=datum1)

            v3 = a.instances['Bridge Deck-1'].vertices
            v4 = a.instances['Hanger-1-lin-1-2'].vertices
            a.WirePolyLine(points=((v3.findAt(coordinates=(Xh[i],
                  deck_height,
            arch_width[jj]-sec_width)), v4.findAt(coordinates=(Xh[i],
                  deck_height, arch_width[jj]))), ),
                  mergeWire=OFF, meshable=OFF)
```

```
            e1 = a.edges
            edges1 = e1.findAt(((Xh[i],deck_height,arch_width[jj]),))
            region=regionToolset.Region(edges=edges1)
            csa = a.SectionAssignment(sectionName='Pin-fixed',
                  region=region)
            a.ConnectorOrientation(region=csa.getSet(),
                  localCsys1=datum1)

#-----------------------------------------------------------------------
# ADDING COUPLING BETWEEN HANGERS AND ARCH
#-----------------------------------------------------------------------
if n_hang[kk] > 0:
            for i in range(n_hang[kk]):
                  a = mdb.models['ARCH BRIDGE'].rootAssembly
                  v1 = a.instances['ARCH-1-lin-1-2'].vertices
                  v2 = a.instances['Hanger-1-lin-1-2'].vertices

      a.WirePolyLine(points=((v1.findAt(coordinates=(Xh[i],
            Yh[i], arch_width[jj])),v2.findAt(coordinates=(Xh[i],
            Yh[i]-dH, arch_width[jj]))), ),mergeWire=OFF,
            meshable=OFF)
                  e1 = a.edges
                  edges1 = e1.findAt(((Xh[i],Yh[i],arch_width[jj]),))
                  region=regionToolset.Region(edges=edges1)
                  a.SectionAssignment(region=region,
                  sectionName='Pin')

                  v3 = a.instances['ARCH-1'].vertices
                  v4 = a.instances['Hanger-1'].vertices

      a.WirePolyLine(points=((v3.findAt(coordinates=(Xh[i],
                  Yh[i], 0.0)), v4.findAt(coordinates=(Xh[i],
                  Yh[i]-dH,0.0))), ),mergeWire=OFF,meshable=OFF)
                  e1 = a.edges
                  edges1 = e1.findAt(((Xh[i], Yh[i], 0.0), ))
                  region=regionToolset.Region(edges=edges1)
                  a.SectionAssignment(region=region,
                        sectionName='Pin')

#-----------------------------------------------------------------------
# ADDING COUPLING BETWEEN HORIZONTAL BRACING AND ARCH
#-----------------------------------------------------------------------
if n_Hbrac[rr] > 0:
      for i in range(n_Hbrac[rr]):
            a = mdb.models['ARCH BRIDGE'].rootAssembly
            v1 = a.instances['ARCH-1-lin-1-2'].vertices
            v2 = a.instances['Hbracing-'+str(i+1)].vertices
            a.WirePolyLine(points=((v1.findAt(coordinates=(Xhb[i],
                  Yhb[i], arch_width[jj])),
v2.findAt(coordinates=(Xhb[i],
                  Yhb[i], arch_width[jj]-(dHb/2)))), ),
                  mergeWire=OFF, meshable=OFF)
            e1 = a.edges
            edges1 = e1.findAt(((Xhb[i], Yhb[i],
                  arch_width[jj]-(dHb/2)), ))
            region=regionToolset.Region(edges=edges1)
            a.SectionAssignment(region=region, sectionName='Fixed')
```

```
            v1 = a.instances['ARCH-1'].vertices
            v2 = a.instances['Hbracing-'+str(i+1)].vertices
            a.WirePolyLine(points=((v1.findAt(coordinates=(Xhb[i],
                  Yhb[i], 0)), v2.findAt(coordinates=(Xhb[i], Yhb[i],
                  (dHb/2)))), ), mergeWire=OFF, meshable=OFF)
            e1 = a.edges
            edges1 = e1.findAt(((Xhb[i], Yhb[i],(dHb/2)), ))
            region=regionToolset.Region(edges=edges1)
            csa = a.SectionAssignment(sectionName='Fixed',
                  region=region)
            a.ConnectorOrientation(angle1=90.0, region=csa.getSet(),
                  localCsys1=datum1)

