

# Identification of structural properties of high-rise buildings

Application of a model updating technique for estimating the structural properties of high-rise buildings.

Master thesis  
Isa Sophie Rittfeld



# Identification of structural properties of high-rise buildings

Application of a model updating technique for estimating the structural properties of high-rise buildings.

by

Isa Sophie Ritfeld

Student Name	I.S. Ritfeld
Student nummer	4892534

Supervisors of TU Delft:	Dr.ir. K.N. van Dalen and Dr. E. Lourens
Supervisors of TNO:	Ir. A.J. Bronkhorst and Ir .D. Moretti
Supervisor of Aronsohn:	Ir. P. Lagendijk
Project Duration:	October, 2023 - July, 2024
Study, Specialisation:	Structural Engineering, Structural Mechanics
Faculty:	TU Delft Faculty Of Civil Engineering and Geo sciences

Cover:	Picture modified by Isa Ritfeld
Style:	TU Delft Report Style, with modifications by Isa Ritfeld

# Preface

This research is conducted to fulfill the requirements for the Master degree in Structural Engineering, with a specialization in Structural Mechanics, at the Technical University of Delft. The research is carried out from October 18th to August 23th, in cooperation with TNO, a Dutch applied scientific research organization, and Aronsohn, an engineering firm specializing in building design. It involves two high-rise buildings, the New Orleans and the Delftse Poort, both located in Rotterdam.

The research combines two of my greatest interests. The first interest is high-rise buildings, which I developed during my work experience at Aronsohn. There, I had the opportunity to work on building projects as a junior structural engineer. The complex modeling and calculations fascinated me. My other interest involves construction mechanics. Previously, I worked as a Teacher Assistant (TA) for the mechanics courses in the Bachelor's program of Civil Engineering. While the research focuses on the dynamics of buildings, a mechanical understanding, such as grasping the behavior of Finite Element (FE) models, is required to interpret the outcomes of this research. Moreover, as I had limited chances during my Master to see dynamics applied on such a scale. Therefore, this research provided a unique opportunity to explore the practical application of dynamics.

My experience with the research was highly positive. Through this research, I gained substantial knowledge, ranging from theoretical concepts to practical applications. I had the freedom to explore my own curiosities and discuss my findings with five great experts in the field. Communication is crucial, not only due to the involvement of multiple parties but also to enhance the research quality. Therefore, I would like to thank the supervisors Karel van Dalen, Eliz-Mari Lourens, Paul Lagendijk, Okke Bronkhorst, and Davide Moretti for guiding this thesis and allowing me to grow along the way. Moreover, I extend my gratitude to TNO and Aronsohn for providing me with weekly guidance, a workspace to conduct this research, and the necessary equipment. Lastly, I want to thank my family and friends for their interest and unwavering support during this time. I hope the reader will be as enthusiastic about reading this research as I was during the process.

Isa Sophie Ritfeld  
Delft, August 2024



# Summary

This research focused on applying vibration-based model updating to estimate the structural properties of high-rise buildings using a discrete Timoshenko beam model. Previous studies by Moretti et al. [1] and Taciroglu et al. [2] showed limitations in estimating the structural properties of buildings using model updating with a uniform Euler-Bernoulli beam model or a uniform Timoshenko beam model. Both models were not able to describe the third measured bending mode accurately after updating. Moreover, with the uniform Euler-Bernoulli beam model, when only the lowest two bending modes were used to fit the model, the estimates of the rotational stiffness of the foundation around the y-axis,  $K_{r,y}$ , showed significant uncertainty after model updating, with a Coefficient of Variation (CV) of 46.9%. Therefore, this research aimed to improve the accuracy of structural property estimations for high-rise buildings by using a more detailed model.

The model updating method applied in this research was an indirect vibration-based technique, which adjusted the input parameters of the chosen model to minimize the difference between the model output and the measured data. Both the model output and the measured data were in the shape of natural frequencies and mode shapes. The technique was applied to two high-rise buildings: the residential tower New Orleans, which is bending-dominant in behavior, and the office tower Delftse Poort, which is shear-dominant in behavior and exhibits irregular stiffness across its height. The discrete Timoshenko beam models used to approximate the dynamic behavior of these buildings were created using the Finite Element (FE) models of the buildings.

The research findings highlighted several key aspects essential for obtaining more accurate estimations of the structural properties of high-rise buildings. For a more accurate estimation of parameter  $K_{r,y}$ , it is of importance to incorporate shear deformations in the model and to account for irregular stiffness along the height. For the New Orleans, using the discrete Timoshenko beam model led to estimates of parameter  $K_{r,y}$  with low uncertainty, indicated by a Coefficient of Variation of 6.91%. This was an improvement compared to the study by Moretti et al. [1], which showed a Coefficient of Variation of 46.9% using the uniform Euler-Bernoulli model.

Moreover, obtaining accurate values for the bending stiffness was crucial for achieving more precise estimations of the structural properties. It was challenging to determine the bending stiffness values using the FE models. These challenges posed a problem for model updating, as the initial structural property ratios were maintained for both high-rise buildings. Maintaining these ratios gives weight to the initial structural property values. These values must be correct. Otherwise, model updating is limited by the incorrect ratios and will not be able to accurately match the measured modal properties.

For the Delftse Poort, more pure bending modes were needed for better accuracy. The discrete Timoshenko beam model was unable to match the measured second bending mode in the y-direction, as it exhibited twisting, which the model could not represent due to its limitation to pure bending modes. Furthermore, since the discrete Timoshenko beam model requires a single displacement value per height but multiple sensors were used to measure displacements, the measurements had to be averaged. This averaging process may lead to a mode shape that deviates from the actual behavior, resulting in an inaccurate representation of reality.



# Contents

<b>Preface</b>	<b>i</b>
<b>Summary</b>	<b>ii</b>
<b>Nomenclature</b>	<b>xiii</b>
<b>1 Introduction</b>	<b>1</b>
1.1 Research problem statement . . . . .	1
1.2 Research objectives . . . . .	2
1.3 Research questions . . . . .	2
1.4 Significance research . . . . .	2
1.5 Scope . . . . .	3
1.6 Limitations . . . . .	3
1.7 Organisation research . . . . .	3
<b>2 Theoretical background</b>	<b>4</b>
2.1 Background . . . . .	4
2.2 Key concepts and theories . . . . .	5
2.2.1 Parameters . . . . .	5
2.2.2 Solution space . . . . .	5
2.2.3 Parametrization . . . . .	6
2.2.4 Constraints . . . . .	6
2.2.5 Regularization . . . . .	6
2.2.6 Optimization Algorithm . . . . .	7
2.2.7 Exploration and exploitation . . . . .	7
2.2.8 Objective function . . . . .	7
2.3 Optimization algorithms . . . . .	7
2.3.1 Individual-based algorithms . . . . .	7
2.3.2 Population-based algorithms . . . . .	8
2.4 Methodologies . . . . .	9
2.4.1 The direct methods using modal data . . . . .	9
2.4.2 Indirect methods using modal data . . . . .	10
2.4.3 Methods using Frequency Domain Data (FDD) . . . . .	10
2.5 Conclusion . . . . .	11
<b>3 Literature review</b>	<b>13</b>
3.1 The residential tower New Orleans in Rotterdam . . . . .	13
3.2 The Millikan Library building in Pasadena . . . . .	16
3.3 Conclusion . . . . .	19
<b>4 Methodology</b>	<b>20</b>
4.1 The model updating method . . . . .	20
4.2 The discrete Timoshenko beam model . . . . .	20
<b>5 The residential tower New Orleans</b>	<b>22</b>
5.1 Studies . . . . .	22
5.1.1 Influence Features Model . . . . .	22
5.1.2 Influence Parameters . . . . .	26
5.1.3 Influence Optimization Algorithm . . . . .	26
5.1.4 Influence Amount of Measured Modal Properties . . . . .	27
5.1.5 Influence Measurements Uncertainties . . . . .	27
5.1.6 Estimating Structural Properties . . . . .	28

5.2	Results . . . . .	29
5.2.1	Influence Features Model . . . . .	29
5.2.2	Influence Parameters . . . . .	31
5.2.3	Optimization Algorithm . . . . .	33
5.2.4	Influence Amount of Measured Modal Properties . . . . .	35
5.2.5	Measurement Uncertainties . . . . .	37
5.2.6	Estimating Structural Properties . . . . .	38
5.3	Main Findings . . . . .	40
<b>6</b>	<b>The office tower the Delftse Poort</b>	<b>41</b>
6.1	Studies . . . . .	41
6.1.1	Influence Features Model . . . . .	41
6.1.2	Influence Parameters . . . . .	44
6.1.3	Influence Optimization Algorithm . . . . .	45
6.1.4	Influence Amount of Measured Modal Properties . . . . .	45
6.1.5	Influence Measurements Uncertainties . . . . .	45
6.1.6	Estimating Structural Properties . . . . .	46
6.2	Results . . . . .	47
6.2.1	Influence Model . . . . .	47
6.2.2	Influence Parameters . . . . .	49
6.2.3	Influence Optimization Algorithm . . . . .	51
6.2.4	Influence Amount of Measured Modal Properties . . . . .	53
6.2.5	Measurement Uncertainties . . . . .	55
6.2.6	Estimating Structural Properties . . . . .	55
6.3	Main Findings . . . . .	57
	<b>Discussion</b>	<b>60</b>
	<b>Conclusion</b>	<b>62</b>
	<b>References</b>	<b>63</b>
<b>A</b>	<b>Timoshenko beam model</b>	<b>65</b>
A.1	The governing equations of dynamics . . . . .	66
A.2	Eigenvalue problem . . . . .	67
A.3	Eigenvalue analysis . . . . .	67
A.4	Boundary and continuity conditions . . . . .	69
A.5	The eigenvalues and the eigenmodes . . . . .	70
A.6	Numerical example . . . . .	71
A.7	Conclusion . . . . .	76
<b>B</b>	<b>Verification Timoshenko beam model</b>	<b>78</b>
B.1	Studies . . . . .	78
B.1.1	Verification code model . . . . .	78
B.1.2	Verification behavior model . . . . .	78
B.2	Results . . . . .	79
B.2.1	Verification code model . . . . .	79
B.2.2	Verification behavior model . . . . .	80
B.3	Conclusion . . . . .	80
<b>C</b>	<b>Structural Properties</b>	<b>81</b>
C.1	Function description . . . . .	81
C.2	The residential tower New Orleans . . . . .	82
C.2.1	Segment 0 . . . . .	83
C.2.2	Segment I . . . . .	84
C.2.3	Segment II . . . . .	86
C.2.4	Segment III . . . . .	87
C.2.5	Segment IV . . . . .	89
C.2.6	Bending stiffness . . . . .	90

---

C.2.7	Structural properties . . . . .	91
C.3	The office tower the Delfste Poort . . . . .	92
C.3.1	Segment 0 and I . . . . .	93
C.3.2	Segment II . . . . .	96
C.3.3	Segment III . . . . .	97
C.3.4	Bending stiffness . . . . .	98
C.3.5	Structural properties . . . . .	98
C.4	Conclusion . . . . .	99
<b>D</b>	<b>Measured modal properties</b>	<b>102</b>
D.1	Measurement setup the New Orleans . . . . .	102
D.2	Measurement setup the Delfste Poort . . . . .	103
D.3	Data analysis . . . . .	104
D.4	Results data analysis . . . . .	105
D.5	Conclusion . . . . .	108
<b>E</b>	<b>Influence of parameters</b>	<b>109</b>
<b>F</b>	<b>Python</b>	<b>111</b>



# List of Figures

2.1	A typical two-dimensional finite element mesh [3]. . . . .	4
2.2	The solution space (a) of the variables $x$ and $y$ with several constraints. The top view is indicated with (b). The axis $f(x, y)$ indicates the fitting of the solutions. The constraints are represented with white shading within the solution space, indicating infeasible solutions [10]. . . . .	6
2.3	The application of the barrier penalty function on the solution space (a). The top view is indicated with (b). The axis $f(x, y)$ indicates the fitting of the solutions. . . . .	7
2.4	The processes of nature mimicked with evolutionary-based algorithms: (a) illustrates the crossover process, used to exploit the search. (b) Depicts the mutation process, used to explore the solution space. . . . .	8
3.1	A picture of the residential tower New Orleans. . . . .	13
3.2	The uniform Euler-Bernoulli beam model (2D) used to approximate the dynamic behavior of the high-rise building. . . . .	14
3.3	(a) Picture and (b, c, d and e) drawings indicating the acceleration sensors positions (green squares) on the 15th floor, the 34th floor, and the 44th floor. The green arrows indicate the directions of the acceleration sensors [1]. . . . .	15
3.4	A picture (a) and a schematic (b) of the Millikan Library building [2]. . . . .	16
3.5	The uniform Timoshenko beam model (1D) use to approximate the dynamic behavior of the high-rise building. . . . .	17
4.1	A schematic representation of (a) a high-rise building and (b) the discrete Timoshenko beam model used to approximate the dynamic behavior of that building. . . . .	21
5.1	Pictures of (a) the FE model of the New Orleans, (b) the 5 segments defined for the New Orleans model, and (c) the five-segment Timoshenko beam model. . . . .	23
5.2	The floor plan of segment I. . . . .	24
5.3	The floor plan of segment III. . . . .	25
5.4	Pictures of (a) the uniform Timoshenko beam model (b) the discrete Euler-Bernoulli beam model. . . . .	26
5.5	Pictures (a,b) of parameter $kG$ scaled by a factor of 10. . . . .	27
5.6	A visualization of the random errors applied on the first natural frequency and mode shape. 28	
5.7	An schematic (a) of the New Orleans and the positions of the acceleration sensors (green squares) on (b) the 44th floor, (c) the 34th floor, and (d) the 15th floor. The green arrows indicate the directions of the acceleration sensors [17, 12]. . . . .	29
5.8	The measured and model mode shapes in x-direction . . . . .	31
5.9	The measured and model mode shapes in y-direction . . . . .	31
5.10	The influence of parameter $K_r$ on the model output in y-direction. . . . .	32
5.11	The influence of parameter $K_t$ on the model output (a,b) in y-direction. . . . .	32
5.12	The influence of parameter $E$ on the model output (a,b) in y-direction. . . . .	33
5.13	The influence of parameter $I$ on the model output (a,b) in y-direction. . . . .	33
5.14	The influence of parameter $kG$ on the model output (a,b) in y-direction. . . . .	33
5.15	The influence of the different algorithms on the model updating result of parameter $K_r$ . 34	
5.16	The influence of the different algorithms on the model updating result of parameter $K_t$ . 35	
5.17	The influence of the different algorithms on the model updating result of the product $EI$ . 35	
5.18	The influence of the amount of modal information on the model updating result of parameter $K_r$ . . . . .	36
5.19	The influence of the amount of modal information on the model updating result of parameter $K_t$ . . . . .	36

5.20	The influence of the amount of modal information on the model updating result of $EI$ .	37
5.21	The influence of measurement uncertainties on the model updating results.	37
5.22	The estimated parameter values of Case 1 in x-direction.	39
5.23	The estimated parameter values of Case 1 in y-direction.	39
6.1	Pictures of (a) the FE model of the Delftse Poort, (b) the 4 segments defined for the Delftse Poort model, and (c) the four-segment Timoshenko beam model.	42
6.2	The floor plan of segment 0.	43
6.3	Pictures of (a) the uniform Timoshenko beam model and (b) the discrete Euler-Bernoulli beam model.	44
6.4	A visualization of the random errors applied on the first natural frequency and mode shape.	46
6.5	Pictures of (e) the Delftse Poort and the positions of the acceleration sensors (green squares) on (d) floor 1, on (c) the 19th floor, (b) the 30th floor and (a) the 40th floor. The arrows indicate the directions of the acceleration sensors.	47
6.6	The measured and model mode shapes.	48
6.7	The influence of parameter $K_r$ on the model output (a,b).	49
6.8	The influence of parameter $K_t$ on the model output (a,b).	50
6.9	The influence of parameter $E$ on the model output (a,b).	50
6.10	The influence of parameter $I$ on the model output (a,b).	50
6.11	The influence of parameter $kG$ on the model output (a,b).	51
6.12	The influence of the different algorithms on the model updating result of parameter $K_r$ .	52
6.13	The influence of the different algorithms on the model updating result of parameter $K_t$ .	52
6.14	The influence of the different algorithms on the model updating result of the product $EI$ .	53
6.15	The influence of the amount of modal information on the model updating result of parameter $K_r$ .	54
6.16	The influence of the amount of modal information on the model updating result of parameter $K_t$ .	54
6.17	The influence of the amount of modal information on the model updating result of $EI$ .	54
6.18	The influence of measurement uncertainties on the model updating results.	55
6.19	The estimated parameter values of Case 1 in x-direction.	56
6.20	The mode shape of the second bending mode in y-direction of the Delftse Poort.	59
A.1	The Timoshenko beam model (1D) with various boundary conditions: (a.) a clamped beam, (b.) a simply supported beam, (c.) a beam supported by a rotational spring and (d.) a beam supported by a translational spring.	65
A.2	The sign convention of the Timoshenko beam of the publication 'Wave motion in Elastic Solids' of Karl F. Graff [20]. This sign convention is used in this Appendix.	66
A.3	A illustration of the first five mode shapes : (a.) shows the results of $Y(x)$ obtained with the two-piece Timoshenko beam model and (b.) shows the results of the article [19]. The functions $V_1$ , $V_2$ , $V_3$ , $V_4$ and $V_5$ are defined in article [19].	71
A.4	A illustration of the first five mode shapes : (a.) shows the results of $\phi(x)$ obtained with the two-piece Timoshenko beam model and (b.) shows the results of the article [19]. The functions $\Phi_1$ , $\Phi_2$ , $\Phi_3$ , $\Phi_4$ and $\Phi_5$ are defined in article [19].	72
A.5	The plotted mode shape at the transition frequency: (a.) shows the result of $Y(x)$ obtained with the two-piece Timoshenko beam model and (b.) shows the result of the article [19]. A double eigenvalue can occur at the transition frequency (an example of that is shown in red by the article) [19]. The functions $V_{28}$ and $V_{28*}$ are defined in article [19].	73
A.6	The plotted mode shape at the transition frequency: (a.) shows the result of $\Psi(x)$ obtained with the two-piece Timoshenko beam model and (b.) shows the result of the article. A double eigenvalue can occur at the transition frequency (an example of that is shown in red by the article) [19]. The functions $\Phi_{28}$ and $\Phi_{28*}$ are defined in article [19].	73