#------------------------------------------------------------------------
# ADDING COUPLING BETWEEN CROSS-BRACING BRACING AND ARCH
#------------------------------------------------------------------------
if use_Cbrac[ss]==1:
      # Create partition in the Arch alligned with the cross-bracing
      for i in range(n_Hbrac[rr]-1):
            dist=sqrt(Xhb[i]**2+(Yhb[i]**2))
            angle=acos((dist**2-2*radius**2)/(-2*radius**2))+
                  real_el_len/radius
            Ynext[i]=radius*sin(((pi+sec_ang)/2)-angle)-
                  (radius-arch_height[nn])
            Xnext[i]=radius*cos(((pi+sec_ang)/2)-angle)+sp_len[ii]/2
            theta[i]=atan((Ynext[i]-Yhb[i])/(Xnext[i]-Xhb[i]))
            Xdp[i]=Xhb[i]+xx[i]
            Ydp[i]=Yhb[i]+(tan(theta[i])*xx[i])
      for i in range(n_Hbrac[rr]-1):
            a.DatumPointByCoordinate(coords=(Xdp[i], Ydp[i], 0.0))
      for i in range(n_Hbrac[rr]-1):
            a.DatumPointByCoordinate(coords=(Xdp[i], Ydp[i],
                  arch_width[jj]))
      e11 = a.instances['ARCH-1'].edges
      e12 = a.instances['ARCH-1-lin-1-2'].edges
      d11 = a.datums
      datum_list=d11.keys()
      for i in range(n_Hbrac[rr]-1):
            a.PartitionEdgeByPoint(edge=e11.findAt(coordinates=(
                  Xdp[i], Ydp[i], 0.0)), point=d11[datum_list[i+1]])
      for i in range(n_Hbrac[rr]-1):

      a.PartitionEdgeByPoint(edge=e12.findAt(coordinates=(Xdp[i],
                  Ydp[i], arch_width[jj])),
                  point=d11[datum_list[n_Hbrac[rr]+i]])
      Xdp2=range(n_fack)
      Ydp2=range(n_fack)
      Xnext=range(n_fack)
      Ynext=range(n_fack)
      theta=range(n_fack)

      for i in range(n_fack):
            dist=sqrt(Xhb[i+1]**2+(Yhb[i+1]**2))
            angle=acos((dist**2-2*radius**2)/(-2*radius**2))-
                  real_el_len/radius
            Ynext[i]=radius*sin(((pi+sec_ang)/2)-angle)-
                  (radius-arch_height[nn])
            Xnext[i]=radius*cos(((pi+sec_ang)/2)-angle)+sp_len[ii]/2
```

```
            theta[i]=atan((Xnext[i]-Xhb[i+1])/(Ynext[i]-Yhb[i+1]))
            Xdp2[i]=Xhb[i+1]-xx[i]
            Ydp2[i]=Yhb[i+1]-(xx[i]/tan(theta[i]))
      for i in range(n_fack):
            a.DatumPointByCoordinate(coords=(Xdp2[i], Ydp2[i], 0.0))
      for i in range(n_fack):
            a.DatumPointByCoordinate(coords=(Xdp2[i], Ydp2[i],
                  arch_width[jj]))
      e11 = a.instances['ARCH-1'].edges
      e12 = a.instances['ARCH-1-lin-1-2'].edges
      d11 = a.datums
      datum_list=d11.keys()
      for i in range(n_fack):

      a.PartitionEdgeByPoint(edge=e11.findAt(coordinates=(Xdp2[i],
                  Ydp2[i], 0.0)),
point=d11[datum_list[2*n_fack+1+i]])
      for i in range(n_fack):
            a.PartitionEdgeByPoint(edge=e12.findAt(coordinates=(
                  Xdp2[i], Ydp2[i], arch_width[jj])),
                  point=d11[datum_list[3*n_fack+1+i]])

if use_Cbrac[ss]==1:
      a.DatumCsysByThreePoints(name='Datum csys-2',
            coordSysType=CARTESIAN, origin=(0.0, 0.0, 0.0),
            point1=(0.0, -1.0, 0.0), point2=(1.0, 0.0, 0.0))
      csysdatum=mdb.models['ARCH BRIDGE'].rootAssembly.datums.keys()

      datum11 = mdb.models['ARCH BRIDGE'].rootAssembly.datums[
            csysdatum[len(csysdatum)-1]]
      Xcb=range(len(Yhb))
      Ycb=range(len(Yhb))
      Zcb=range(len(Yhb))
      for i in range(n_fack):
            a = mdb.models['ARCH BRIDGE'].rootAssembly
            v1 = a.instances['ARCH-1'].vertices
            v2 = a.instances['Cbracing-'+str(i+1)+'-1'].vertices
            a.WirePolyLine(points=((v1.findAt(coordinates=(Xdp[i],
                  Ydp[i], 0)), v2.findAt(coordinates=(xx[i]+Xhb[i],
                  yy[i]+Yhb[i],zz[i])))),),mergeWire=OFF,meshable=OFF)
            e1 = a.edges
            edges2 = e1.findAt(((xx[i]+Xhb[i],yy[i]+Yhb[i],zz[i]), ))
            region=regionToolset.Region(edges=edges2)

            v1 = a.instances['ARCH-1-lin-1-2'].vertices
            v2 = a.instances['Cbracing-'+str(i+1)+'-1'].vertices

            Coord = v2.getClosest(coordinates=((Xhb[i+1]-xx[i],
                  Yhb[i+1]-yy[i],arch_width[jj]-zz[i]),(Xhb[i+1]-
                  xx[i],Yhb[i+1]-yy[i],arch_width[jj]-zz[i]),))

            Xcb[i+1] = Coord[0][1][0]
            Ycb[i+1] = Coord[0][1][1]
            Zcb[i+1] = Coord[0][1][2]

            a.DatumCsysByThreePoints(name='Datum csys-2',
                  coordSysType=CARTESIAN, origin=(Xhb[i]+xx[i],
                  Yhb[i]+yy[i], zz[i]), point1=(Xcb[i+1], Ycb[i+1],
```

```
                  Zcb[i+1]), point2=(Xhb[i]+xx[i], Yhb[i]+yy[i],
                  arch_width[jj]-zz[i]))
            csysdatum=mdb.models['ARCH
BRIDGE'].rootAssembly.datums.keys()
            datum11 = mdb.models['ARCH BRIDGE'].rootAssembly.datums[
                  csysdatum[len(csysdatum)-1]]

            csa = a.SectionAssignment(sectionName='Pin-fixed',
                  region=region)
            a.ConnectorOrientation(angle1=-90, axis1=AXIS_3,
                  region=csa.getSet(), localCsys1=datum11)

            a.WirePolyLine(points=((v1.findAt(coordinates=(Xdp2[i],
                  Ydp2[i], arch_width[jj])), v2.findAt(coordinates=(
                        Xcb[i+1], Ycb[i+1],Zcb[i+1]))), ),
                        mergeWire=OFF, meshable=OFF)
            e1 = a.edges
            edges1 = e1.findAt(((Xcb[i+1], Ycb[i+1],Zcb[i+1]), ))
            region=regionToolset.Region(edges=edges1)
            a.SectionAssignment(region=region, sectionName='Pin')

      Xcb=range(len(Yhb))
      Ycb=range(len(Yhb))
      Zcb=range(len(Yhb))
      for i in range(n_fack):
            a = mdb.models['ARCH BRIDGE'].rootAssembly
            v1 = a.instances['ARCH-1-lin-1-2'].vertices
            v2 = a.instances['Cbracing-'+str(i+1)+'-2'].vertices
            a.WirePolyLine(points=((v1.findAt(coordinates=(Xdp[i],
                  Ydp[i], arch_width[jj])), v2.findAt(coordinates=(
                  Xhb[i]+xx[i],Yhb[i]+yy[i],arch_width[jj]-
                  zz[i]))),), mergeWire=OFF, meshable=OFF)
            e1 = a.edges
            edges2 = e1.findAt(((Xhb[i]+xx[i], Yhb[i]+yy[i],
                  arch_width[jj]-zz[i]), ))
            region=regionToolset.Region(edges=edges2)

            v1 = a.instances['ARCH-1'].vertices
            v2 = a.instances['Cbracing-'+str(i+1)+'-2'].vertices

            Coord = v2.getClosest(coordinates=((Xhb[i+1]-xx[i],
                  Yhb[i+1]-yy[i], zz[i]),(Xhb[i+1]-xx[i],
                  Yhb[i+1]-yy[i], zz[i]),))
            Xcb[i+1] = Coord[0][1][0]
            Ycb[i+1] = Coord[0][1][1]
            Zcb[i+1] = Coord[0][1][2]

            a.DatumCsysByThreePoints(name='Datum csys-2',
                  coordSysType=CARTESIAN, origin=(Xhb[i]+xx[i],
                  Yhb[i]+yy[i], arch_width[jj]-zz[i]),
                  point1=(Xcb[i+1], Ycb[i+1], Zcb[i+1]),
                   point2=(Xhb[i]+xx[i], Yhb[i]+yy[i], zz[i]))
            csysdatum=mdb.models['ARCH
                  BRIDGE'].rootAssembly.datums.keys()
            datum11 = mdb.models['ARCH BRIDGE'].rootAssembly.datums[
                  csysdatum[len(csysdatum)-1]]

            csa = a.SectionAssignment(sectionName='Pin-fixed',
```

```
                region=region)
            a.ConnectorOrientation(angle1=-90, axis1=AXIS_3,
                region=csa.getSet(), localCsys1=datum11)

            a.WirePolyLine(points=((v1.findAt(coordinates=(Xdp2[i],
                Ydp2[i], 0.0)), v2.findAt(coordinates=(Xcb[i+1],
                Ycb[i+1], Zcb[i+1])))), ), mergeWire=OFF,
                meshable=OFF)

            edges2 = e1.findAt(((Xcb[i+1], Ycb[i+1], Zcb[i+1]), ))
            region=regionToolset.Region(edges=edges2)
            a.SectionAssignment(region=region, sectionName='Pin')