A.7	The plotted mode shapes of modes 27, 28, 30 and 31: (a.) shows the results of $Y(x)$ obtained with the two-piece Timoshenko beam model and (b.) shows the results of the article [19]. The functions $V_{27}$ , $V_{28}$ , $V_{30}$ , $V_{31}$ are defined in article [19]. These mode shapes are part of the so-called second spectrum of modes, where the shape of the eigemode is identical to those appearing at the first part of the spectrum, however, they are associated with higher wave numbers. . . . .	75
A.8	The plotted mode shapes of modes 27, 28, 30 and 31: (a.) shows the results of $\Psi(x)$ obtained with the two-piece Timoshenko beam model and (b.) shows the results of the article [19]. The functions $\Phi_{27}$ , $\Phi_{28}$ , $\Phi_{30}$ , $\Phi_{31}$ are defined in article [19]. These mode shapes are part of the so-called second spectrum of modes, where the shape of the eigemode is identical to those appearing at the first part of the spectrum, however, they are associated with higher wave numbers. . . . .	75
A.9	The plotted mode shapes of modes 29 and 32: (a.) shows the results obtained with the two-piece Timoshenko beam model and (b.) shows the results of the article [19]. They are not part of the second spectrum of modes as these modes have no equivalent at lower frequencies. The functions $V_{29}$ , and $V_{32}$ are defined in article [19]. . . . .	76
A.10	The plotted mode shapes of modes 29 and 32: (a.) shows the results of $\Psi(x)$ obtained with the two-piece Timoshenko beam model and (b.) shows the results of the article [19]. They are not part of the second spectrum of modes as these modes have no equivalent at lower frequencies. The functions $\Phi_{29}$ , and $\Phi_{32}$ are defined in article [19]. . . . .	76
B.1	The three-piece Timoshenko beam model (a) and the three-piece Euler-Bernoulli beam model (c). . . . .	78
B.2	An illustration of the first three mode shapes: (a) shows the results of transverse displacement obtained with the three-piece Timoshenko beam model, and (b) shows the results of the transverse displacement obtained by Taciroglu et al. [2]. . . . .	79
B.3	The natural frequencies of the two models in question. The black dotted lines represent the natural frequencies of the Euler-Bernoulli beam model. The dots represent the natural frequencies of the Timoshenko beam model. . . . .	80
C.1	Pictures of (a) the FE model of the New Orleans, (b) the 5 segments defined for the New Orleans model, and (c) the five-segment Timoshenko beam model. . . . .	82
C.2	The floor plan of segment 0. . . . .	83
C.3	The floor plan of segment I. . . . .	85
C.4	The floor plan of segment II. . . . .	86
C.5	The floor plan of segment III. . . . .	88
C.6	The floor plan of segment IV. . . . .	90
C.7	Pictures of (a) the FE model of the Delftse Poort, (b) the 4 segments defined for the Delftse Poort model, and (c) the four-segment Timoshenko beam model. . . . .	92
C.8	The floorplan of segment 0 and I. . . . .	95
C.9	The floor plan of segment II. . . . .	100
C.10	The floor plan of segment III. . . . .	101
D.1	An picture of the residential tower New Orleans. . . . .	102
D.2	Pictures of (a) the New Orleans and the positions of the acceleration sensors (green squares) on (b) the 44th floor, (c) the 34th floor, and (d) the 15th floor. The green arrows indicate the directions of the acceleration sensors. . . . .	103
D.3	An picture of the office tower the Delfste Poort. . . . .	103
D.4	Pictures of (e) the Delfste Poort and the positions of the acceleration sensors (green squares) on (d) floor 1, on (c) the 19th floor, (b) the 30th floor and (a) the 40th floor. The arrows indicate the directions of the acceleration sensors. . . . .	104
D.5	The dimensionless spectrum of the first singular value. The green dots indicate the natural frequencies of the residential tower New Orleans [24]. . . . .	105
E.1	The influence of parameter $K_r$ on the model output (a,b) in x-direction. . . . .	109
E.2	The influence of parameter $K_t$ on the model output (a,b) in x-direction. . . . .	109
E.3	The influence of parameter $E$ on the model output (a,b) in x-direction. . . . .	110



---

E.4	The influence of parameter $I$ on the model output (a,b) in x-direction. . . . .	110
E.5	The influence of parameter $kG$ on the model output (a,b) in x-direction. . . . .	110

# List of Tables

2.1	The strengths and limits of Individual-based algorithms. . . . .	8
2.2	The strengths and limits of population-based algorithms. . . . .	9
2.3	An overview of the direct methods [5]. . . . .	9
2.4	The strengths and limits of the direct methods. . . . .	9
2.5	An overview of the iterative methods of the model updating technique [5]. . . . .	10
2.6	The strengths and limits of the indirect methods. . . . .	10
2.7	An overview of the iterative methods of the model update technique [5]. . . . .	11
2.8	The strengths and limits of the methods using FDD. . . . .	11
3.1	The design values assigned as initial input parameter values for the Euler-Bernoulli beam model [1]. . . . .	14
3.2	The identified pure bending modes in dominant x- or y- direction [1, 12]. . . . .	14
3.3	The modal properties of the residential tower New Orleans in terms of natural frequencies and MAC [1]. . . . .	15
3.4	The structural properties results of the model update procedure of Case 1. For each property, the median value, the 90% confidence intervals and the coefficient of variation are computed. The design values are given for comparison [1]. . . . .	16
3.5	The structural properties of the Millikan Library building [2]. . . . .	17
3.6	The natural frequencies identified from the 2002 Yorba Linda earthquake data [13] . . .	17
3.7	The soil and foundation properties of the Millikan library building. $G_s$ is the soil shear modulus, $\rho_s$ is the soil mass density and $\theta_s$ is the Poisson ratio of the soil. $B$ and $D$ represent the dimension foundation and $e$ denotes the embedment depth of the foundation. . . . .	18
3.8	The dimensionless frequency $a_o$ and the shear wave velocity of the soil $V_s$ . . . . .	18
3.9	Normalized sway and rocking soil-foundation stiffnesses. . . . .	18
3.10	The obtained and identified natural frequencies of the Millikan Library building. . . . .	19
3.11	The obtained and identified natural frequencies of the Millikan Library building using a fixed base. . . . .	19
5.1	The structural property values obtained with the floor plans and FE model of the New Orleans. . . . .	23
5.2	The bending stiffnesses and the total displacements determined with the different approaches. . . . .	24
5.3	The random scalar applied on the initial structural properties values of Table 5.1. . . . .	27
5.4	The measured natural frequencies and modal displacements of the residential tower New Orleans. . . . .	28
5.5	The measured natural frequencies, the natural frequencies of the FEM, and the natural frequencies of the discrete Timoshenko beam model. . . . .	30
5.6	The natural frequencies of the discrete Euler-Bernoulli beam model and the uniform Timoshenko beam model. . . . .	30
5.7	The random scalar applied on the initial structural properties values of Table 5.1. . . . .	34
5.8	The modal properties of the numerical case. . . . .	34
5.9	The measured and estimated natural frequencies and MAC values between the measured and estimated mode shapes. . . . .	38
6.1	The structural property values obtained with the floor plans and FE model of the Delftse Poort. . . . .	42
6.2	The bending stiffnesses and the total displacements determined with the different approaches. . . . .	43
6.3	The random scalar applied on the initial structural properties values of Table 6.2. . . . .	45

6.4	The measured natural frequencies and modal displacements of the office tower the Delfste Poort. . . . .	46
6.5	The measured natural frequencies, the natural frequencies of the FE model and the natural frequencies of the discrete Timoshenko beam model. . . . .	48
6.6	The natural frequencies obtained with the uniform Timoshenko and discrete Euler-Bernoulli beam models. . . . .	48
6.7	The random scalar applied on the initial structural properties values of Table 6.1. . . . .	51
6.8	The modal properties of the numerical case. . . . .	51
6.9	The measured and estimated natural frequencies and MAC values between the measured and estimated mode shapes. . . . .	56
A.1	The boundary conditions of a Timoshenko beam [21, 2, 19]. . . . .	69
A.2	The continuity conditions of a piece-wise Timoshenko beam [22]. . . . .	69
A.3	The parameters (with units) of a uniform simply supported beam [19]. To verify the code, the results of this beam are reproduced using a two-piece Timoshenko beam. With that beam, each discrete element is given the same parameters as the uniform beam to obtain the same results as for the uniform beam. . . . .	71
A.4	The eigenvalues (in $rad/s$ ) of the first part of the spectrum. The first column contains the results using a two-piece Timoshenko beam model. The second column contains the results of article [19]. . . . .	72
A.5	The eigenvalues (in $rad/s$ ) at the transition frequency. The first column contains the results using a two-piece Timoshenko beam model. The second column contains the results of article [19]. . . . .	73
A.6	The first 24 eigenvalues (in $rad/s$ ) of the second spectrum. The first column contains the results using a two-piece Timoshenko beam model. The second column contains the results of article [19]. . . . .	74
B.1	The input parameter values applied by Taciroglu et al. [2]. . . . .	79
B.2	The natural frequencies of the first three modes: (a) shows the results obtained with the three-piece Timoshenko beam model, and (b) shows the results from the article by Taciroglu et al. [2]. . . . .	80
C.1	The second moment of area $I_y$ of segment 0. . . . .	84
C.2	The second moment of area $I_x$ of segment 0. . . . .	84
C.3	The second moment of area $I_y$ of segment I. . . . .	84
C.4	The second moment of area $I_x$ of segment I. . . . .	85
C.5	The second moment of area $I_y$ of segment II. . . . .	87
C.6	The second moment of area $I_x$ of segment II. . . . .	87
C.7	The second moment of area $I_y$ of segment III. . . . .	87
C.8	The second moment of area $I_x$ of segment III. . . . .	89
C.9	The second moment of area $I_y$ of segment IV. . . . .	89
C.10	The second moment of area $I_x$ of segment IV. . . . .	89
C.11	The bending stiffnesses and the total displacements determined with the different approaches. . . . .	91
C.12	The dead and variable load of the residential tower New Orleans. . . . .	91
C.13	The structural property values obtained with the floor plans and FE model of the New Orleans. . . . .	92
C.14	The second moment of area $I_x$ of segment 0 and I. . . . .	93
C.15	The second moment of area $I_y$ of segment 0 and I. . . . .	94
C.16	The second moment of area $I_x$ of segment II. . . . .	96
C.17	The second moment of area $I_y$ of segment II. . . . .	96
C.18	The second moment of area $I_x$ of segment III. . . . .	97
C.19	The second moment of area $I_y$ of segment III. . . . .	97
C.20	The bending stiffnesses and displacements determined with the different approaches. . . . .	98
C.21	The dead and variable load of the office tower the Delfste Poort. . . . .	98
C.22	The structural property values obtained with the floor plans and FE model of the Delftse Poort. . . . .	99



D.1	The identified measured modes (1-6) of the residential tower New Orleans (normalized).	106
D.2	The identified measured modes (7-11) of the residential tower New Orleans (normalized).	106
D.3	The MAC of the bending modes of Tables D.1 and D.2. The MAC values higher then 0.35 are indicated in lightsteelblue. . . . .	106
D.4	The identified measured modes (1-5) of the office tower the Delfste Poort (normalized)..	107
D.5	The identified measured modes (7-11) of the office tower the Delfste Poort (normalized).	107
D.6	The MAC of the modes of Tables D.4 and D.5. The MAC values higher then 0.35 are indicated in lightsteelblue. . . . .	107

# Nomenclature

## Abbreviations

Abbreviation	Definition
CMC	Crude Monte Carlo
DE	Differential Evolution
FEM	Finite Element model
FRF	Frequency Response Function
MAC	Modal Assurance Criterion
PSO	Partical Swarm Optimisation

## Symbols

Symbol	Definition	Unit
$A$	Cross-sectional area	[m <sup>2</sup> ]
$L$	Length	[m]
$\rho$	Mass density	[kg/m <sup>3</sup> ]
$M$	Mass	[kg]
$I$	Second moment of Inertia	[m <sup>4</sup> ]
$E$	Elastic modulus	[N/m <sup>2</sup> ]
$EI$	Bending stiffness	[Nm <sup>2</sup> ]
$\nu$	Poisson ratio	[-]
$G$	Shear modulus	[N/m <sup>2</sup> ]
$k$	Shear coefficient	[-]
$K_r$	Rotational spring stiffness	[Nm/rad]
$K_t$	Translational spring stiffness	[N/m]
$G_s$	Soil shear modulus	[N/m]
$\rho_s$	Soil mass density	[kg/m <sup>3</sup> ]
$\theta_s$	Poisson ratio soil	[-]
$e$	Embedment depth	[m]
$a_0$	Dimensionless frequency	[-]
$V_s$	Shear wave velocities	[m/s]
$f_n$	Natural frequency	[Hz]
$\phi$	Mode shape	[-]
$CV$	Coefficient of Variation	[-]
$x$	Direction main building axis	[-]
$y$	Direction orthogonal to x	[-]
$z$	Vertical building axis	[-]
$\sigma$	standard deviation	[-]
$\mu$	mean	[-]



# 1

## Introduction

In structural engineering, Finite Element (FE) models are used for simulations, design, and risk assessment. These models are solved using the Finite Element method, a numerical approach that divides a system into well-defined components known as elements. By subdividing a system into well-defined elements, whose behavior is completely understood, it becomes possible to estimate the behavior of such systems [3].

According to Zienkiewicz and Taylor [4], the FE method is considered the most appropriate tool for numerical modeling, as it can handle complex structural geometries, large assemblies of structural components, and various types of analysis. However, while FE models strive to accurately replicate the physics of their corresponding structures, discrepancies still exist between the measured and model output responses. This is due to the presence of measurement and modeling errors [3].

Measurement errors consider the inaccuracies of the measured data and arise from random noise in the measurements or from the measurement system setup. Modeling errors, on the other hand, consider the inaccuracies of the model prediction and arise from uncertainties in the geometry or boundaries, inaccurate simplifications or discretization, or uncertainty in the governing physical equations of the system [4, 5].

Due to awareness of modeling errors, efforts have been made to develop methods that improve models using measurements. This area is known as system identification and falls under control theory, a field dealing with the control of dynamical systems in engineering [6]. One technique within this area is vibration-based model updating, a technique that uses vibration measurements to enhance model accuracy. This technique includes several methods, one of which, called the indirect method, adjusts the model input parameters values to minimize the difference between the model output and the measured data. Both the model output and the measured data are in the shape of natural frequencies and mode shapes with this method.

In research, attempts have been made to use this indirect method of vibration-based model updating to estimate the structural properties of buildings. Moretti et al. [1] applied the technique to estimate the structural properties of the residential tower New Orleans in Rotterdam, utilizing a uniform Euler-Bernoulli beam model to approximate the dynamic behavior of the building. Similarly, Taciroglu et al. [2] applied the technique to estimate the dynamic stiffnesses of the soil foundation of the multi-storey building Millikan Library in Pasadena, using a uniform Timoshenko beam model to approximate its dynamic behavior. The structural properties were the input parameters of the models. Both studies used the lowest three measured bending modes to fit the model.

### 1.1. Research problem statement

While the study by Moretti et al. [1] showed that the uniform Euler-Bernoulli beam model accurately describes the lowest two measured bending modes after model updating, it also demonstrated that the chosen model fails to accurately describe the third measured bending mode after updating.

Moreover, when only the lowest two bending modes were used to fit the model, the estimates of the rotational stiffness of the foundation around the y-axis,  $K_{r,y}$ , showed large uncertainty after model updating, with a Coefficient of Variation (CV) of 46.9%, which was 6 to 7 times higher than that of other updating parameters. Similarly, for the Millikan Library building, the study by Taciroglu et al. [2] showed that the uniform Timoshenko beam model also failed to accurately describe the third measured bending mode after model updating.

According to Moretti et al. [1], the uniform Euler-Bernoulli beam model did not include all necessary aspects to accurately describe the third bending mode, limiting the accuracy of the results obtained. Additionally, they suggested that the large uncertainty in parameter  $K_{r,y}$  may also be related to the model choice, as the chosen model assumed uniform stiffness across the height and does not account for the frequency dependency of the foundation stiffness. Similarly, Taciroglu et al. [2] suggested that the discrepancy between the estimated and third measured bending mode was due to the modeling choice.

A more detailed model could provide insights into the observed discrepancy in the third bending mode and offer explanations for the large uncertainty obtained in the estimations of parameter  $K_{r,y}$ . Therefore, this research investigates the effectiveness of vibration-based model updating in estimating the structural properties of high-rise buildings using a discrete Timoshenko beam model.

## 1.2. Research objectives

The research has two objectives:

- **Main objective:** a more accurate estimation of the structural properties of high-rise buildings using vibration-based model updating.
- **Secondary objective:** the application of vibration-based model updating with a discrete Timoshenko beam model.

## 1.3. Research questions

The main question the research aims to answer is:

**How can the structural properties of high-rise buildings be more accurately estimated using vibration-based model updating?**

The sub-questions that help answer the main question are:

- What is vibration-based model updating and which methods exist in literature?
- How is vibration-based model updating applied on buildings and what are the findings?
- What are the choices in the settings of the technique, and how do these choices influence the model updating results?
- What is a Timoshenko beam model and what are important aspects of the model?
- How can the Timoshenko beam model be used to approximate the dynamic behavior of high-rise buildings?
- How effective is vibration-based model updating in estimating the structural properties of high-rise buildings using a discrete Timoshenko beam model?

## 1.4. Significance research

The research provides valuable insights into the impact of a discrete Timoshenko beam model on the estimation of structural properties for high-rise buildings. Moreover, it expands the application of vibration-based model updating to high-rise buildings that are shear-dominant and have irregular stiffnesses across their height.

## 1.5. Scope

The research focuses on modeling changes within the superstructure of the beam model. Model updating is only conducted using bending modes. Torsional and mixed modes are outside the scope of this study. Nonlinear behavior in modal properties, such as the effects of vibration amplitude or building lifespan on the natural frequencies, is not considered.

## 1.6. Limitations

Due to the variability of high-rise building in type and behavior, the results of the research cannot be universally applied to all high-rise structures. The studies of the research are conducted on two specific in-situ high-rise buildings: the residential tower New Orleans, which is bending-dominant in behavior, and the office tower Delftse Poort, which is shear-dominant in behavior and has irregular stiffness across its height. Therefore, the results can only be applied to high-rise buildings with similar types and behaviors to those of the New Orleans and Delftse Poort

In addition, the research acknowledges the limitations of the discrete Timoshenko beam model, as it remains a simplified beam model and therefore may not fully capture the complex behavior of the high-rise buildings in question. While the model can accurately estimate the overall behavior of the building, small deviations from the actual behavior are expected. Therefore, the model updating results of the structural properties should be considered indicative rather than precise.

Furthermore, the research includes studies that investigate the influence of certain settings and uncertainties in model updating using a discrete Timoshenko beam model. These studies are limited to examining the effects of model choice, its parameters, the optimization algorithm, the number of measured modal properties, and measurement uncertainties on the results obtained. The optimization algorithms are kept at their default settings.

## 1.7. Organisation research

This report starts with a theoretical background, explaining what vibration-based model updating is and reviewing the methods available in literature. Chapter 3 discusses its application on buildings. Chapter 4 describes the methodology of the research, including a description of the model updating method applied and the discrete Timoshenko beam model used. Chapter 5 presents the studies conducted on the residential tower New Orleans, evaluating the effectiveness of model updating in estimating the structural properties of a bending-dominant building. Chapter 6 presents the studies conducted on the office tower Delftse Poort, assessing the effectiveness of model updating in estimating the structural properties of a shear-dominant building with irregular stiffness across its height. The research concludes with a discussion and conclusion.

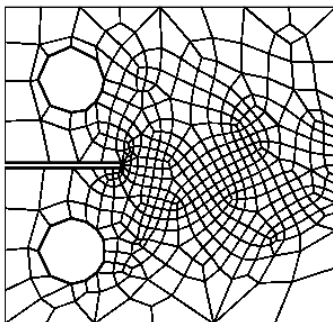
# 2

## Theoretical background

This chapter explains what vibration-based model updating is and outlines the methods available in the literature. The chapter begins with Section 2.1, which explains the background of vibration-based model updating. Next, Section 2.2 explains the key concepts and theories underlying the technique. Section 2.3 explains the type of optimization algorithms which can be applied with the technique. Finally, Section 2.4 presents the methods of vibration-based model known in literature.

### 2.1. Background

In structural engineering, Finite Element (FE) models are used for simulations, design, and risk assessments. The models are solved using the Finite Element method, a numerical technique that divides a system into a finite number of well-defined components called elements. These elements create a FE mesh, as shown in Figure 2.1. By subdividing a system into well-defined elements, whose behavior is completely understood, it becomes possible to estimate the behavior of such systems [3].



**Figure 2.1:** A typical two-dimensional finite element mesh [3].

According to Zienkiewicz and Taylor [4], the FE method is considered the most appropriate tool for numerical modeling, as it is capable of handling complex structural geometries, large assemblies of structural components, and different types of analysis. However, while FE models attempt to exactly replicate the physics of their corresponding structures, discrepancies still exist between the measured and model output responses. This is due to the presence of measurement and model errors.

Measurement errors consider the inaccuracies of the measured data and are encountered as random or systematic errors. Random errors arise from random noise in the measurements and are by nature unpredictable. Systematic errors arise from the measurement system setup and are encountered as [4]:

- Mounting errors, where incorrect mounting of the equipment does not allow accurate modeling of the system.

- Mass or stiffening errors, where attachment of equipment to the system causes mass loading and local stiffening.

The processing of data may also produce errors. Random and systematic errors can never be completely eliminated. Therefore, ensuring high-quality measurements is essential to minimize these errors. According to Friswell, there is no substitute for high-quality measurements [7].

Modeling errors consider the inaccuracies of the model prediction and are encountered as [5]:

- Model structure errors, which occur when there is uncertainty in the governing physical equations of the system.
- Model parameter errors, which involve uncertainties in properties and geometry, inaccurate application of boundary conditions, or inaccurate simplification of the model.
- Model order errors, which arise due to the discretization of complex systems, resulting in poor replication of the system.

Due to the awareness of modeling errors, efforts have been made to develop methods that improve models using measurements. This area is known as system identification and falls under control theory, a field dealing with the control of dynamical systems in engineering [6]. Vibration-based model updating falls within this area.

According to Natke, model updating can be seen as an indirect type of system identification, where 'indirect' indicates the use of a reference model to represent the system [8]. According to Friswell, model updating can be seen as a parameter estimation problem, an area within system identification where vibration measurements are used to correct the parameters of the reference model [7]. Both definitions indicate the aim of the technique to improve the accuracy of models using vibration measurements.

The technique includes several methods for updating models. However, before these methods can be explained, the key concepts and theories need to be explained, as these form the basis of the different methods known. Therefore, the key concepts and theories are explained in the next Section (Section 2.2). The methods are explained in Section 2.4.

## 2.2. Key concepts and theories

Within the area of model updating, several key concepts and theories exist that form the basis of model updating. These key concepts and theories are in detail explained in the sections below.

### 2.2.1. Parameters

Parameters refer to the variables or factors within a structural model that influence its behavior or output. They are the primary inputs of the system and may include physical parameters, mass matrices, or stiffness matrices [9].

Physical parameters are intrinsic properties of the system, such as material properties (density, elasticity), geometric properties (dimensions, shapes), or boundary conditions. Stiffness and mass matrices define the relations between the physical parameters and the model output [9].

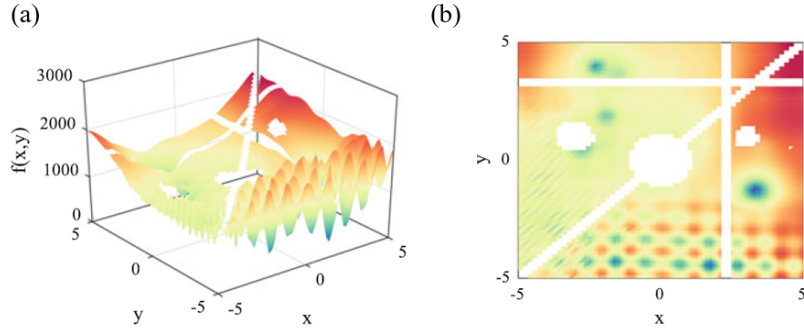
Research in this field has primarily focused on updating the physical parameters of the model [9].

### 2.2.2. Solution space

A solution space refers to an area containing all possible combinations or configurations of parameters that can be considered when searching for an optimal solution to a problem. It represents the range of feasible solutions, called candidate solutions, within a given problem domain, as illustrated in Figure 2.2.

The dimensions depend on the number of updated parameters. For example, with two variables, the search space is two-dimensional (2D). The solution space is unimodal if only one single solution exists and multimodal if it contains multiple local solutions, with one being the global solution of the given problem [10].

Real-world problems often involve complex, multidimensional multimodal solution spaces [10].



**Figure 2.2:** The solution space (a) of the variables  $x$  and  $y$  with several constraints. The top view is indicated with (b). The axis  $f(x,y)$  indicates the fitting of the solutions. The constraints are represented with white shading within the solution space, indicating infeasible solutions [10].

### 2.2.3. Parametrization

Parametrization refers to the process of reducing the set of updating parameters in order to help ill-posedness or ill-conditioning of the problem. Model update problems are often ill-posed, indicating the absence of a unique solution. Even when the problem is well-posed, it can still be ill-conditioned, as minor fluctuations in measured data or the initial model state can significantly influence the updating outcomes [7, 9]. Therefore, parametrization is the key to success in model updating. According to Friswell and Mottershead, three essential conditions should be satisfied to achieve successful parameterization [5]:

- Limit the number of parameters to avoid ill-posedness
- Choose parameters that address the model uncertainties
- Ensure that the model outputs are sensitive to the chosen parameters

### 2.2.4. Constraints

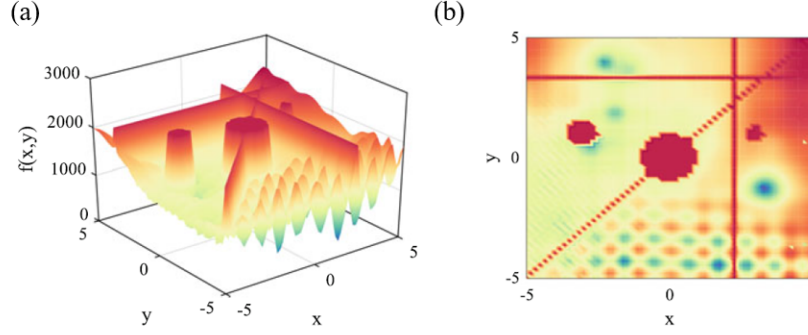
There are two types of constraints. Explicit constraints refer to the constraints directly specified in the problem formulation, indicating the limits of the system (infeasible solutions). They are considered secondary inputs of the system and depicted as white areas in Figure 2.2. Penalty functions can be applied to handle these constraints during model updating. An example of a penalty function is illustrated in Figure 2.3. To ensure that the constraints are not violated, a high penalty (indicated in dark red) is assigned to these constraints. This ensures that they are not considered as solutions [10].

Implicit constraints, in contrast, are not explicitly defined in the problem formulation. They refer to the constraints introduced during the optimization process to help the ill-posedness or ill-conditioning of the problem (regularization) [7, 9].

### 2.2.5. Regularization

Regularization refers to the process of imposing additional constraints (implicit constraints) to a problem in order to help the ill-posedness of the problem. Model update problems are often ill-posed, indicating the absence of a unique solution. Even when the problem is well-posed, it can still be ill-conditioned, as minor fluctuations in measured data or the initial model state can significantly influence the updating outcomes. Regularization modifies the problem to make it better conditioned.

It involves adding penalty terms to the objective function being optimized. Objective functions are explained in Section 2.2.8. These penalty terms ensure that parameters are adjusted minimally during optimization, guiding the process towards simpler and more stable solutions [7, 9].



**Figure 2.3:** The application of the barrier penalty function on the solution space (a). The top view is indicated with (b). The axis  $f(x,y)$  indicates the fitting of the solutions.

### 2.2.6. Optimization Algorithm

Optimization algorithms refer to the methods or procedures used to search for the global solution within the solution space of a given problem. The different types of algorithms are explained in Section 2.3. They initiate the process by generating a random set of candidate solutions (feasible solutions) within the solution space. They refine these candidate solutions to optimize or improve a certain objective function until an optimal solution is found. The manner in which these solutions are refined depends on the type of algorithm applied. [10].

### 2.2.7. Exploration and exploitation

Exploration and exploitation refer to different strategies used by algorithms to refine the candidate solutions within the solution space. With exploration, the candidate solutions are adjusted to explore different regions and to identify promising areas of the solution space. With exploitation, the best solution found so far is identified within the candidate solutions [10].

### 2.2.8. Objective function

An objective function refers to a mathematical function that quantifies the difference or discrepancy between the model output and the observed data. They evaluate the fitting of the candidate solutions. The aim is to minimize the objective function. The shape of the objective function depends on the method applied. The methods are explained in Section 2.4.

## 2.3. Optimization algorithms

This section describes the different types of algorithms which can be applied with the technique. The algorithms are divided into two categories: individual-based algorithms and population-based algorithms. These categories are explained in Sections 2.3.1 and 2.3.2, including a list of the limitations and strengths of each group.

### 2.3.1. Individual-based algorithms

Individual-based algorithms update the model by considering only one candidate solution, which is adjusted and evaluated until the objective function is minimized. The limitations and strengths of these group of algorithm are listed in Table 2.1 [10].

A strength of individual-based algorithms is their minimal need for function evaluations, as only one solution is considered. This can be advantageous in scenarios where function evaluations are computationally expensive. However, a limitation is their susceptibility to premature convergence. Real-world problems often involve complex, multidimensional solution spaces. Consequently, the algorithm may quickly converge to a local optimum, resulting in suboptimal performance on challenging optimization problems [10].

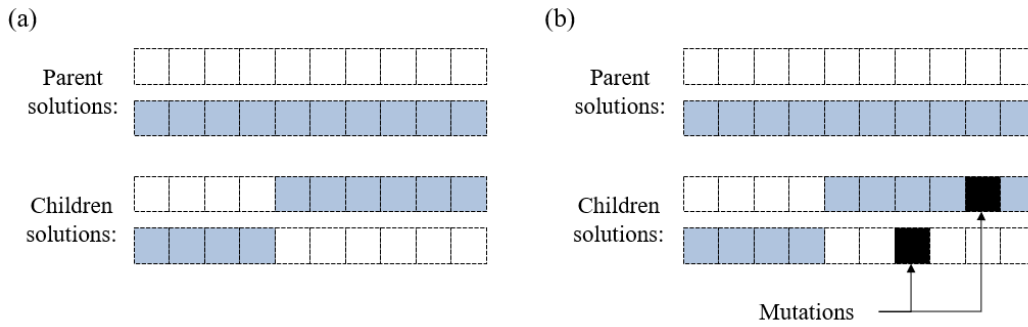
Strengths	Limitations
+ Minimal need for function evaluations	- Premature convergence

**Table 2.1:** The strengths and limits of Individual-based algorithms.

### 2.3.2. Population-based algorithms

Population-based algorithms update the model by considering a set of candidate solutions, which are evaluated and adjusted until the objective function is minimized. They are stochastic algorithms and rely on probabilistic rules. They are categorized into two groups: evolution-based algorithms and swarm-based algorithms [10].

Evolutionary algorithms mimic natural processes (crossover and mutations) to solve the optimization problem. The processes are visualized in Figure 2.4. Through crossovers, candidate solutions are combined to exploit the search space. Through mutations, candidate solutions are altered to explore the solution space [10].



**Figure 2.4:** The processes of nature mimicked with evolutionary-based algorithms: (a) illustrates the crossover process, used to exploit the search. (b) Depicts the mutation process, used to explore the solution space.

Swarm algorithms use position vectors to solve optimization problems. Position vectors denote the positions of the candidate solutions within the search space. During the updating process, the positions are updated according to specific movement rules dictating direction and speed.

Initially, the solutions move quickly in diverse directions to explore the search space. As the algorithm progresses through iterations, the magnitude of movement decreases to fine-tune the search and exploit the best positions discovered during the exploration phase [10]. The limitations and strengths are listed in Table 2.2.

A strength of population-based algorithms is their lower probability of becoming trapped in local optima: if one solution gets stuck in a local optimum, other solutions can assist in avoiding it in other iterations. However, a limitation is the requirement for more function evaluations, which increases the computational expense of these algorithms [10].

Population-based algorithms use both exploration and exploitation to refine candidate solutions. However, using these algorithms presents a challenge, as it requires a balance between exploration and exploitation. These two aspects are in conflict: a more exploratory behavior may lead to finding solutions of poor quality, as the algorithm never has a chance to improve solution accuracy. On the other hand, a more exploitative behavior risks trapping the algorithm in local minima because it does not sufficiently adapt solutions [10].



Strengths	Limits
+ Lower probability of entrapment in local optima	- More function evaluations needed
	- The balance between exploration and exploitation

**Table 2.2:** The strengths and limits of population-based algorithms.

## 2.4. Methodologies

This section describes the existing methods for model updating. The methods are divided into three categories: the direct methods, the indirect methods, and the methods using frequency domain data. For each group of methods, the limitations and strengths are listed. The details are given in the Sections below.

### 2.4.1. The direct methods using modal data

With direct methods, the models themselves (the stiffness and mass matrices) are modified to update the models. The methods falling into this category are listed in Table 2.3. The limitations and strengths of these methods are listed in Table 2.4.

Category	Methods	References
Direct methods using modal data	Reference basis methods	Baruch and Berman (1978)
	Matrix mix approach	Thoren (1972) and Ross (1971)
	Eigenstructure assignment method	Minnas and Inman (1988)
	Pole placement method	Minnas and Inman (1988)
	Inverse eigenvalue methods	Glaswell (1986)

**Table 2.3:** An overview of the direct methods [5].

A strength of these methods is their ability to exactly reproduce the measured modal data. They do not need iterative refinement or extensive batch processes, which makes them computationally cheaper. A limitation, however, is the additional requirement for accurate modeling and high-quality measurements. It is unlikely that the measured and numerical data will be exactly equal due to the presence of measurement and modeling errors [5, 11].

Another limitation is their physical meaning. The updated matrices are generally fully populated, while the initial matrices contain only non-zero values on their diagonal. Furthermore, the positive definiteness of the updated mass and stiffness matrices, as well as the connectivity of the nodes, is not guaranteed [5].

Morover, because lower frequency modes are measured and higher frequency modes contribute most to the stiffness matrices, interpreting the results becomes complicated [5].

Strengths	Limitations
+ Able to exactly reproduce the measured modal data	- Accurate modeling and high-quality measurements needed
+ Computationally cheaper	- Physical meaning updated matrices
	- Interpretation updated matrices

**Table 2.4:** The strengths and limits of the direct methods.

### 2.4.2. Indirect methods using modal data

With indirect methods, the input parameter values are adjusted to update the model. These methods are executed by an algorithm and involve the use of an objective function, denoted as  $J$  [1]:

$$J = w_f \sum_{i=1}^N \frac{|f_{n,i} - \hat{f}_{n,i}|}{\hat{f}_{n,i}} + w_\phi \sum_{i=1}^N (1 - MAC(\phi_i, \hat{\phi}_i)) \quad (2.1)$$

where index  $i$  denotes the mode considered,  $f_{n,i}$  and  $\phi_i$  denotes the estimated modal properties, and the  $\hat{f}_{n,i}$  and  $\hat{\phi}_i$  denotes the measured modal properties. The first term of the objective function  $J$  indicates the mismatch between the obtained and measured frequencies. The second term of the objective function  $J$  indicates the mismatch between the obtained and measured mode shapes. To quantify the mode shape mismatch, the Modal Assurance Criterion (MAC) is used [1]:

$$MAC(\phi_i, \hat{\phi}_i) = \frac{|\phi_i \cdot \hat{\phi}_i|^2}{(\phi_i \hat{\phi}_i)(\phi_i \cdot \hat{\phi}_i)} \quad (2.2)$$

The MAC returns a value between 0 and 1, indicating no or complete similarity between the two mode shapes. The weight factor  $w_f$  or  $w_\phi$  determine the contributions of the terms in the objective function [1] [5]. The methods falling into this category are listed in Table 2.5. The limitations and strengths of these methods are listed in Table 2.6.

Category	Methods	References
Iterative methods using modal data	Penalty functions methods	Friswell and Mottershead
	Minimum Variance methods	Collins et al. (1972)

**Table 2.5:** An overview of the iterative methods of the model updating technique [5].

The strength of these methods lies in the wide choice of parameters and the ability to weigh the terms of the objective function, which gives them power and versatility. However, it does require complete understanding which weigh factors to use [5].

Another limitation is the often ill-posed nature of the problem, indicating the potential absence of a unique solution. Additionally, long computational times can be expected due to the need to evaluate the model at every iteration [5]. Convergence problems may also arise [7].

Strengths	Limitations
+ Wide choice of parameters	- Understanding which weigh factors to use
+ Weighing the terms objective function	- Ill-posed nature of the problem
	- Long computational times
	- Convergence problems

**Table 2.6:** The strengths and limits of the indirect methods.

Additionally, it is essential for these methods that the model and measured data correspond to the same mode, which can pose a significant challenge. Simply arranging the natural frequencies in ascending order of magnitude may not suffice, especially when two modes are closely located in the frequency domain. Moreover, obtaining accurately measured modes can be problematic, and certain modes may not be excited during the experiment, resulting in no corresponding measured mode for the analytical modes [5].

### 2.4.3. Methods using Frequency Domain Data (FDD)

The last group involves methods that use an objective function directly based on the Frequency Response Function (FRF) data to update the model. The methods falling within this category are listed in table 2.7. Both methods are based on the equation of motion in the frequency domain for viscous damping:

$$[-\omega^2 M + i\omega C + K]x(\omega) = f(\omega) \quad (2.3)$$

The equation error approach minimizes the error in the equation of motion, given as:

$$\epsilon_{ee} = f(\omega) - [-\omega^2 M + i\omega C + K]x(\omega) \quad (2.4)$$

Where  $x(\omega)$  and  $f(\omega)$  are measured quantities. The output error approach minimizes the output error, defined as the difference between the measured and the estimated response:

$$\epsilon_{oe} = [-\omega^2 M + i\omega C + K]^{-1}f(\omega) - x(\omega) \quad (2.5)$$

Because the Frequency Response Functions (FRFs) are directly used, the force and displacement are not available individually. Therefore, with both approaches, the force spectrum is assumed to be white noise, and the displacement is replaced by the FRFs data. The limitations and strengths are listed in Table 2.8.