#-----------------------------------------------------------------------
# BOUNDARY CONDITIONS
#-----------------------------------------------------------------------
a = mdb.models['ARCH BRIDGE'].rootAssembly

#BC-1 - ARCH @ x = 0
v1 = a.instances['ARCH-1'].vertices
verts1 = v1.findAt(((Xs[0], Ys[0], Zs[0]), ))
v2 = a.instances['ARCH-1-lin-1-2'].vertices
verts2 = v2.findAt(((Xs[0], Ys[0], arch_width[jj]), ))
region = regionToolset.Region(vertices=verts1+verts2)
mdb.models['ARCH BRIDGE'].DisplacementBC(name='Arch-1',
      createStepName='Initial', region=region, u1=SET, u2=SET,
      u3=SET, ur1=SET, ur2=SET, ur3=SET, amplitude=UNSET,
      distributionType=UNIFORM, fieldName='', localCsys=None)

#BC-2 - ARCH @ x = SPAN LENGTH
v1 = a.instances['ARCH-1-lin-1-2'].vertices
verts1 = v1.findAt(((Xs[n_el], Ys[n_el], arch_width[jj]), ))
v2 = a.instances['ARCH-1'].vertices
verts2 = v2.findAt(((Xs[n_el], Ys[n_el], Zs[n_el]), ))
region = regionToolset.Region(vertices=verts1+verts2)
mdb.models['ARCH BRIDGE'].DisplacementBC(name='Arch-2',
      createStepName='Initial', region=region, u1=SET, u2=SET,
      u3=SET, ur1=SET, ur2=SET, ur3=SET, amplitude=UNSET,
      distributionType=UNIFORM, fieldName='', localCsys=None)

#BC-3 - BRIDGE DECK @ x = 0
e1 = a.instances['Bridge Deck-1'].edges
edges1 = e1.findAt(((0.0, deck_height, arch_width[jj]/2), ),)
region = regionToolset.Region(edges=edges1)
mdb.models['ARCH BRIDGE'].DisplacementBC(name='Deck-1',
      createStepName='Initial', region=region, u1=SET, u2=SET,
      u3=SET, ur1=UNSET, ur2=UNSET, ur3=UNSET, amplitude=UNSET,
      distributionType=UNIFORM, fieldName='', localCsys=None)

#BC-3 - BRIDGE DECK @ x = SPAN LENGTH
e1 = a.instances['Bridge Deck-1'].edges
edges1 = e1.findAt(((sp_len[ii], deck_height, arch_width[jj]/2), ),)
region = regionToolset.Region(edges=edges1)
mdb.models['ARCH BRIDGE'].DisplacementBC(name='Deck-2',
      createStepName='Initial', region=region, u1=UNSET, u2=SET,
      u3=SET, ur1=UNSET, ur2=UNSET, ur3=UNSET, amplitude=UNSET,
      distributionType=UNIFORM, fieldName='', localCsys=None)
```

```
#<><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><>
# 6. MESH
#<><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><>
a = mdb.models['ARCH BRIDGE'].rootAssembly
elemType1 = mesh.ElemType(elemCode=S4R, elemLibrary=STANDARD,
      secondOrderAccuracy=OFF, hourglassControl=DEFAULT)
elemType2 = mesh.ElemType(elemCode=S3, elemLibrary=STANDARD)
elemType3 = mesh.ElemType(elemCode=B31, elemLibrary=STANDARD)
elemType4 = mesh.ElemType(elemCode=T3D2, elemLibrary=STANDARD)

# ASSIGNS ELEMENT TYPE TO BRIDGE DECK
f1 = a.instances['Bridge Deck-1'].faces
pickedRegions =(f1, )
a.setElementType(regions=pickedRegions, elemTypes=(elemType1,
      elemType2))
partInstances =(a.instances['Bridge Deck-1'], )
a.seedPartInstance(regions=partInstances, size=0.5,
      deviationFactor=0.1, minSizeFactor=0.1)
a.generateMesh(regions=partInstances)