Category	Methods	References
Methods using FDD	Equation error approach	Friswell and Penny (1992)
	Output error approach	

**Table 2.7:** An overview of the iterative methods of the model update technique [5].

Both methods eliminate the need for extracting natural frequencies and mode shapes, which can be challenging for structures with closely spaced modes or high modal density. Although Frequency Response Functions (FRFs) contain more data than the modal model, it should however not be assumed that they provide a proportionally increased amount of information [5].

However, a significant limitation is that damping must be included in the finite element model. Damping is important for achieving good correspondence between the measured and predicted FRFs, however, it is difficult to model accurately. Methods that use modal data do not face this issue, as they can rely on undamped models. This is possible because natural frequencies and damping ratios can be separated [5].

Strengths	Limitations
+ No need for extraction modal properties	- The inclusion of damping

**Table 2.8:** The strengths and limits of the methods using FDD.

## 2.5. Conclusion

This chapter provided an in-depth explanation of vibration-based model updating, a technique that improves models using vibration measurements. This technique falls within the domain of system identification and falls under the control theory, a field dealing with the control dynamic systems in engineering. The technique uses a model to represent the system and includes several methods:

- **Direct methods** modify the model themselves (the stiffness and mass matrices) to align the model output with the measured data. These methods can exactly reproduce the measured modal data. However, they require accurate modeling and high-quality measurements. Another limitation is the physical meaning, as the updated matrices are generally fully populated, while the initial matrices typically contain non-zero values only on their diagonals.
- **Indirect methods** adjust the input parameter values to minimize the difference between the model output and the measured data. Both the model output and the measured data are in the shape of natural frequencies and mode shapes. The strength of these methods lies in the wide choice of updating parameters, giving them power and versatility. However, the problems are often ill-posed, meaning there may be no unique solution. Additionally, long computational times can be expected due to the need to evaluate the model every time a change in the input parameter values is made.

- **Methods based on Frequency Domain Data** utilize an objective function directly based on the FRF data to update the model. These methods eliminate the need to extract modal properties, which can be challenging for structures with closely spaced modes or high modal density. However, a significant limitation is the necessity of including damping in the finite element model. Damping is crucial for achieving good correspondence between the measured and predicted FRFs, but it is difficult to model accurately.

While each method for vibration-based model updating offers unique advantages, they also come with specific challenges and limitations. How vibration-based model updating is applied on buildings is shown in the Chapter 3.

# 3

## Literature review

This chapter shows how vibration-based model updating is applied on building. It describes two applications, which both use the technique to estimate the structural properties. The first application is a study conducted by Moretti et al. [1] on the residential tower New Orleans in Rotterdam, explained in Section 3.1. The second application is a study on the Millikan Library building in Pasadena conducted by Taciroglu et al. [2], explained in Section 3.2. Their findings are highlighted at the end.

### 3.1. The residential tower New Orleans in Rotterdam

The first application involves a study on the residential tower New Orleans in Rotterdam, the Netherlands, conducted by Moretti et al. [1]. The high-rise building is shown in Figure 3.1.



**Figure 3.1:** A picture of the residential tower New Orleans.

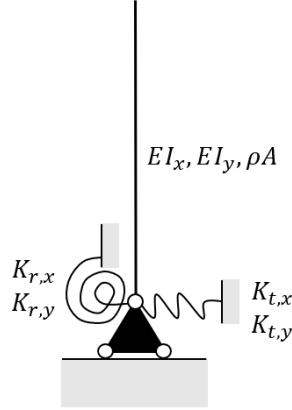
High-rise buildings are sensitive to wind-induced vibrations, making these vibrations important for the design of the serviceability limit state (SLS) and the ultimate limit state (ULS).

The main parameters of these vibrations are the eigenfrequencies and damping ratios. Accurately estimating these parameters during the design phase is difficult, as they depend on factors such as building mass, building stiffness and foundation stiffness. Therefore, to obtain a better prediction of the dynamic behavior, a better prediction of these structural properties is needed.

To obtain this, it is important to know the actual values of these structural properties and how they relate to the design values. Therefore, to obtain this information, an indirect method of vibration-based model updating is used, explained in Section 2.4.2. The weight factors of the objective function (Equation 2.1) are set to 1.

The model applied to approximate the dynamic behavior of the New Orleans is a beam model and is shown in Figure 3.2. The superstructure is modeled as a uniform Euler-Bernoulli beam with bending stiffnesses  $EI_x$  and  $EI_y$ , and a mass per unit length  $\rho A$ . The foundation is modeled as a combination of rotational ( $k_{r,x}$  and  $k_{r,y}$ ) and translational springs ( $k_{t,x}$  and  $k_{t,y}$ ). Due to the choice of the model, only bending modes can be considered in the updating process.

The initial input parameter values are given in Table 3.1. The measured modal data applied to fit the model are listed in Table 3.2, obtained by analyzing vibration measurements on the 15th, 34th, and 44th floors. The measurement setup used to obtain the vibration measurements is shown in Figure 3.3.



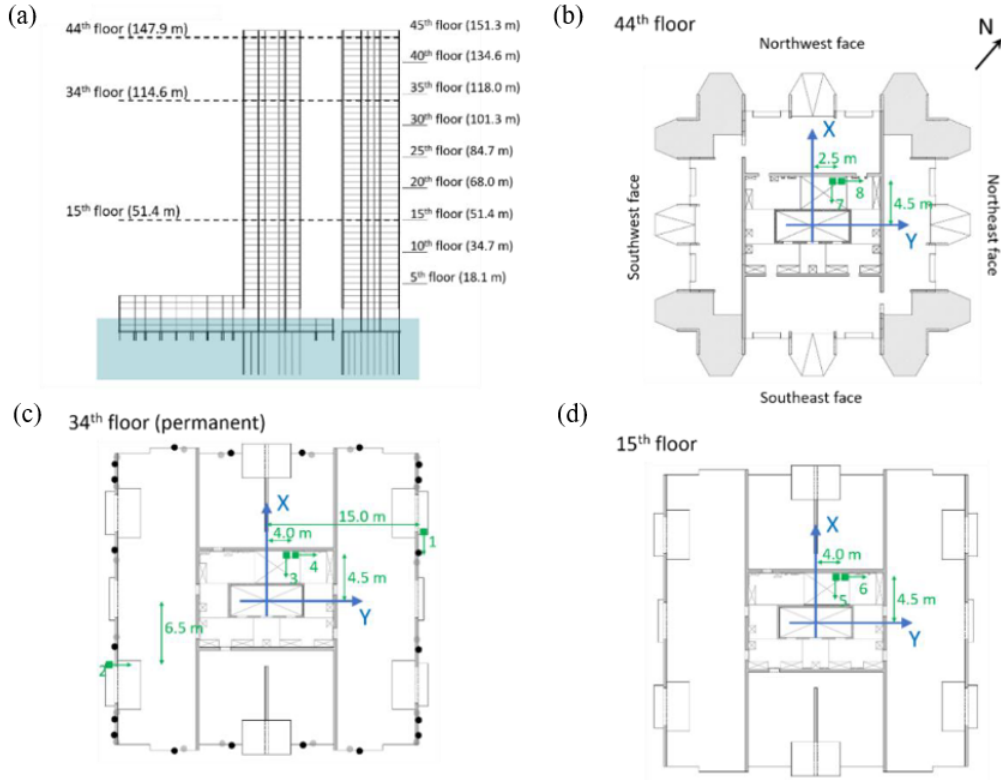
**Figure 3.2:** The uniform Euler-Bernoulli beam model (2D) used to approximate the dynamic behavior of the high-rise building.

Properties [unit]	
$EI_x$ [Nm <sup>2</sup> ]	$3.06e13$
$EI_y$ [Nm <sup>2</sup> ]	$2.43e13$
$K_{r,x}$ [Nm/rad]	$1.88e12$
$K_{r,y}$ [Nm/rad]	$1.88e12$
$K_{t,x}$ [N/m]	$\infty$
$K_{t,y}$ [N/m]	$\infty$
$M$ [kg]	$6.50e7$
$L$ [m]	155
$A$ [m <sup>2</sup> ]	841
$\rho$ [kg/m <sup>3</sup> ]	500

**Table 3.1:** The design values assigned as initial input parameter values for the Euler-Bernoulli beam model [1].

Mode	1	2	3	4	5
Dominant direction	X	Y	Y	X	Y
Natural frequency [Hz]	0.282	0.291	1.332	1.527	2.771
<b>Modal displacement</b>					
Height 1 (51.4 m)	−0.30	−0.29	−0.90	0.89	−0.73
Height 2 (114.6 m)	−0.71	−0.78	0.19	−0.23	0.83
Height 3 (147.9 m)	−1.00	1.00	1.00	−1.00	−1.00

**Table 3.2:** The identified pure bending modes in dominant x- or y- direction [1, 12].



**Figure 3.3:** (a) Picture and (b, c, d and e) drawings indicating the acceleration sensors positions (green squares) on the 15th floor, the 34th floor, and the 44th floor. The green arrows indicate the directions of the acceleration sensors [1].

Two scenarios are considered:

- Case 1: model updating with 2 bending modes in x- and y-direction.
- Case 2: model updating with 2 bending modes in x-direction and 3 bending modes in y-direction.

In both cases, the parameters  $EI_x$ ,  $EI_y$ ,  $Kr_x$ ,  $Kr_y$ ,  $Kt_x$ ,  $Kt_y$ , and  $\rho$  are updated. The findings are given in Tables 3.3 and 3.4.

According to Table 3.3, both Case 1 and Case 2 show alignment with the measured modal data after updating. However, the accuracy ( $\pm\sigma$ ) of Case 2 is lower.

The Modal Assurance Criterion (MAC) in Case 2 indicates that the third bending mode in y-direction is not properly matched. This suggests that the chosen model is only capable of accurately describing the lower two bending modes in both directions. According to Moretti et al. [1], with higher bending modes, other modeling aspects become important, which are not included in the chosen model.

Mode	Design [Hz]	Measured [Hz]	Case 1 ( $\pm\sigma$ ) [Hz]	Case 2 ( $\pm\sigma$ ) [Hz]	MAC Case 1 %	MAC Case 2 %
1 (x)	0.166	0.282	0.282 ( $\pm 5e-5$ )	0.286 ( $\pm 0.03$ )	99.84	97.58
2 (y)	0.153	0.291	0.291 ( $\pm 5e-5$ )	0.253 ( $\pm 0.02$ )	99.73	99.22
4 (y)	0.989	1.322	1.332 ( $\pm 5e-5$ )	1.333 ( $\pm 0.04$ )	99.90	98.47
5 (x)	1.089	1.527	1.516 ( $\pm 5e-5$ )	1.548 ( $\pm 0.08$ )	99.70	95.45
7 (y)	2.823	2.771	-	<b>3.111 (<math>\pm 0.56</math>)</b>	-	<b>78.11</b>

**Table 3.3:** The modal properties of the residential tower New Orleans in terms of natural frequencies and MAC [1].

Looking at the results of the structural properties of Case 1 (Table 3.4), it can be observed that, except for  $\rho$ , the design values of the structural properties fall outside the value ranges obtained with updating.

The values obtained provide a good basis for making design choices that can lead to better prediction of the dynamic behavior.

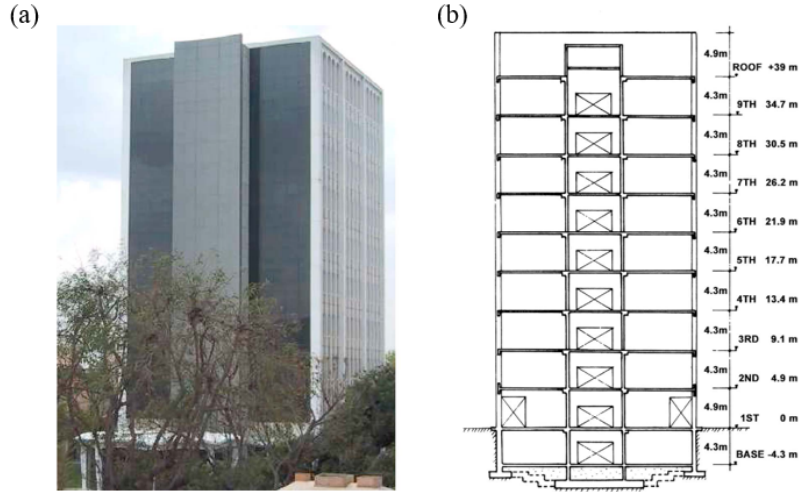
However, this cannot be said for the rotational stiffness  $K_{ry}$ . A large uncertainty (46.9%) can be seen in the obtained results of the rotational stiffness  $K_{ry}$ , which is 6 to 7 times higher than that of other updating parameters. According to Moretti et al. [1], this may be also related to the choice of model, as the current model assumes uniform stiffness over the whole height and does not consider the frequency dependence of the foundation stiffness. A more detailed model could provide insight into the mismatch of the third bending mode and the observed uncertainty in the estimate for  $K_{ry}$ .

Property [unit]	Design	Median	Lower bound ( $\pm\%$ )	Upper bound ( $\pm\%$ )	CV (%)
$EI_x$ [Nm <sup>2</sup> ]	3.06e13	9.70e13	8.62e13 (-11.1 %)	10.8e13 (+11.3 %)	7.0 %
$K_{r,x}$ [Nm/rad]	1.88e12	3.07e12	2.76e12 (-10.1 %)	3.33e13 (+8.6%)	5.8 %
$K_{t,x}$ [N/m]	$\infty$	2.85e9	2.85e9 (-9.5 %)	3.07e13 (+7.7 %)	5.4 %
$EI_y$ [Nm <sup>2</sup> ]	2.43e13	6.51e13	5.78e13 (-11.2 %)	7.50e13 (+15.2 %)	7.9 %
$K_{r,y}$ [Nm/rad]	1.88e12	1.11e13	0.64e13 (-42.6 %)	2.29e13 (+105.8 %)	<b>46.9 %</b>
$K_{t,y}$ [N/m]	$\infty$	2.20e9	1.98e9 (-10.3 %)	2.40e13 (+9.0 %)	5.9 %
$\rho$ [kg/m <sup>3</sup> ]	500	468.6	432.0 (-7.8 %)	502.9 (+7.3 %)	4.7 %

**Table 3.4:** The structural properties results of the model update procedure of Case 1. For each property, the median value, the 90% confidence intervals and the coefficient of variation are computed. The design values are given for comparison [1].

### 3.2. The Millikan Library building in Pasadena

The second application is a study on the Millikan Library building at the California Institute of Technology in Pasadena, conducted by Taciroglu et al. [2]. The multi-storey building is show in Figure 3.4. This study presents a new approach to estimate the dynamic stiffnesses of soil foundation systems using model updating.



**Figure 3.4:** A picture (a) and a schematic (b) of the Millikan Library building [2].

The model chosen to approximate the dynamic behavior of the Millikan Library is a beam model and is shown in Figure 3.5. The superstructure is modeled as a uniform Timoshenko beam with bending stiffness  $EI$ , shear stiffness  $kGA$  and a mass per unit length  $\rho A$ . The foundation is modeled as a combination of rotational ( $k_{r,x}$  and  $k_{r,y}$ ) and translational springs ( $k_{t,x}$  and  $k_{t,y}$ ) and includes frequency dependency. The model takes both bending and shear deformations into account.

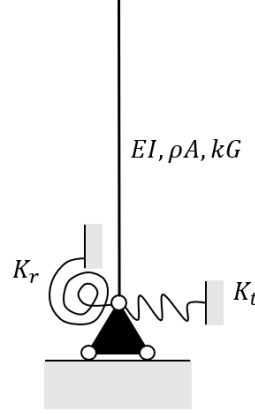


Dimensionless parameters (equations 3.1 and 3.2) are defined through the relation of the input parameters:

$$b^2 = \frac{\rho A \omega^2 L^4}{EI}, s^2 = \frac{EI}{kGA L^2} \quad (3.1)$$

$$k_t(\omega_i) = \frac{K_t(\omega_i)}{kGA/L}, k_r(\omega_i) = \frac{K_r(\omega_i)}{EI/L} \quad (3.2)$$

where  $\omega_i$  indicates the frequency dependency of the foundation. The initial input parameter values are given in table 3.5. The measured natural frequencies, obtained by analyzing vibration measurements from the 2002 Yorba Linda earthquake [13], are listed in Table 3.5.



**Figure 3.5:** The uniform Timoshenko beam model (1D) use to approximate the dynamic behavior of the high-rise building.

Properties [unit]	
$EI$ [Nm <sup>2</sup> ]	17.75e3
$M$ [kg]	11.95e6
$L$ [m]	43.28
$A$ [m <sup>2</sup> ]	483

**Table 3.5:** The structural properties of the Millikan Library building [2].

Mode	1	2	3
Natural frequency [Hz]	1.68	6.64	12.48

**Table 3.6:** The natural frequencies identified from the 2002 Yorba Linda earthquake data [13]

Two scenarios are considered:

- Scenario I: the measurements of the rocking response of the foundation are considered.
- Scenario II: the measurements of the rocking response of the foundation are not considered.

When the measurements of the rocking response are considered (scenario I), an iterative method of the model updating is applied, which is explained in Section 2.4.2. The dimensionless parameters  $b$ ,  $s$ ,  $k_r(\omega_1)$  and  $k_t(\omega_1)$  are chosen as the updating parameters. Only the lowest measured bending mode is used to fit the model. The weight factors of the objective function indicated with Equation 2.1 are set to 1.

After updating the dimensionless parameters, the physical values of the dynamic stiffness of the soil foundation system are obtained using the Equation 3.3:

$$K_t(\omega_1) = \frac{K_t M}{b^2 s^2}, K_r(\omega_1) = \frac{K_t M L^2}{b^2} \quad (3.3)$$

When the measurements of the rocking response are not considered (scenario II), equations 3.4 till 3.9 are used to approximate the stiffness of a rigid rectangular foundation:

$$K_t(\omega_i) = K_t^s k_t^d(\omega_i) \quad (3.4)$$

$$K_t^s(\omega_i) = \frac{G_s D}{2(2 - \theta_s)} [6.8(B/D)^{0.65} + 2.4 + 0.8(B/D - 1)] [1 + (0.33 + \frac{1.34}{1 + B/D})(\frac{2e}{D})^{0.8}] \quad (3.5)$$

$$k_t^d(\omega_i) = 1 \quad (3.6)$$

$$k_r(\omega_i) = K_r^s k_r^d(\omega_i) \quad (3.7)$$

$$K_r^s = \frac{G_s D^3}{8(1 - \theta_s)} [3.2(B/D) + 0.8] [1 + (\frac{e}{D} + \frac{1.6}{0.35 + B/D})(\frac{2e}{D})^2] \quad (3.8)$$

$$k_r(\omega_i) = 1 - \frac{da_o(\omega_i)^2}{b + a_o(\omega_i)^2} \quad (3.9)$$

where  $k_t^d$  and  $k_r^d$  are dimensionless parameter representing the frequency dependency, and  $K_t^s$  and  $K_r^s$  are the static stiffnesses of the foundation. The parameter values are listed in Table 3.7. Functions of the dimension are listed in Table 3.8.

Property [unit]	Value
$G_s$ [N/m]	2.68e8
$\rho_s$ [kg/m <sup>3</sup> ]	2.7e3
$\theta_s$ [-]	0.3
$B$ [m]	23
$D$ [m]	21
$e$ [m]	4

**Table 3.7:** The soil and foundation properties of the Millikan library building.  $G_s$  is the soil shear modulus,  $\rho_s$  is the soil mass density and  $\theta_s$  is the Poisson ratio of the soil.  $B$  and  $D$  represent the dimension foundation and  $e$  denotes the embedment depth of the foundation.

Property [unit]	Function
$a_o$ [-]	$\frac{\omega_i D}{2V_s}$
$V_s$ [m/s]	$\sqrt{\frac{G_s}{\rho_s}}$

**Table 3.8:** The dimensionless frequency  $a_o$  and the shear wave velocity of the soil  $V_s$ .

The findings of the study are given in Table 3.9 and 3.10. To compare the obtained values with a previous study conducted by Ghahari et al. [13], where the FE model of the Millikan Library is directly updated using the same measured modal data, the sway and rocking stiffness values are normalized by  $Ga$  and  $Ga^3$  respectively, where  $G_s = 2.68e8$  N/m denotes the soil shear modulus and  $a = 13.7$  m the reference foundation length.

According to Table 3.9, the stiffness values obtained using the presented methods are close to the identified values from the study conducted by Ghahari et al. [13].

Study	Sway [-]	Rocking [-]
Ghahari et al. [13]	6.2	3.9
Scenario I	6.7	4.2
Scenario II	6.7	4.1

**Table 3.9:** Normalized sway and rocking soil-foundation stiffnesses.

According to Table 3.10, the first two natural frequencies are close to the identified values. However, the third natural frequency shows a significant difference from the identified value.

Mode	Measured		FE model [13]	Scenario I
	[13]	[Hz]	[Hz]	[Hz]
1	1.68		1.69	1.68
2	6.64		6.64	6.67
3	12.48		12.53	14.05

**Table 3.10:** The obtained and identified natural frequencies of the Millikan Library building.

To determine if this discrepancy is due to the choice of foundation, also a scenario with a fixed-base was considered. The results obtained with a fixed-base are given in Table 3.11.

The third natural frequency from the Timoshenko beam model with a fixed base show a 12% difference compared to the FE model. This suggests that the error is not solely due to the chosen foundation. A more detailed model is needed to predict the higher mode responses accurately.

Mode	Measured		FE model [13]	Scenario I
	[13]	[Hz]	[Hz]	[Hz]
1	2.05		2.07	2.07
2	-		7.51	7.56
3	-		13.96	15.67

**Table 3.11:** The obtained and identified natural frequencies of the Millikan Library building using a fixed base.

### 3.3. Conclusion

This chapter shows how vibration-based model updating is applied to estimate the structural properties of buildings with two applications. The first application is a study on the residential tower New Orleans by Moretti et al. [1], which used an uniform Euler-Bernoulli beam model to approximate the dynamic behavior of the residential tower New Orleans. The second application is a study on the Millikan Library building in Pasadena by Taciroglu et al. [2], which used a uniform Timoshenko beam model to approximate its dynamic behavior.

The study of Moretti et al. [1] showed that while the uniform Euler-Bernoulli beam model is able to describe the lowest two bending modes after model updating, it fails to accurately describe the third measured bending mode after updating. Moreover, when only the lowest two bending modes were used to fit the model, the estimates of the rotational stiffness of the foundation in the y-direction,  $K_{r,y}$  showed a large uncertainty after model updating, with a coefficient of variation of 46.9 %, which is 6 to 7 times higher than that of other updating parameters. Similarly, the study of Taciroglu et al. [2] showed that the uniform Timoshenko beam after model updating failed to accurately describe the third measured bending mode.

According to Moretti et al. [1], the uniform Euler-Bernoulli beam model did not include all necessary aspects to accurately describe the third bending mode, limiting the accuracy of the results obtained. Additionally, they suggested that the large uncertainty in parameter  $K_{r,y}$  may also be related to the model choice, as the chosen model assumed uniform stiffness across the height and does not account for the frequency dependency of the foundation stiffness. Similarly, Taciroglu et al. [2] suggested that the discrepancy between the estimated and third measured bending mode was due to the modeling choice.

Applying a more detailed model with vibration-based model updating could provide insights into the observed discrepancy in the third bending mode and offer explanations for the significant uncertainty encountered in the estimations of parameter  $K_{r,y}$ . Therefore, this research investigates the effectiveness of vibration-based model updating in estimating the structural properties of high-rise buildings using a discrete Timoshenko beam model. The methodology of the research is explained in Chapter 4.

# 4

## Methodology

This chapter explains the research methodology, including a detailed description of the model updating method applied and the discrete Timoshenko beam model used. Section 4.1 discusses the vibration-based model updating method applied. Section 4.2 describes the discrete Timoshenko model.

### 4.1. The model updating method

For this research, an indirect method of vibration-based model updating is chosen, the same method applied by Moretti et al. [1] and Taciroglu et al. [2]. This method adjusts the input parameter values of the chosen model to minimize the difference between the model output and the measured data [1]. A detailed explanation of the chosen method is provided in Section 2.4.2. The chosen model, the discrete Timoshenko beam model, is described in Section 4.2.

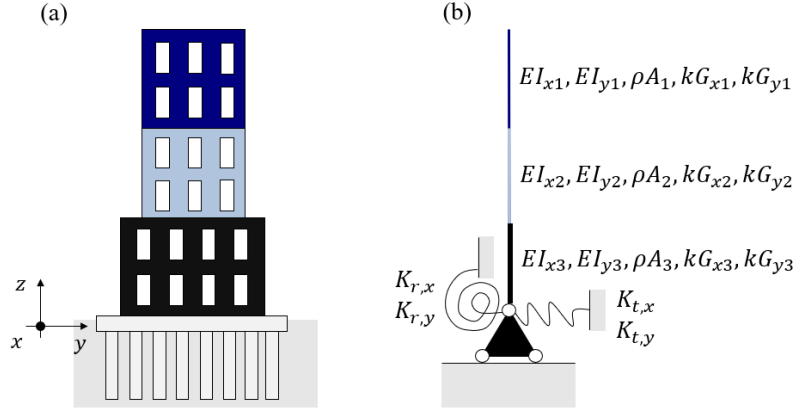
The objective function, which is used to quantify the difference between the model output and the measured data, is indicated in Equation 2.1. The weight factors of the objective function are set to 1, consistent with the studies by Moretti et al. [1] and Taciroglu et al. [2]. Both the model output and the measured data are in the form of natural frequencies and mode shapes, referred to as modal properties. However, compared to the model mode shapes, the measured mode shapes are spatially incomplete. The number of sensors used to obtain the modal displacements is much smaller than the degrees of freedom in the chosen model [7]. Only the modal displacements at the measured heights are known, whereas the model provides the modal displacements at all heights.

Since the mismatch of the mode shapes in the objective function is quantified using the Modal Assurance Criterion (MAC), as indicated in Equation 2.2, and since the MAC requires that the mode shapes be of the same dimension, the mode shapes of the model are reduced to match the dimensions of the measured mode shapes. This is done by considering only the modal displacements of the model corresponding to the measured heights.

The optimization algorithms used in this research are the Sequential Least Squares Quadratic Programming (SLSQP) algorithm, the Differential Evolution (DE) algorithm, and the Particle Swarm Optimization (PSO) algorithm. SLSQP is an individual-based optimization algorithm that uses the gradient of the objective function to guide the search in the solution space [10]. DE and PSO are evolutionary and swarm algorithms, respectively. Details of individual-based optimization algorithms are found in Section 2.3.1. Details of the evolutionary and swarm optimization algorithms are found in Section 2.3.2.

### 4.2. The discrete Timoshenko beam model

The model used to approximate the dynamic behavior of the two high-rise buildings is a discrete Timoshenko beam model, as illustrated in Figure 4.1. The superstructure is modeled as a discrete Timoshenko beam with bending stiffnesses  $EI_x$  and  $EI_y$ , shear stiffnesses  $kG_xA$  and  $kG_yA$ , and masses  $\rho A$  per unit length.



**Figure 4.1:** A schematic representation of (a) a high-rise building and (b) the discrete Timoshenko beam model used to approximate the dynamic behavior of that building.

The foundation is modeled as a combination of rotational ( $k_{r,x}$  and  $k_{r,y}$ ) and translational springs ( $k_{t,x}$  and  $k_{t,y}$ ). The model accounts for both bending and shear deformations, and considers irregular stiffness across the height. The model derivation can be found in Appendix A. Its verification can be found in Appendix B. The written Python code can be found in Appendix F.

An important aspect of the model is its frequency spectrum. The frequency spectrum of the discrete Timoshenko beam model is separated in two parts, by a so-called transition frequency, indicated with Equation 4.1:

$$\omega = \sqrt{\frac{kGA}{\rho I}} \quad (4.1)$$

where the eigenmodes below and above this transition frequency exhibit differences in behaviors. Details of the eigenmodes are described in Appendix A.

In the part of the frequency spectrum above this transition frequency, a phenomenon called "the second spectrum of modes" can occur, initially observed by Traill-Nash and Collar [14]. The shapes of these eigenmodes are identical to those appearing in the first part of the spectrum, however, they are associated with higher frequencies.

The findings of Traill-Nash and Collar caused various responses: Abbas and Thomas argued that, aside from a simply supported beam, no second spectrum of modes exists, stating that earlier conclusions about its existence were misinterpretations [9]. Bhashyam and Prathap provided proof of its existence for various boundary conditions, and developed a methodology to categorize the different modes [15]. Levinson and Cooke claimed that, even for the simply supported Timoshenko beam, only one spectrum of modes exists [16].

The ongoing debate may explain why many papers published since 1921 do not provide a complete solution and often fail to address the potential existence of a second spectrum of modes. Despite this, the phenomenon is physically significant because the modes in this spectrum share the same wavelength as the modes of lower frequencies, even if their frequencies differ. This means, for example, that a high-frequency oscillating force could excite and potentially resonate with these modes. Ignoring them could lead to overlooking such phenomena. However, for this research, this is not considered a problem, as the measured modes used are expected to be well below this transition frequency. Details about the phenomenon can be found in Appendix A.

How exactly the discrete Timoshenko beam model is applied to approximate the dynamic behaviors of the residential tower the New Orleans and on the office tower the Delftse Poort, is explained per application in the Chapters 5 and 6.

# 5

## The residential tower New Orleans

To evaluate the effectiveness of vibration-based model updating in estimating the structural properties of a bending-dominant building using a discrete Timoshenko beam model, this chapter applies the technique to the residential tower New Orleans in Rotterdam.

Section 5.1 explains the various studies conducted on the New Orleans. Section 5.2 presents the results of these studies, along with a comparison to the findings from the study by Moretti et al. [1]. Finally, the main findings are summarized in Section 5.3.

### 5.1. Studies

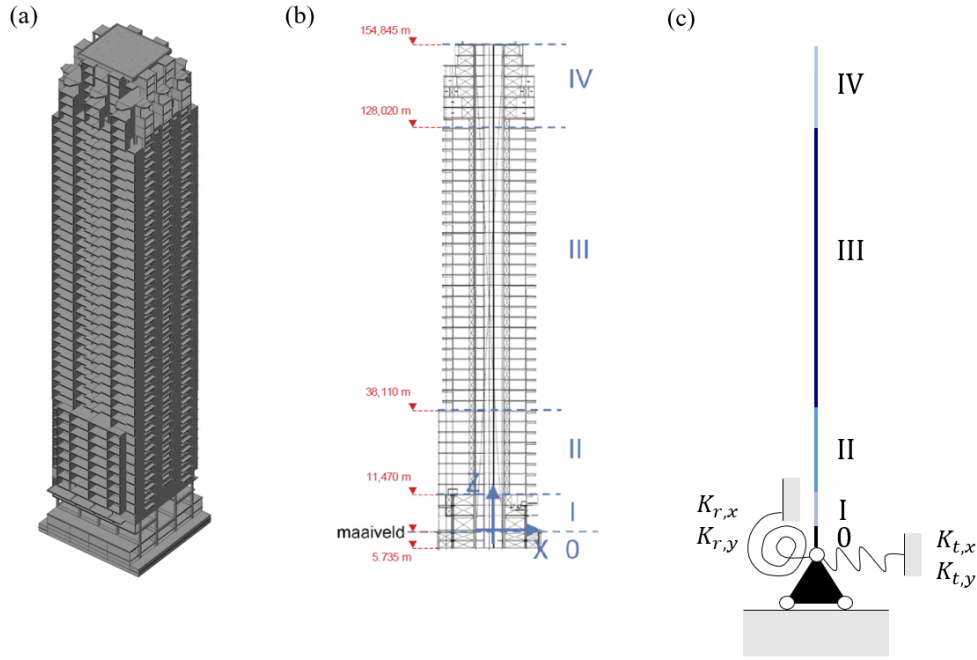
The research consists of six studies. The first five studies examine how different settings and uncertainties in the model updating process, using a discrete Timoshenko beam model, influence the results obtained. The final study applies model updating to estimate the structural properties of the New Orleans, using the discrete Timoshenko beam model and the measured modal properties. The details of the studies are explained in Sections 5.1.1 to 5.1.6. Appendix D describes how the measured modal properties were obtained.

#### 5.1.1. Influence Features Model

The first study is about the features of the Timoshenko beam model. The model incorporates shear deformations and accounts for irregular stiffness across its height. To assess the influence of these features in the beam model, the natural frequencies and mode shapes obtained with this model are compared with those obtained using the uniform Timoshenko beam and discrete Euler-Bernoulli beam model.

The discrete Timoshenko beam model used in the studies to approximate the dynamic behavior of the New Orleans is created using the Finite Element (FE) model of the building. This FE model, shown in Figure 5.1, is divided into five segments (0 till IV). The segments were defined based on the floor plans. The floors within a segment have the same floor plan in the FE model. Therefore, a five-segment Timoshenko beam model is used to represent the FE model in Figure 5.1.

The initial structural property values applied in this beam model are shown in Table 5.1. These values were obtained using the information of the FE model and the floor plans. Details of the approach with which the structural property values were obtained are described in Appendix C. The second moment of area around the x-axis is denoted as  $I_x$ . The rotational stiffness of the foundation around the x-axis is denoted as  $K_{r,x}$ . The translation stiffness of the foundation in x-direction is denoted as  $K_{t,x}$ .



**Figure 5.1:** Pictures of (a) the FE model of the New Orleans, (b) the 5 segments defined for the New Orleans model, and (c) the five-segment Timoshenko beam model.

Segment	0	I	II	III	IV
$L$ [m]	5.735	11.47	26.640	89.91	26.825
$A$ [m <sup>2</sup> ]	784	784	784	784	784
$I_x$ [m <sup>4</sup> ]	1444	1500	1393	1389	1293
$I_y$ [m <sup>4</sup> ]	1532	1548	2796	2324	760
$\rho$ [kg/m <sup>3</sup> ]	1933	495	486	440	383
$E$ [N/m <sup>2</sup> ]	38.2e9	38.2e9	38.2e9	38.2e9	32.8e9
$\nu$ [-]	0.2	0.2	0.2	0.2	0.2
$G$ [N/m <sup>2</sup> ]	15.9e9	15.9e9	15.9e9	15.9e9	13.7e9
$k$ [-]	0.85	0.85	0.85	0.85	0.85
<b>Boundary</b>					
$K_{r,x}$ [GNm/rad]	2625	$K_{t,x}$ [GN/m]	4		
$K_{r,y}$ [GNm/rad]	2380	$K_{t,y}$ [GN/m]	19		

**Table 5.1:** The structural property values obtained with the floor plans and FE model of the New Orleans.

Obtaining the bending stiffnesses of the segments was challenging. Two approaches were applied to determine the bending stiffness  $EI_s$  (with  $s$  being  $x$  or  $y$ ) per segment.

The first approach calculated the bending stiffness per segment  $EI_s$  using the elastic modulus  $E$  applied in the FE model and the second moment of area  $I$ , calculated using the term  $\frac{1}{12}bh^3$  and the Steiner rule:

$$EI_s = E_s \left( \frac{1}{12}bh^3 + bd^2 \right) \quad (5.1)$$

The second approach considered the segments of the FE model. The FE model of the segment was clamped at the bottom, and a force  $F$  was applied at the top, resulting in a relative displacement  $\Delta$ . The bending stiffness  $EI_s$  was then determined using this displacement  $\Delta$  and Equation C.3:

$$EI_s = \frac{FL^3}{3\Delta} \quad (5.2)$$

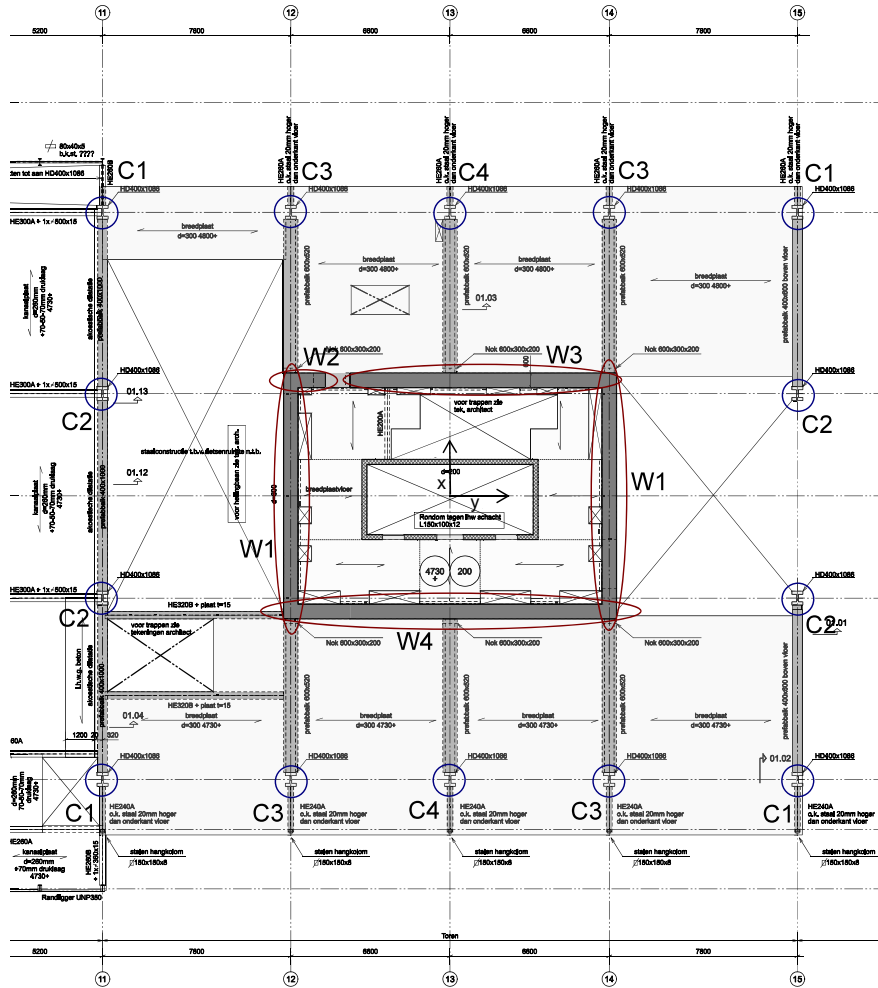
More information about the applied approaches can be found in Appendix C. The values obtained from each approach are shown in Table 5.2.