# ASSIGNS ELEMENT TYPE TO ARCH
e1 = a.instances['ARCH-1-lin-1-2'].edges
e2 = a.instances['ARCH-1'].edges
pickedRegions =(e1, )
a.setElementType(regions=pickedRegions, elemTypes=(elemType3, ))
pickedRegions =(e2, )
a.setElementType(regions=pickedRegions, elemTypes=(elemType3, ))
partInstances =(a.instances['ARCH-1-lin-1-2'], )
a.seedPartInstance(regions=partInstances, size=0.5,
      deviationFactor=0.1, minSizeFactor=0.1)
a.generateMesh(regions=partInstances)
partInstances =(a.instances['ARCH-1'], )
a.seedPartInstance(regions=partInstances, size=0.5,
      deviationFactor=0.1, minSizeFactor=0.1)
a.generateMesh(regions=partInstances)

# ASSIGNS ELEMENT TYPE TO HANGERS
if n_hang[kk] > 0:
      e1 = a.instances['Hanger-1-lin-1-2'].edges
      e2 = a.instances['Hanger-1'].edges
      pickedRegions =(e1, )
      a.setElementType(regions=pickedRegions,elemTypes=(elemType3, ))
      pickedRegions =(e2, )
      a.setElementType(regions=pickedRegions,elemTypes=(elemType3, ))
      partInstances =(a.instances['Hanger-1-lin-1-2'], )
      a.seedPartInstance(regions=partInstances, size=0.5,
            deviationFactor=0.1, minSizeFactor=0.1)
      a.generateMesh(regions=partInstances)
      partInstances =(a.instances['Hanger-1'], )
      a.seedPartInstance(regions=partInstances, size=0.5,
            deviationFactor=0.1, minSizeFactor=0.1)
      a.generateMesh(regions=partInstances)

# ASSIGNS ELEMENT TYPE TO HORISONTAL HANGERS
if n_Hbrac[rr] > 0:
      for i in range(n_Hbrac[rr]):
            e1 = a.instances['Hbracing-'+str(i+1)].edges
            pickedRegions =(e1, )
```

```
            a.setElementType(regions=pickedRegions,
                  elemTypes=(elemType3, ))
            partInstances =(a.instances['Hbracing-'+str(i+1)], )
            a.seedPartInstance(regions=partInstances, size=0.5,
                  deviationFactor=0.1, minSizeFactor=0.1)
            a.generateMesh(regions=partInstances)

# ASSIGNS ELEMENT TYPE TO CROSS BRACING
if use_Cbrac[ss]==1:
      for i in range(n_fack):
            e1 = a.instances['Cbracing-'+str(i+1)+'-1'].edges
            pickedRegions =(e1, )
            a.setElementType(regions=pickedRegions,
                  elemTypes=(elemType3, ))
            e2 = a.instances['Cbracing-'+str(i+1)+'-2'].edges
            pickedRegions =(e2, )
            a.setElementType(regions=pickedRegions,
                  elemTypes=(elemType3, ))
            partInstances =(a.instances['Cbracing-'+str(i+1)+'-1'], )
            a.seedPartInstance(regions=partInstances, size=0.5,
                  deviationFactor=0.1, minSizeFactor=0.1)
            a.generateMesh(regions=partInstances)
            partInstances =(a.instances['Cbracing-'+str(i+1)+'-2'], )
            a.seedPartInstance(regions=partInstances, size=0.5,
                  deviationFactor=0.1, minSizeFactor=0.1)
            a.generateMesh(regions=partInstances)

# CREATE NODE SET FOR THE TOP NODE
a = mdb.models['ARCH BRIDGE'].rootAssembly
n1 = a.instances['ARCH-1'].vertices
x_coord = range(len(n1))
y_coord = range(len(n1))
z_coord = range(len(n1))
for i in range(len(n1)):
      x_coord[i] = n1[i].pointOn[0][0]
      y_coord[i] = n1[i].pointOn[0][1]
      z_coord[i] = n1[i].pointOn[0][2]
y_node = max(y_coord)
index= (y_coord.index(y_node))
v1 = a.instances['ARCH-1'].vertices
verts1 = v1.findAt(((x_coord[index], y_coord[index],
      z_coord[index]), ))
a.Set(vertices=verts1, name='topnode')
# CREATE NODE SET FOR 1ST QUARTER NODE
index2 = int(ceil(len(x_coord)/4))
v1 = a.instances['ARCH-1'].vertices
verts2 = v1.findAt(((x_coord[index2], y_coord[index2],
      z_coord[index2]), ))
a.Set(vertices=verts2, name='q1node')
# CREATE NODE SET FOR 2ND QUARTER NODE
index3 = int(ceil(len(x_coord)*3/4))
v1 = a.instances['ARCH-1'].vertices
verts3 = v1.findAt(((x_coord[index3], y_coord[index3],
      z_coord[index3]), ))
a.Set(vertices=verts3, name='q2node')
# CREATE NODE SET FOR ALL THREE NODES
verts4 = verts1+verts2+verts3
a.Set(vertices=verts4, name='all3nodes')
```

```
#<><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><>
# 7. STEPS
#<><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><>

#--------------------------------------------------------------------
# CREATES FREQUENCY STEP
#--------------------------------------------------------------------
mdb.models['ARCH BRIDGE'].FrequencyStep(name='Frequency',
      previous='Initial',minEigen=0.1, maxEigen=10.0,
      normalization=MASS)
#--------------------------------------------------------------------
# CREATES RANDOM RESPONSE STEP
#--------------------------------------------------------------------
mdb.models['ARCH BRIDGE'].PsdDefinition(name='rekt',
      data=((1.0, 0.0, 0.1), (1.0, 0.0, 10.0)))
mdb.models['ARCH BRIDGE'].RandomResponseStep(name='RR',
      previous='Frequency', freq=((0.1, 10.0, 10, 3), ),
      directDamping=None, compositeDamping=None,
rayleighDamping=None,
      structuralDamping=None, directDampingByFrequency=((0.1,
            damping), (10.0,damping)))
regionDef=mdb.models['ARCH BRIDGE'].rootAssembly.sets['all3nodes']
mdb.models['ARCH BRIDGE'].fieldOutputRequests['F-Output-
2'].setValues(
      variables=('U', 'RU'), region=regionDef, sectionPoints=DEFAULT,
    rebar=EXCLUDE)
```

BRIDGE_DATA.py

```
#----------------------------------------------------------------------
# BRIDGE_DATA.PY
# By: Jonathan Johansson, Daniel Josefsson
#----------------------------------------------------------------------
# INPUT FILE - SHALL BE PLACED IN THE SAME DIRECTORY AS MAIN SCRIPT

#----------------------------------------------------------------------
# GEOMETRY INPUT
#----------------------------------------------------------------------

# BRIDGE GEOMETRY
sp_len=[]                #Span length [m]
arch_height=[]           #Arch height [m]
deck_height=             #Bridge deck height above the spring line
t_deck=[]                #Thickness of deck [m]
arch_width=[]            #Width between the two arches [m]
opt_el_len=              #[m]
free_height=             #Free height [m] Value according to VGU

                         #Type of section for arch
                         # 1 = I-profile, 2 = Box, 3 = Pipe,
                         # 4 = Circular, 5 = Rectangular,
type=                    # 6 = Hexagonal, 7 = Trapezoidal
                         # 8 = L-profile, # 9 = T-profile


#GEOMETRY HANGERS
n_hang=[]                #Number of hangers [-]
phi_hang=[]              #Diameter for hanger

#GEOMETRY HORIZONTAL BRACING
n_Hbrac=[]               #Number of horizontal bracing beams

                         #Type of section for horisontal bracing
                         # 1 = I-profile, 2 = Box, 3 = Pipe,
                         # 4 = Circular, 5 = Rectangular,
type=                    # 6 = Hexagonal, 7 = Trapezoidal
                         # 8 = L-profile, # 9 = T-profile


#GEOMETRY CROSS-BRACING
use_Cbrac=[]             #Set 1 if cross bracing shall be used,
                               #otherwise set value 0
phi_Cbrac=[]             #Diameter for horizontal bracing

#----------------------------------------------------------------------
# MATERIAL INPUT
#----------------------------------------------------------------------
damping=                 # Damping coefficient [Timber = 0.01]

#DEFINE MATERIAL FOR ARCH
E1_A =                                #[Pa]
E2_A =                                #[Pa]
E3_A =                                #[Pa]
v12_A =                               #Poisson ratio[-]
v13_A =                               #Poisson ratio[-]
v23_A =                               #Poisson ratio[-]
```

```
G12_A =                                 #[Pa]

G13_A =                                 #[Pa]
G23_A =                                 #[Pa]
D_arch=                                 #[kg/m^3]



#DEFINE MATERIAL FOR DECK
E1_D =                                  #[Pa]
E2_D =                                  #[Pa]
E3_D =                                  #[Pa]
v12_D =                                 #Poisson ratio[-]
v13_D =                                 #Poisson ratio[-]
v23_D =                                 #Poisson ratio[-]
G12_D =                                 #[Pa]

G13_D =                                 #[Pa]
G23_D =                                 #[Pa]
D_deck=                                 #[kg/m^3]

#DEFINE MATERIAL FOR HANGERS
E_hang =                                #[Pa]
v_hang =                                #Poisson ratio[-]
D_hang=                                       #[kg/m^3]