Additionally, the displacements at the top, using either a five-segment beam model with these bending stiffnesses or the FE model, are presented, when a force of 10,000 kN is applied and the base of the model is fixed.

	$EI_x$ [Nm <sup>2</sup> ]		$EI_y$ [Nm <sup>2</sup> ]		<b>FEM</b>	
	Steiner	FE segment	Steiner	FE segment		
0	3.19e13	2.37e13	6.71e13	5.87e13	<b>Y</b>	<b>X</b>
I	3.31e13	3.62e13	6.77e13	3.66e13		
II	3.08e13	1.04e14	1.22e14	4.76e13		
III	3.07e13	8.92e13	1.04e14	4.59e13		
IV	2.45e13	2.97e13	2.85e13	2.73e13		
$\Delta u$ [mm]	887	472	344	613	516	394

**Table 5.2:** The bending stiffnesses and the total displacements determined with the different approaches.

While both approaches offer ways to determine the bending stiffness  $EI_s$ , both have limitations in accurately estimating it. With the Steiner approach, the obtained displacement in the y-direction ( $\Delta u = 887$  mm) showed significant differences with the displacement of the FE model ( $\Delta u = 516$  mm). This is suspected to be due to the exclusion of certain columns and walls in the calculation of the second moment of area  $I_x$ . These are labeled as C1 and C2 in Figure C.2 and Figure C.3, and as W5 in Figure C.4, Figure C.5, and Figure C.6. For illustration, Figure C.3 is shown as Figure 5.2. Figure C.5 is shown as Figure 5.3.



**Figure 5.2:** The floor plan of segment I.



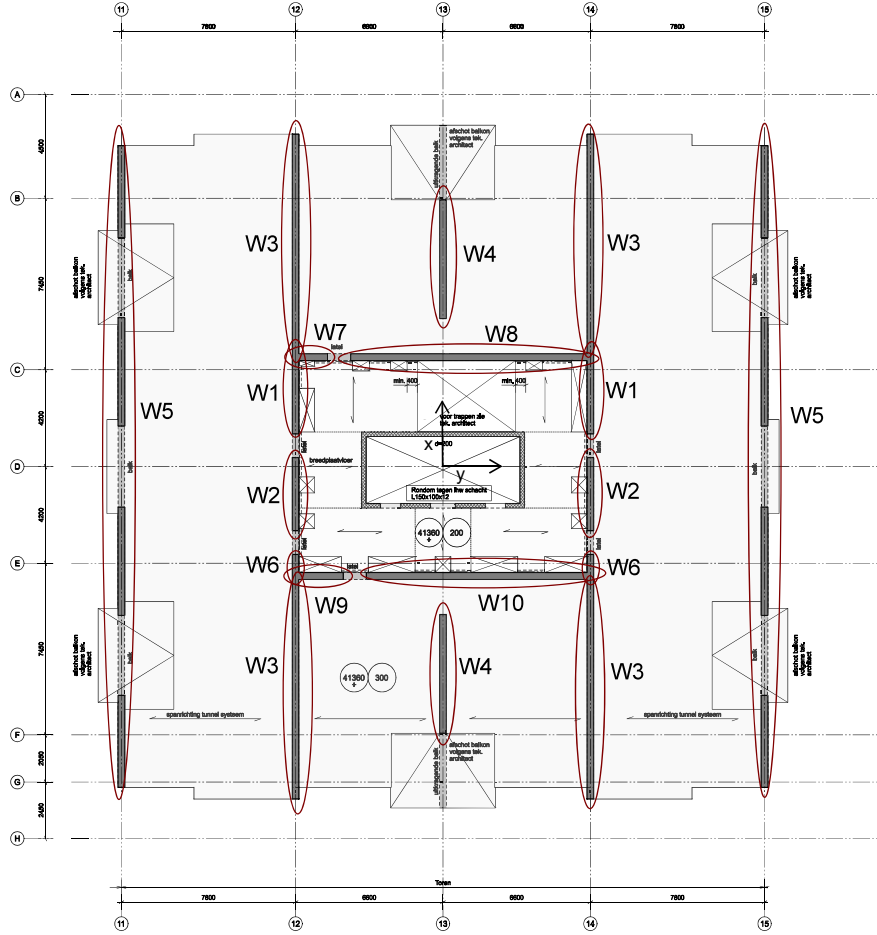


Figure 5.3: The floor plan of segment III.

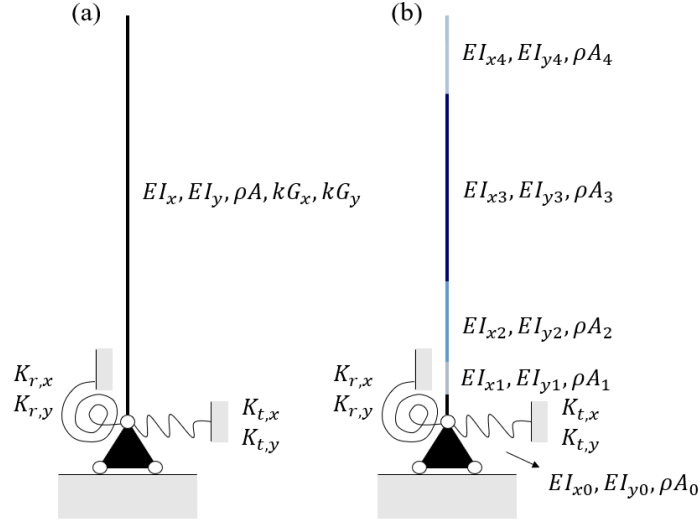
In the Steiner approach, full rigid connections are assumed between the structural elements. Due to the lack of a rigid connection between the core walls and these elements, it was decided not to include them in the calculation. However, despite the absence of a rigid connection, it is reasonable to expect that these elements still contribute partially to the rotational resistance, as they are connected to the core through the floor. The actual contribution of these elements lies somewhere between the extremes of full inclusion and exclusion, but this contribution cannot be calculated using Steiner.

With the FE segment approach, as shown in Table 5.2, the bending stiffnesses  $EI_x$  vary significantly. This is unexpected, as the floor plans between the segments do not change substantially. This approach fails to account for the intermediate relationships between the different segments (0 through IV) of the building. For example, segments 0 and I have column structures (as shown in Figure 5.2 for segment I), which exhibit weaker behavior when considered individually than when integrated into the overall system. This is due to the additional mass from other segments. Therefore, considering these segment individually will lead to an inaccurate assessment of their bending stiffnesses.

Therefore, the bending stiffnesses of the Steiner approach are applied, with a correction factor included to account for the displacement differences between the FE model and five-segment beam using these segment stiffnesses. Since the FE model is stiffer in the y-direction, the segment stiffnesses  $EI_x$  are multiplied by  $\frac{887}{316}$ . Since the FE model is weaker in the x-direction, the segment stiffnesses  $EI_y$  are multiplied by  $\frac{344}{394}$ .

It is assumed that the ratios of the initial structural properties between the segments are well estimated with the Steiner approach. By maintaining these ratios, only one set of parameters of the superstructure needs to be updated, thereby reducing the total number of parameters that needs to be updated.

The uniform Timoshenko beam model and the discrete Euler-Bernoulli beam model used in this study are shown in Figure 5.4. For the discrete Euler-Bernoulli beam, the same initial structural properties are applied as those for the five-segment Timoshenko beam model. For the uniform Timoshenko beam model, the volume weighted averages of these values are used.



**Figure 5.4:** Pictures of (a) the uniform Timoshenko beam model (b) the discrete Euler-Bernoulli beam model.

### 5.1.2. Influence Parameters

The second study investigate the influence of the input parameters of the model. With model updating, a decision must be made about which parameters to update. To understand how each input parameter of the discrete Timoshenko beam influences the modal properties of the model (i.e., natural frequencies and mode shapes), a sensitivity study is conducted where the values of each input parameter are varied within a specified range.

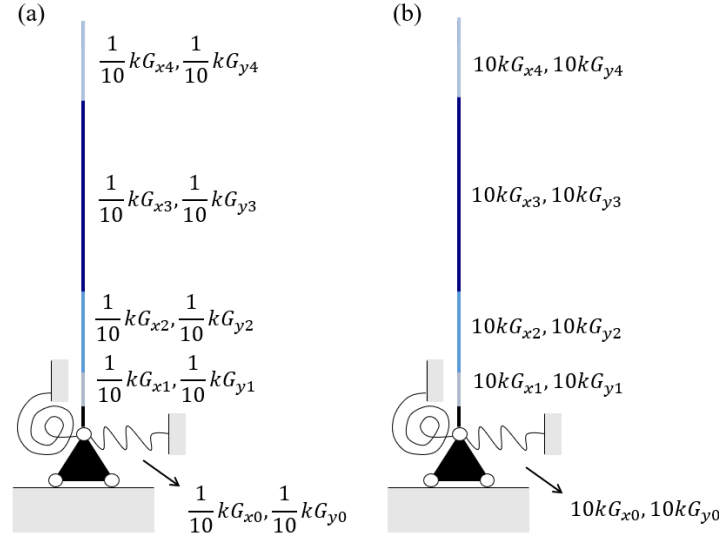
The range over which each input parameter is varied corresponds to its initial structural property value in Table 5.1, scaled by a factor of ten, as shown for the parameter  $kG$  in Figure 5.5. This range is chosen to ensure that a sufficiently broad range is investigated. Parameters  $A$  and  $\rho$ , representing the cross-section and density, respectively, are excluded from this study, as these parameters do not address the model uncertainties.

### 5.1.3. Influence Optimization Algorithm

In the third study, the influence of the optimization algorithm on model updating with a discrete Timoshenko beam model is investigated through a convergence study. The study uses three different algorithms on the same case: Sequential Least Squares Quadratic Programming (SLSQP), Differential Evolution (DE), and Particle Swarm Optimization (PSO). For each algorithm, different amounts of candidate solutions are considered. The updating parameters and the value ranges considered for these updating parameters are based on the study "Influence Parameters" in Section 5.1.2.

To conduct this study, measured modal properties of a building with known structural properties are required. Since such data is not available, as the structural properties of the New Orleans are not known, a numerical case is created using the structural property values from Table 5.1. Random scalars are applied to the initial values in Tables 5.1 in both x- and y-direction, creating another model with known structural and modal properties.

The random scalars applied are listed in Table 5.3. The modal properties of this new model serve as the measured modal properties for the study. The number of modal properties used to fit this model aligns with the amount of measured modal properties of the New Orleans building: two bending modes in the x-direction and three bending modes in the y-direction.



**Figure 5.5:** Pictures (a,b) of parameter  $kG$  scaled by a factor of 10.

The modal displacements considered are corresponding to the measured heights of the New Orleans (51.4 m, 114.6 m and 147.9 m).

It should be noted that the random scaling factors are applied exclusively to the parameters selected for updating. Otherwise, it would be impossible to completely match the model output with the measured modal properties after model updating.

	$K_r$	$K_t$	$E$	$I$	$\rho$	$kG$
<b>Scalar (-)</b>	0.4	9.0	7.0	1.5	2.0	0.2

**Table 5.3:** The random scalar applied on the initial structural properties values of Table 5.1.

#### 5.1.4. Influence Amount of Measured Modal Properties

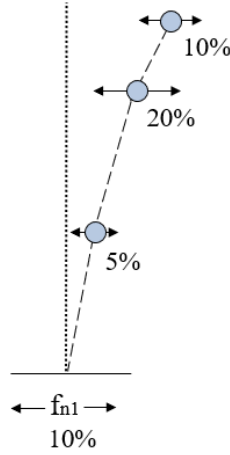
In the fourth study, the influence of the amount of measured modal properties on model updating with the discrete Timoshenko beam model is examined. A convergence study is conducted that considers an increasing number of modes on the same case:

- The first bending mode in the x- and y-direction
- The first two bending modes in the x- and y-direction
- The first two bending modes in the x-direction and the first three bending modes in the y-direction.

Different numbers of candidate solutions are considered. The updating parameters and the value ranges considered for these updating parameters are based on the study "Influence Parameters" in Section 5.1.2. The algorithm applied in the model updating process is determined using the study "Influence Optimization Algorithm" in Section 5.1.3. Since this study also requires measured modal properties of a building with known structural properties, the numerical case from the study "Influence Optimization Algorithm" is also applied here.

#### 5.1.5. Influence Measurements Uncertainties

The fifth study investigates the impact of errors in the measured modal properties (i.e., the measured natural frequencies and mode shapes) on model updating with the discrete Timoshenko beam model. The sensitivity study introduces random errors into the measured modal properties to understand how these uncertainties affect the results obtained. The errors applied to the natural frequencies range up to 10%, and the errors applied to the mode shapes range up to 20%. An example of an error distributions is illustrated in Figure 5.6 for the first natural frequency and mode shape.



**Figure 5.6:** A visualization of the random errors applied on the first natural frequency and mode shape.

The updating parameters and the value ranges considered for these updating parameters are based on the study "Influence Parameters" in Section 5.1.2. The algorithm applied in the model updating process is determined using the study "Influence Optimization Algorithm" in Section 5.1.3. The amount of candidate solutions considered is chosen based on the study "Influence Amount of Measured Modal Properties" in Section 5.1.4.

Since this study also requires measured modal properties of a building with known structural properties, the numerical case from the study "Influence Optimization Algorithm" is applied here as well.

### 5.1.6. Estimating Structural Properties

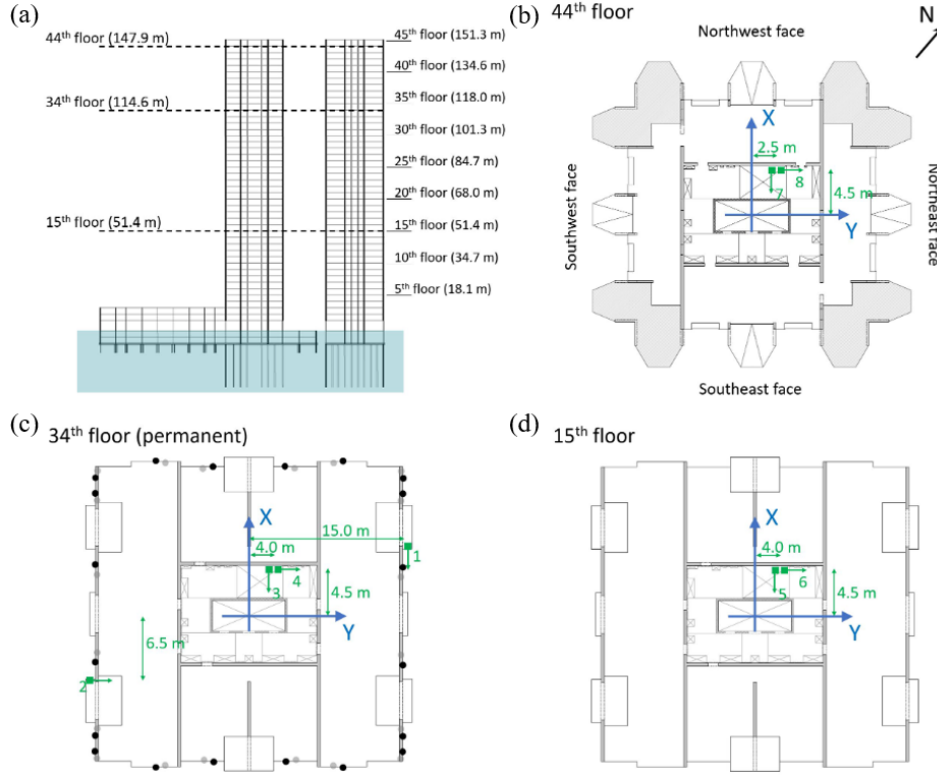
This study applies model updating on the New Orleans to estimate its structural properties using the discrete Timoshenko beam model and the measured modal properties. The measured modal properties of the New Orleans are indicated in Table 5.4.

Mode	1	2	3	4	5
Dominant direction	X	Y	Y	X	Y
Natural frequency [Hz]	0.282	0.291	1.332	1.527	2.771
<b>Modal Displacement</b>					
Height 1 (51.4 m)	-0.30	-0.29	-0.90	0.89	-0.73
Height 2 (114.6 m)	-0.71	-0.78	0.19	-0.23	0.83
Height 3 (147.9 m)	-1.00	-1.00	1.00	-1.00	-1.00

**Table 5.4:** The measured natural frequencies and modal displacements of the residential tower New Orleans.

To determine these modal properties, acceleration sensors were installed on the 15th, 34th, and 44th floors. The positions and orientations of the sensors are illustrated in Figure 5.7. Details of the data analysis applied to obtain the measured modal properties are described in Appendix D. The updating parameters and their value ranges considered are based on the study "Influence Parameters" in Section 5.1.2. The algorithm applied in the model updating process is determined using the study "Influence Optimization Algorithm" in Section 5.1.3. The amount of measured properties from Table 5.4 used to fit the model and the amount of candidate solution considered are based on the study "Influence of the Amount of Measured Properties" in Section 5.1.4.

The model updating process follows the approach described by Moretti et al. [1] and consists of two parts. In the first part, model updating is carried out using the settings specified above. However, unlike the method used by Moretti et al. [1], which retains the lowest quantile of the candidate solutions (i.e., the quantile that best fit the measured modal properties after updating), this study retains the



**Figure 5.7:** An schematic (a) of the New Orleans and the positions of the acceleration sensors (green squares) on (b) the 44th floor, (c) the 34th floor, and (d) the 15th floor. The green arrows indicate the directions of the acceleration sensors [17, 12].

lowest octile of the candidate solutions. This modification is done because the discrete Timoshenko beam model is expected to be more complex than the uniform Euler-Bernoulli beam model, resulting in a more complex solution space for identifying the global minimum. New structural property value ranges for the updating parameters are defined based on the minimum and maximum values of these retained solutions. Using these newly refined ranges, another selection and model updating process is carried out (the second part) in the same manner as the first part. The reasons behind the choices made in this approach by Moretti et al. [1] are unclear. This approach was chosen for this research to ensure a fair comparison between the findings of the study conducted by Moretti et al. [1] and this research.

## 5.2. Results

In this section, the results of the research conducted are presented and discussed. Sections 5.2.1 to 5.2.5 present the results of the studies where the influence of various settings and uncertainties in model updating with a discrete Timoshenko beam model are investigated. Section 5.2.6 show the results of applying model updating to estimate the structural properties of the New Orleans.

### 5.2.1. Influence Features Model

To begin, the results of the first study are presented. The obtained natural frequencies using different beam models are shown in Tables 5.5 and 5.6. Their mode shapes are illustrated in Figures 5.8 and 5.9. The modal properties of the Finite Element (FE) model and the measured modal properties are also shown as reference.

Figures 5.8 and 5.9 show that the mode shapes of the FE model for the two lowest bending modes in both directions closely resemble the measured ones, indicating a good representation of reality. However, the mode shape of the third bending mode does not align closely with the measured third bending mode. The FE model remains a simplified beam model. Factors such as simplifications in boundary conditions, insufficient mesh refinement, or modeling assumptions can cause the differences observed.