#DEFINE MATERIAL FOR HORISONTAL BRACING
E1_Hb =                                 #[Pa]
E2_Hb =                                 #[Pa]
E3_Hb =                                 #[Pa]
v12_Hb =                                #Poisson ratio[-]
v13_Hb =                                #Poisson ratio[-]
v23_Hb =                                #Poisson ratio[-]
G12_Hb =                                #[Pa]

G13_Hb =                                #[Pa]
G23_Hb =                                #[Pa]
D_Hbrac=                                #[kg/m^3]

# DEFINE MATERIAL FOR CROSS-BRACING
E_Cbrac =                               #[Pa]
v_Cbrac  =                              #Poisson ratio[-]
D_Cbrac =                               #[kg/m^3]

#-------------------------------------------------------------------------
# SECTION TYPES FOR ARCH
#-------------------------------------------------------------------------

#--# I
I_t1=[]
I_t2=[]
I_t3=[]
I_b1=[]
I_b2=[]
I_h=[]
I_l=[]
#--# Box
B_b=[]
```

```
B_a=[]
B_t1=[]
B_t2=[]
B_t3=[]
B_t4=[]

#--# Pipe
P_r=[]
P_t=[]

#--# Circular
C_r=[]

#--# Rectangular
R_a=[]
R_b=[]

#--# Hexagonal
H_r=[]
H_t=[]

#--# Trapezoidal
T_a=[]
T_b=[]

#--# L
L_a=[]
L_b=[]
L_t1=[]
L_t2=[]

#--# T
T_b=[]
T_h=[]
T_l=[]
T_tf=[]
T_tw=[]

#-----------------------------------------------------------------------
# SECTION TYPES FOR HORISONTAL BRACING
#-----------------------------------------------------------------------
#--# I
I_t1hb=[]
I_t2hb=[]
I_t3hb=[]
I_b1hb=[]
I_b2hb=[]
I_hhb=[]
I_lhb=[]

#--# Box
B_bhb=[]
B_ahb=[]
B_t1hb=[]
B_t2hb=[]
B_t3hb=[]
B_t4hb=[]
```

```
#--# Pipe
P_rhb=[]
P_thb=[]

#--# Circular
C_rhb=[]

#--# Rectangular
R_ahb=[]
R_bhb=[]
#--# Hexagonal
H_rhb=[]
H_thb=[]

#--# Trapezoidal
T_ahb=[]
T_bhb=[]

#--# L
L_ahb=[]
L_bhb=[]
L_t1hb=[]
L_t2hb=[]

#--# T
T_bhb=[]
T_hhb=[]
T_lhb=[]
T_tfhb=[]
T_twhb=[]
```

OUTPUT.py

```
#-------------------------------------------------------------------
# OUTPUT.PY
# By: Jonathan Johansson, Daniel Josefsson
#-------------------------------------------------------------------
# SCRITP THAT EXTRACTS DISPLACEMENTS AND EIGENFREQUENCIES FOR THE
# ARCH BRIDGE ANALYSIS CREATED BY THE SCRITP ARCH_BRIDGE.py
#
# BRIDGE_DATA.py MUST BE PLACED IN THE SAME FOLDER AS OUTPUT.py

#-------------------------------------------------------------------
# 1. IMPORTS PROGRAMMING COMMANDS FROM ABAQUS
#-------------------------------------------------------------------
from abaqus import *
from abaqusConstants import *
import __main__
import odb
import math
import odbAccess
import odbSection
import odbMaterial
import section
import regionToolset
import displayGroupMdbToolset as dgm
import step
import part
import material
import assembly
import interaction
import load
import mesh
import optimization
import job
import sketch
import visualization
import xyPlot
import displayGroupOdbToolset as dgo
import connectorBehavior
import time
session.journalOptions.setValues(replayGeometry=COORDINATE)
#-------------------------------------------------------------------
# 2. RUN INDATA FILE
#-------------------------------------------------------------------
execfile('BRIDGE_DATA.py')
#-------------------------------------------------------------------
# 3. NUMBER OF PARAMETERS VARYING AND  LOOP OVER ALL PARAMETERS
#-------------------------------------------------------------------
a = range(len(sp_len))
b = range(len(arch_width))
c = range(len(n_hang))
d = range(len(R_a))
e = range(len(R_b))
f = range(len(arch_height))
g = range(len(t_deck))
h = range(len(phi_hang))
k = range(len(n_Hbrac))
l = range(len(use_Cbrac))
m = range(len(phi_Cbrac))
```

```
n = range(len(R_ahb))
o = range(len(R_bhb))

name=range(len(a)*len(b)*len(c)*len(d)*len(e)*len(f)*len(g)*len(h)*
len(k)*len(l)*len(m)*len(n)*len(o))

i=0
for ii in range(len(sp_len)):
    for jj in range(len(arch_width)):
      for kk in range(len(n_hang)):
          for nn in range(len(arch_height)):
             for ll in range(len(R_a)):
                 for mm in range(len(R_b)):
                    for pp in range(len(t_deck)):
                       for qq in range(len(phi_hang)):
                          for rr in range(len(n_Hbrac)):
                             for ss in range(len(use_Cbrac)):
                                for tt in range(len(phi_Cbrac)):
                                   for uu in range(len(R_ahb)):
                                      for vv in range(len(R_bhb)):
      name[i]=str(int(sp_len[ii]))+''+str(int(arch_width[jj]))+'-
      '+str(int(arch_height[nn]))+'-'+str(int(n_hang[kk]))+'-
      '+str(int(phi_hang[qq]*1000))+'-'+str(int(n_Hbrac[rr]))+'-
      '+str(int(t_deck[pp]*1000))+'-'+str(int(R_a[ll]*1000))+'-
      '+str(int(R_b[mm]*1000))+'-'+str(int(R_ahb[uu]*1000))+'-
      '+str(int(R_bhb[vv]*1000))+'-'+str(int(use_Cbrac[ss]))+'-
      '+str(int(phi_Cbrac[tt]*1000))
      i=i+1


for j in range(len(name)):
      # -----------------------------------------------------------------
      # 3.1 OPEN .ODB FILE
      # -----------------------------------------------------------------
      odb = session.openOdb('Input\\'+name[j]+'.odb')
      assembly = odb.rootAssembly
      # -----------------------------------------------------------------
      # 3.2 Obtaining the displacements
      # -----------------------------------------------------------------
      session.viewports['Viewport: 1'].setValues(displayedObject=odb)
      session.odbData['Input\\'+name[j]+'.odb'].setValues(activeFrame
            s=(('Frequency', ('0:-1', )), ))
      U = session.xyDataListFromField(odb=odb, outputPosition=NODAL,
            variable=(('U', NODAL, ((COMPONENT, 'U3'), )), ),
            nodeSets=('Q1NODE', 'Q2NODE', 'TOPNODE', ))
      # -----------------------------------------------------------------
      # 3.3 Obtaining the RMS-displacements
      # -----------------------------------------------------------------
      session.viewports['Viewport: 1'].setValues(displayedObject=odb)
      session.odbData['Input\\'+name[j]+'.odb'].setValues(
            activeFrames=(('RR', ('0:-1', )), ))
      RU = session.xyDataListFromField(odb=odb, outputPosition=NODAL,
            variable=(('RU',NODAL,((COMPONENT, 'RU3'), )), ),
            nodeSets=('Q1NODE', 'Q2NODE','TOPNODE', ))
      # -----------------------------------------------------------------
      # 3.4 Obtaining the eigenfrequencies
      # -----------------------------------------------------------------
      freq_bulk = odb.steps['Frequency'].historyRegions[
```

```
            'Assembly ASSEMBLY'].historyOutputs['EIGFREQ']
      freq = range(len(freq_bulk.data))
      for i in range(len(freq_bulk.data)):
            freq[i] = freq_bulk.data[i][1]
      # ------------------------------------------------------------
      #  3.5 Writing the obtained data to an output file
      # ------------------------------------------------------------
      outputFile = open('Output\\'+name[j]+'_RMS.txt','w')
      for i in range(len(RU[0])):
            outputFile.write('%10.20E\t%10.20E\t%10.20E\t%10.20E\n' %
            (RU[0][i][0],RU[0][i][1],RU[1][i][1],RU[2][i][1]))
      outputFile.close()

      outputFile = open('Output\\'+name[j]+'_U.txt','w')
      for i in range(len(U[0])):
            outputFile.write('%10.20E\t%10.20E\t%10.20E\t%10.20E\n' %
            (freq[i],U[0][i][1],U[1][i][1],U[2][i][1]))
      outputFile.close()
      # ------------------------------------------------------------
      # 3.6 Delete created xy-data
      # ------------------------------------------------------------
      n_data = len(session.xyDataObjects)
      xyname = session.xyDataObjects.keys()
      for i in range(n_data):
            del session.xyDataObjects[xyname[i]]
      # ------------------------------------------------------------
      # 3.7 Closing .odb file
      # ------------------------------------------------------------
      odb.close()
#------------------------------------------------------------------
# 4. WRITE A .TXT FILE WITH THE FILENAMES FROM THE ANALYSIS
#------------------------------------------------------------------
outputFile = open('Output\\Filename.txt','w')
for j in range(len(name)):
      outputFile.write(name[j].replace('-','\t')+'\n')
outputFile.close()
```