Tables 5.5 and 5.6 show that the natural frequencies of the FE model tend to differ more from the measured natural frequencies than those of the discrete Timoshenko beam model. This is unexpected, as the FE model incorporates more realistic boundary conditions and better captures complex geometries compared to the discrete Timoshenko model. However, it is unlikely that the discrete Timoshenko beam model provides a more accurate approximation of the high-rise building than the FE model. The natural frequency of the third bending mode and the mode shapes of the higher bending modes in the y-direction of this beam model exhibit significant discrepancies from the measured data.

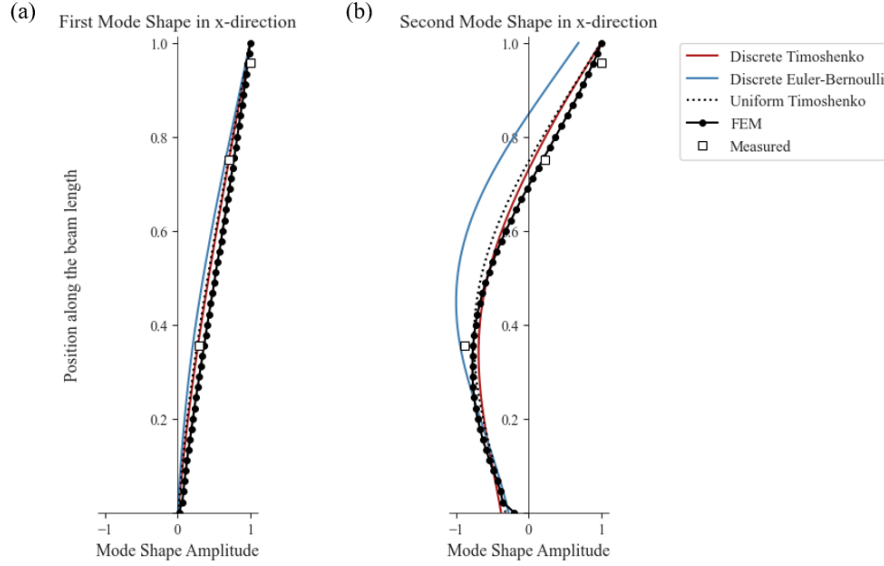
The discrete Timoshenko beam model and the Euler-Bernoulli beam model show similarities in their natural frequencies, indicating that shear effects have minimal impact on the overall behavior of the structure. However, Figure 5.8 shows that shear effects are significant in describing the mode shape of the second bending mode in the x-direction. The uniform Timoshenko beam model shows comparable results with the discrete Timoshenko beam model. This suggest that the initial values of the parameters of the different segments are similar. For the second and third bending mode in the y-direction, the FE model exhibits greater translation at the lower boundary compared to the beam models. Further investigation is required to identify the cause of this discrepancy.

Measured [Hz]		FEM ( $\Delta_{\text{measured}}$ %) [Hz]		Discrete Timoshenko beam model ( $\Delta_{\text{measured}}$ %) [Hz]	
X	Y	X	Y	X	Y
0.28	0.29	0.23 (−17.9)	0.20 (−31.0)	0.25 (−10.7)	0.23 (−20.7)
1.53	1.33	1.25 (−18.3)	1.02 (−23.3)	1.48 (−3.3)	1.43 (+7.5)
	2.77		2.14 (−22.7)		3.78 (+36.5)

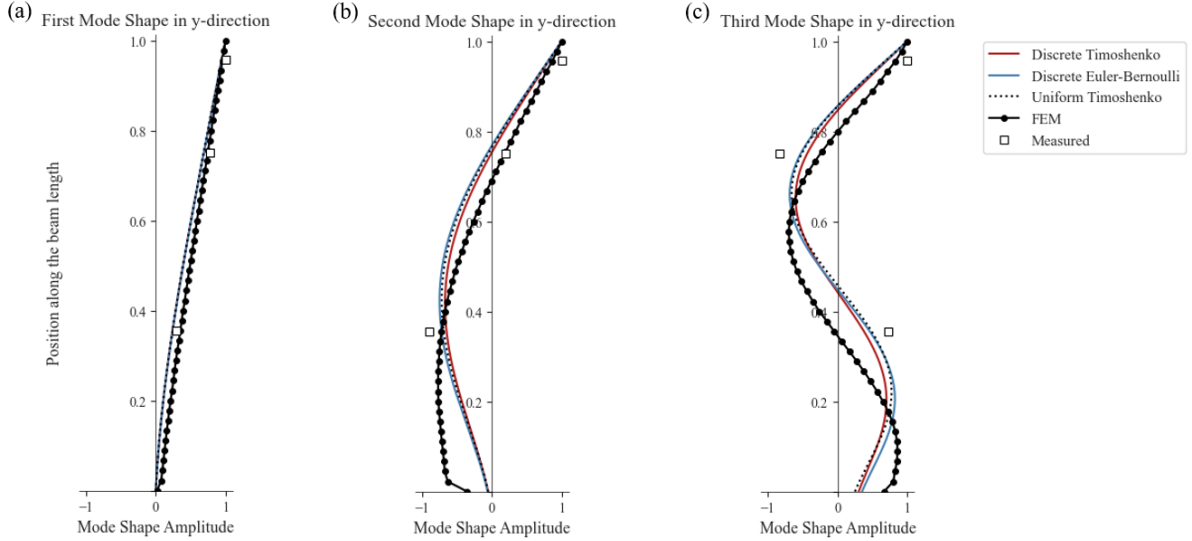
**Table 5.5:** The measured natural frequencies, the natural frequencies of the FEM, and the natural frequencies of the discrete Timoshenko beam model.

Discrete Euler-Bernoulli beam model ( $\Delta_{\text{measured}}$ %) [Hz]		Uniform Timoshenko beam model ( $\Delta_{\text{measured}}$ %) [Hz]	
X	Y	X	Y
0.22 (−21.4)	0.21 (−27.6)	0.21 (−25.0)	0.19 (−34.5)
1.49 (−2.6)	1.38 (+3.8)	1.30 (−15.0)	1.26 (−5.3)
	3.75 (+35.4)		3.48 (+25.6)

**Table 5.6:** The natural frequencies of the discrete Euler-Bernoulli beam model and the uniform Timoshenko beam model.



**Figure 5.8:** The measured and model mode shapes in x-direction



**Figure 5.9:** The measured and model mode shapes in y-direction

### 5.2.2. Influence Parameters

This section presents the results of second study, which changed the input parameter values of the discrete Timoshenko beam model to see their influence on the modal properties. The results of parameters  $K_r$  and  $K_t$  in the y-direction are shown in Figures 5.10 and 5.11. The results of parameters  $E$  and  $I$  in the y-direction are shown in Figures 5.13 and 5.12. The result of parameter  $kG$  in the y-direction is shown in Figure 5.14. The change in the mode shape is quantified using the MAC (Equation 2.2).

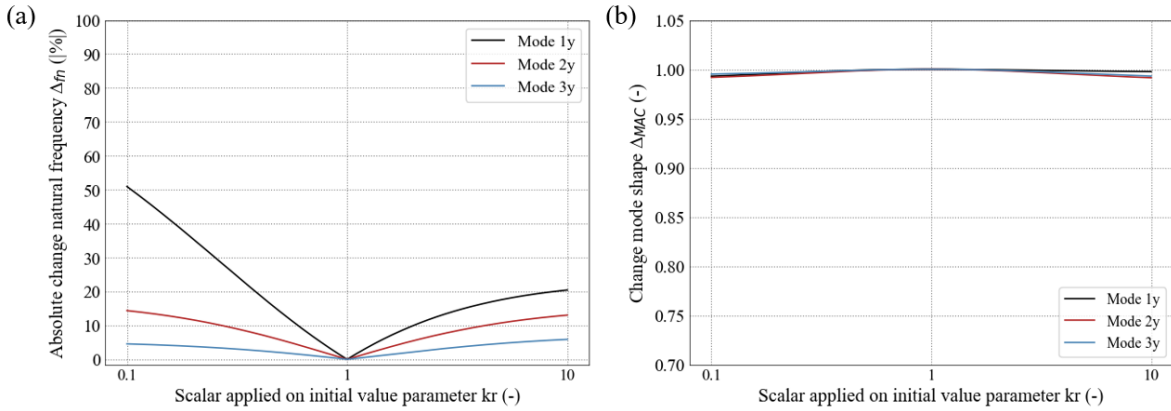
According to Figure 5.10, adjusting the value of parameter  $K_r$  primarily influence the first natural frequency. Changes in  $K_r$  do not affect the mode shapes. In contrast, Figure 5.11 shows that parameter  $K_t$  only affects the higher bending modes.

Parameters  $E$  and  $I$  have the same influence on the bending modes. These parameters often occupy similar positions within the model, leading to comparable influences on the natural frequencies and mode shapes. Due to these similarities between parameters  $E$  and  $I$ , it is chosen to consider the up-

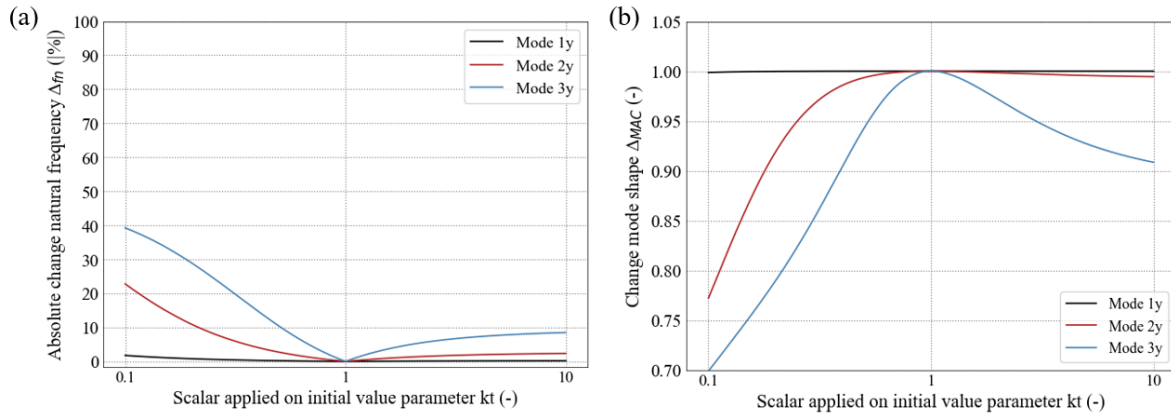
dating results of these parameters together as  $EI$  for the rest of the research. Figure 5.14 shows that parameter  $kG$  does not affect the model output. This is in line with the results obtained in Table 5.5 in Section 5.2.1, which indicated that shear effects have minimal impact on the overall behavior of the New Orleans. Therefore,  $kG$  is not selected as an updating parameter.

The results in the x-direction are comparable to the results found in the y-direction. Therefore, only the results in y-direction are shown. The results in x-direction are shown in Appendix E. Based on the results, it is chosen to update parameters  $K_r$ ,  $K_t$ ,  $E$ , and  $I$  in the following studies of the research.

To summarize, it should be noted that parameter  $K_r$  primarily influence the first natural frequency, and parameter  $K_t$  affects the higher bending modes. Parameters  $E$  and  $I$  exhibit similar influences, as the parameters often occupy similar positions within the model. Therefore, the results for these parameters will be considered together as  $EI$  for the rest of the research. Parameter  $kG$  is not selected, as the modal properties of the model are insensitive to this parameter.

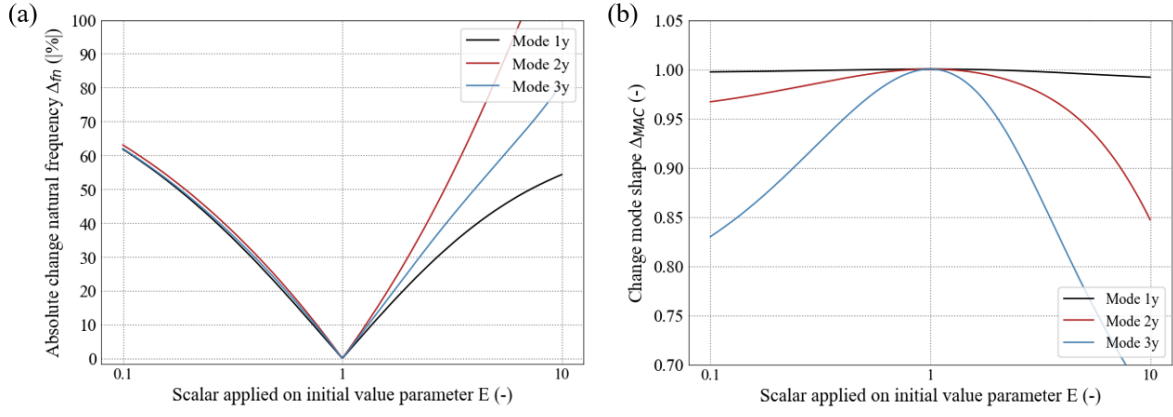


**Figure 5.10:** The influence of parameter  $K_r$  on the model output in y-direction.

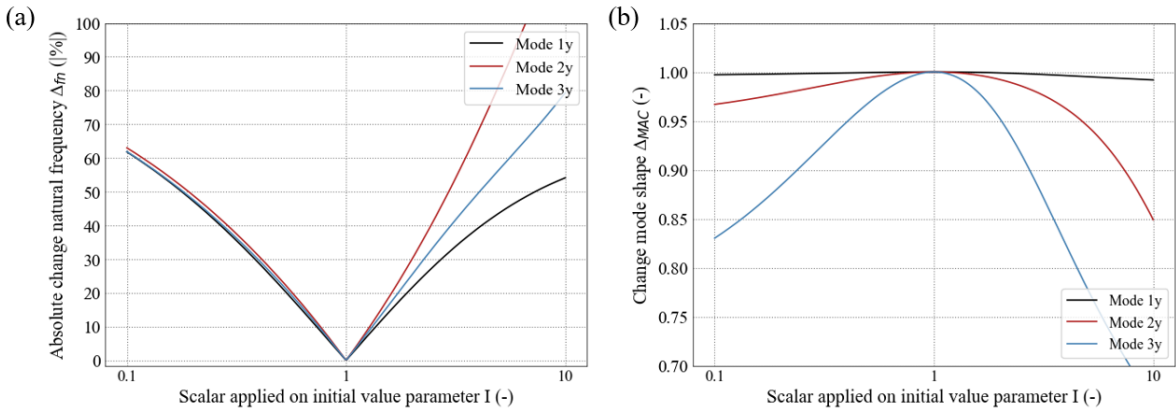


**Figure 5.11:** The influence of parameter  $K_t$  on the model output (a,b) in y-direction.

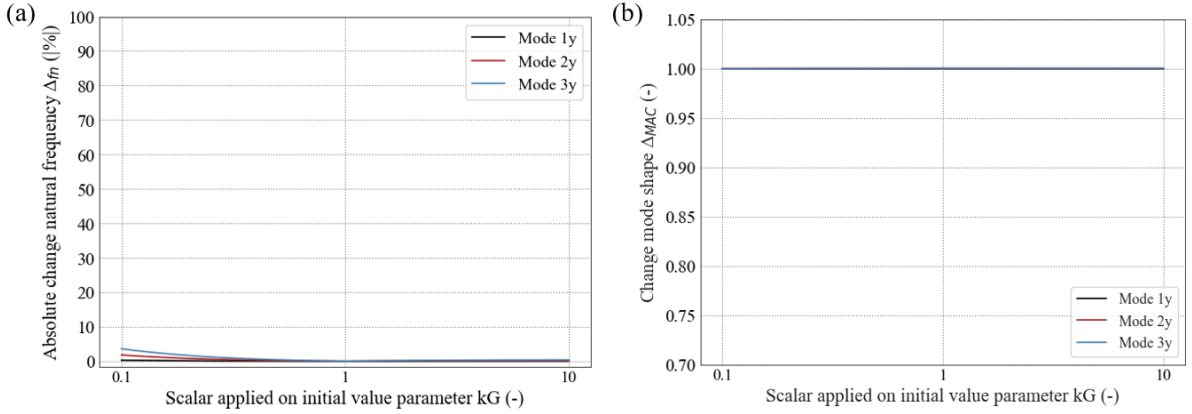




**Figure 5.12:** The influence of parameter  $E$  on the model output (a,b) in y-direction.



**Figure 5.13:** The influence of parameter  $I$  on the model output (a,b) in y-direction.



**Figure 5.14:** The influence of parameter  $kG$  on the model output (a,b) in y-direction.

### 5.2.3. Optimization Algorithm

In this section, the results of the third study are shown, which applied different optimization algorithms on the same numerical case. The updating parameters were  $K_r$ ,  $K_t$ ,  $E$  and  $I$ , which were updated at the same time.

To create this numerical case, random scalars were applied to the initial values of the chosen updating parameters, indicated in Table 5.1, to generate another model with known structural properties and

mode shapes. The scalars applied are indicated in Table 5.7. The modal properties of this new model, indicated in Table 5.8, were selected as measured modal properties for this study.

	$K_r$	$K_t$	$E$	$I$
<b>Scalar (-)</b>	0.4	9.0	7.0	1.5

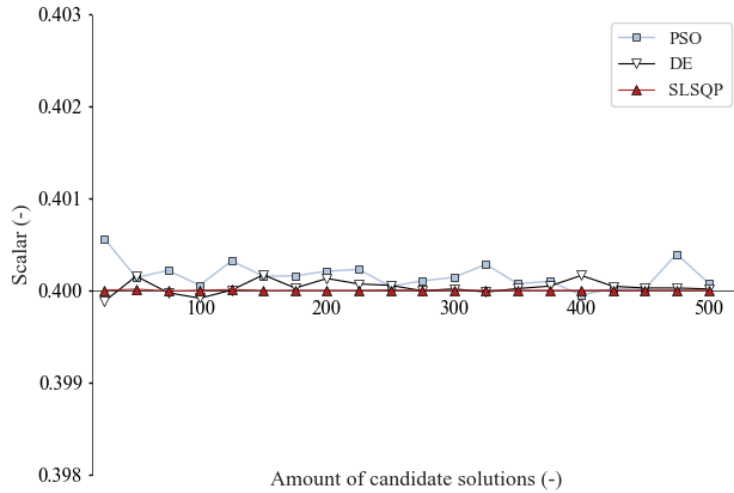
**Table 5.7:** The random scalar applied on the initial structural properties values of Table 5.1.

<b>Mode</b>	<b>1</b>	<b>2</b>	<b>3</b>	<b>4</b>	<b>5</b>
Dominant direction	X	Y	Y	X	Y
Natural frequency [Hz]	0.226	0.235	3.83	4.345	11.132
<b>Modal Displacement</b>					
Height 1 (51.4 m)	0.36	0.36	-0.75	-0.78	0.39
Height 2 (114.6 m)	0.78	0.78	0.10	0.13	-0.57
Height 3 (147.9 m)	1.00	1.00	1.00	1.00	1.00

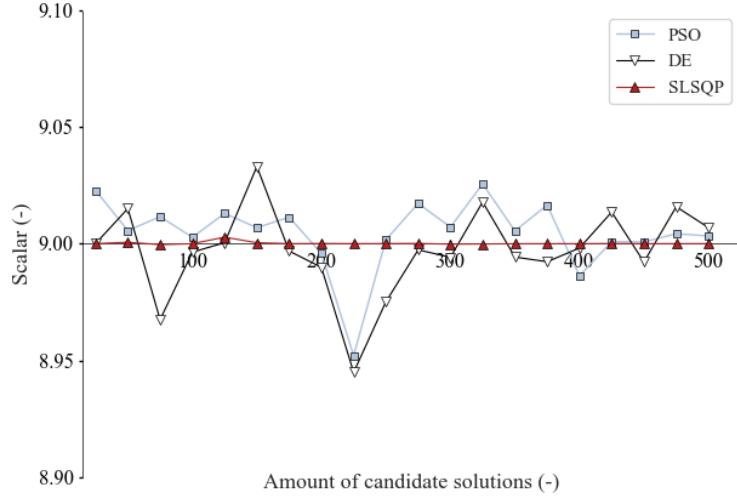
**Table 5.8:** The modal properties of the numerical case.

Different amounts of candidate solutions were considered, and for each amount, the candidate solution that best fitted the measured modal properties after model updating is shown. To maintain the compactness of the results, the results are presented as ratios relative to their initial values in Table 5.1. For example, a value of 0.8 indicates that after the model updating, the new value of the structural property is 80% of its initial value. Therefore, if the model updating was successful with the chosen algorithm, the results should match the scalars listed in Table 5.7. The results of parameters  $K_r$  and  $K_t$  are shown in Figures 5.15 and 5.16. The results of parameters  $E$  and  $I$  are shown together as  $EI$  in Figure 5.17.

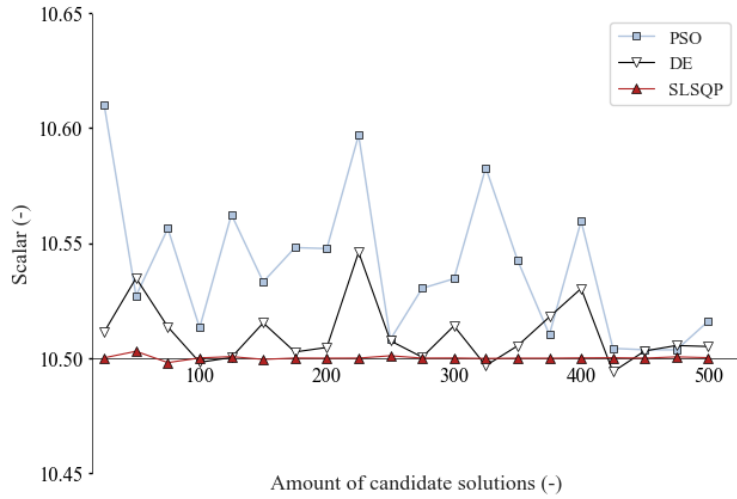
According to the Figures,  $K_r$ ,  $K_t$  and  $EI$  are well estimated by all algorithms. The differences between the algorithms are small. The SLSQP algorithm achieved the most accurate estimation by all parameters. Therefore, for the following studies, the SLSQP algorithm is selected for model updating.



**Figure 5.15:** The influence of the different algorithms on the model updating result of parameter  $K_r$ .



**Figure 5.16:** The influence of the different algorithms on the model updating result of parameter  $K_t$ .



**Figure 5.17:** The influence of the different algorithms on the model updating result of the product  $EI$ .

#### 5.2.4. Influence Amount of Measured Modal Properties

This section presents the model updating results of the fourth study, which applied different amount of measured modal properties on the same numerical case. The parameters chosen to update in this study were  $K_r$ ,  $K_t$ ,  $E$  and  $I$ , which were updated at the same time. The SLSQP was chosen as optimization algorithm.

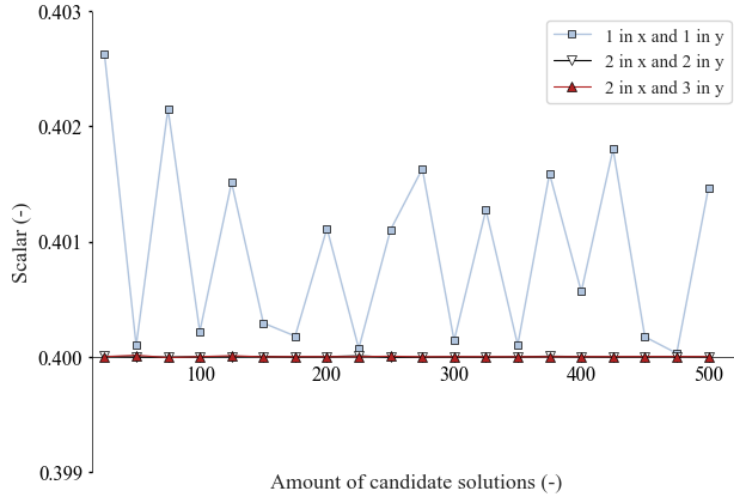
This study used the same numerical case described in the study "Influence Optimization Algorithm" in Section 5.1.3. Per amount of measured modal properties, different amounts of candidate solutions were considered, and for each amount of candidate solutions, only the candidate solution that best fitted the measured modal properties after updating is shown.

The results are presented as ratios relative to their initial values in Table 5.1. Therefore, if model updating was successful with the amount of measured modal properties, the results should match the scalars listed in Table 5.7. The results of the parameters  $K_r$  and  $K_t$  are shown in Figures 5.18 and 5.19. The result of  $EI$  is shown in Figure 5.20.

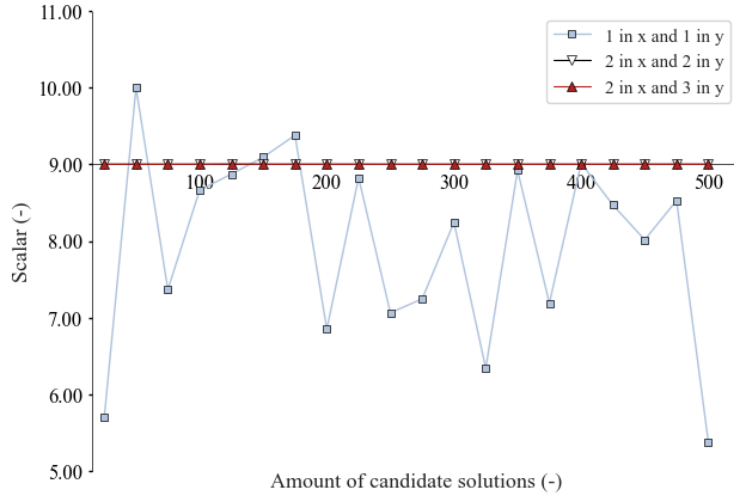
Figure 5.18 shows that it is sufficient to estimate parameter  $K_r$  by considering only the lowest bending mode in both the x- and y-directions. This is in line with the influence observed in Figure 5.10 in Section

5.2.2, which indicated that this parameter mainly affects the first natural frequency. However, Figures 5.19 and 5.20 indicate that considering only the lowest bending mode in both the x- and y-directions is insufficient to estimate  $K_t$  and  $EI$ . For parameter  $K_t$ , this is in line with Figure 5.11 in Section 5.2.2, which shows that parameter  $K_t$  has a insensitivity to the first bending mode. However, Figures 5.12 and 5.13 shows that parameters  $E$  and  $I$  do not exhibit insensitivity to the first bending mode. It is suspected that the first bending mode alone does not provide enough information to estimate parameters  $E$  and  $I$ . Higher bending modes, which are more sensitive to the bending stiffness, are needed, as they provide more comprehensive information.

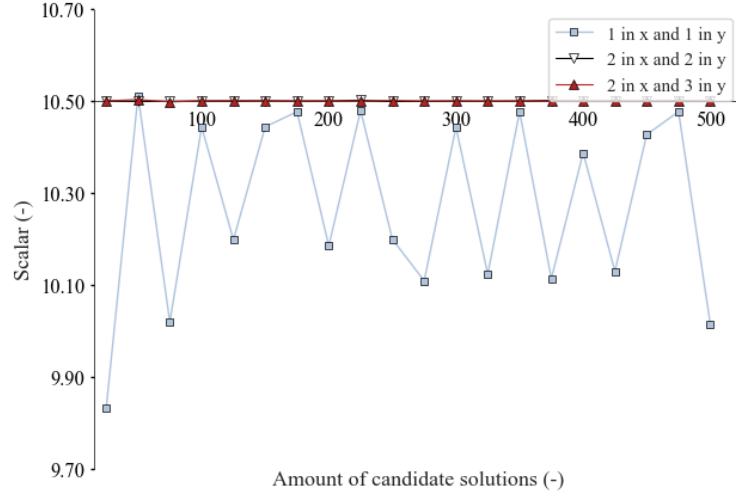
The difference in the results  $K_r$ ,  $K_t$ ,  $EI$  when considering two or three bending modes in y-direction is small. Therefore, both scenarios are considered in the study "Estimating Structural Properties", using 400 candidate solutions. For the study "Influence Measurement Uncertainties", 200 candidate solutions are considered.



**Figure 5.18:** The influence of the amount of modal information on the model updating result of parameter  $K_r$ .



**Figure 5.19:** The influence of the amount of modal information on the model updating result of parameter  $K_t$ .

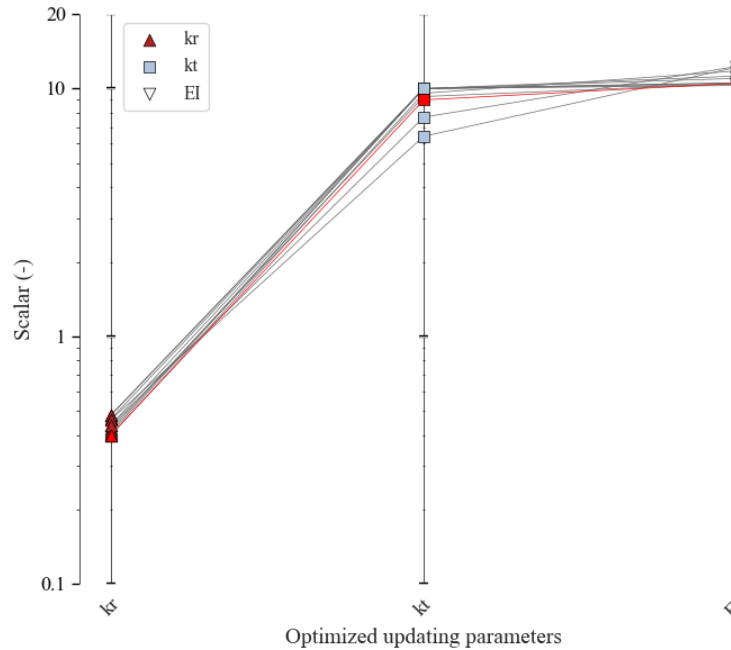


**Figure 5.20:** The influence of the amount of modal information on the model updating result of  $EI$ .

### 5.2.5. Measurement Uncertainties

This section shows the results of the fifth study, which applied errors to the measured modal properties to see their influence on the results obtained. The parameters updated in this study were  $K_r$ ,  $K_t$ ,  $E$  and  $I$ . These were updated at the same time, using the SLSQP optimization algorithm. The study used the same numerical case described in the study "Influence Optimization Algorithm" in Section 5.1.3.

Two hundred candidate solution were considered, and only the candidate solution with the lowest objective function value (i.e., the solution that best fitted the measured modal properties after updating) is shown. As adding random errors to the measured modal properties and conducting model updating was repeated ten times, ten solution are shown in total. The results are presented as ratios relative to the initial values shown in Table 5.1. Therefore, if the model updating was insensitive to the errors applied, the results should match the scalars listed in Table 5.7.



**Figure 5.21:** The influence of measurement uncertainties on the model updating results.

The results of  $K_r$ ,  $K_t$ ,  $EI$  are shown in Figure 5.21. The values of the different parameters that belong together are connected with grey lines. The red solution indicates the scalars of Table 5.7.

Parameter  $K_r$  shows limited sensitivity to the uncertainties in the measurements, with values obtained between 0.40 and 0.47. Parameters  $K_t$  and  $EI$ , however, show significant sensitivity to the uncertainties. The values of parameter  $K_t$  range between 6.41 and 10.0.  $EI$  shows values ranging between 10.29 and 12.28.

### 5.2.6. Estimating Structural Properties

To conclude, the results using the measured properties of the New Orleans are shown. The parameters updated in this study were  $K_r$ ,  $K_t$ ,  $E$  and  $I$ , using the SLSQP algorithm. Four hundred candidate solutions were considered, and the lowest octile of these solutions, i.e., the octile with the smallest objective function values, were retained.

Table 5.9 presents the estimated and measured natural frequencies and the Modal Assurance Criterion (MAC) values between the estimated and measured mode shapes. Case 1 indicates the results of the model updating using the two lowest bending modes in both x- and y-directions, and Case 2 indicates the results of the model updating using two bending modes in the x-direction and three bending modes in the y-direction.

Mode [-]	Natural frequency [Hz]				MAC [-]	
	Design	Measured	Case 1 ( $\Delta_{\text{measured}}$ %)	Case 2 ( $\Delta_{\text{measured}}$ %)	Measured vs. Case 1	Measured vs. Case 2
1 (x)	0.200	0.282	0.282 (0%)	0.282 (0%)	0.998	0.998
2 (x)	1.020	1.527	1.527 (0%)	1.527 (0%)	0.999	0.999
1 (y)	0.230	0.291	0.291 (0%)	0.241 (-17.2%)	0.999	0.995
2 (y)	2.140	1.332	1.332 (0%)	1.332 (0%)	1.000	0.963
3 (y)	2.440	2.771	-	2.773 (+0.1%)	-	0.872

**Table 5.9:** The measured and estimated natural frequencies and MAC values between the measured and estimated mode shapes.

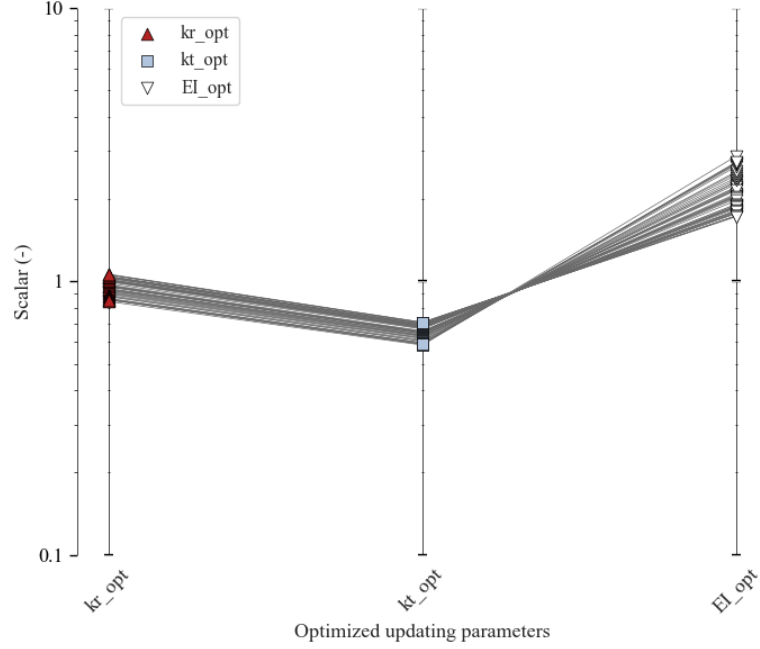
In Case 1, the median natural frequencies closely match the measured natural frequencies. However, in Case 2, there are differences in the estimated and measured first natural frequency. Adjustments to the values of the structural properties to better match the third bending mode negatively affected the estimation of the first natural frequency. Moreover, the MAC values in Case 2 indicate that the measured third bending mode shape is not accurately described. Despite incorporating additional features compared to the model used in the study by Moretti et al. [1], the discrete Timoshenko beam model is still insufficient for accurately describing the third bending mode. Therefore, the obtained structural properties values of Case 2 are not considered.

Figures 5.22 and 5.23 illustrate the estimated structural property values in the x- and y-directions obtained through model updating for Case 1. The results are presented as ratios relative to the initial values shown in Table 5.1. The values of the different parameters that belong together are connected with grey lines. For the estimates per parameter, the Coefficient of Variation (CV) is calculated. The CV shows the relative variability of the estimates:

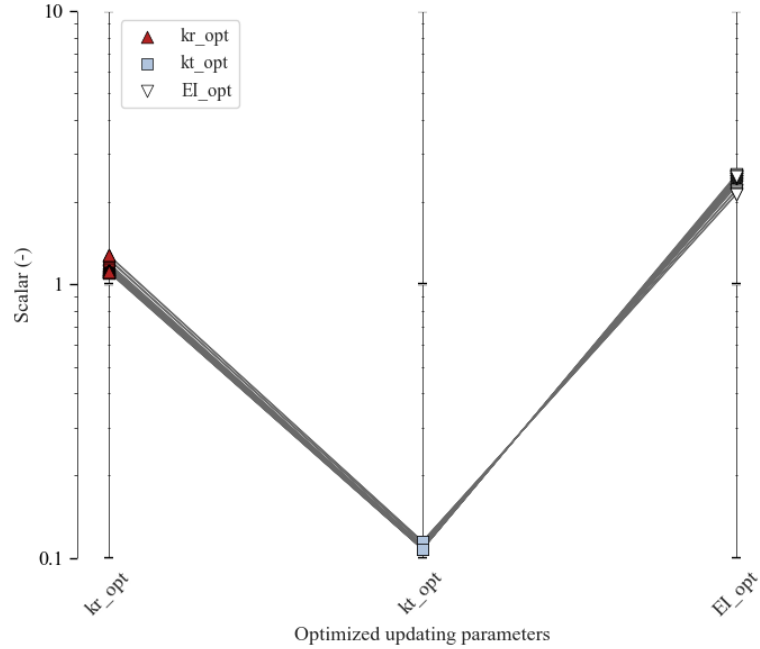
$$CV = \left( \frac{\sigma}{\mu} \right) \times 100\% \quad (5.3)$$

where  $\sigma$  is the standard deviation and  $\mu$  is the mean. A lower CV indicates less variability relative to the mean, suggesting more consistency in the estimates. A higher CV indicates greater dispersion in the estimates. The distributions of  $K_{r,x}$  and  $K_{r,y}$  exhibit low Coefficients of Variation (CVs), of 4.54% and 6.91%, respectively, suggesting that the parameters should remain around their initial values. This is an improvement compared to the study by Moretti et al. [1], as in that study, the estimations of the structural property  $K_{r,y}$  showed high uncertainty (a Coefficient of Variation of 49.6%) after model

updating. Similarly, the distributions of  $K_{t,x}$  and  $K_{t,y}$  exhibit low CVs of 2.27% and 5.87%, respectively. These distributions suggest that their values should be lower than the design values, although there is a noticeable magnitude difference in these estimates. The distribution of  $EI_x$  and  $EI_y$  show also low uncertainties, indicated with Coefficients of Variation (CVs) of 5.06% and 15.45% respectively, suggesting that the bending stiffness should be higher. Based on these results, it can be stated that estimating the structural parameters of the New Orleans using vibration-based model updating was effective.



**Figure 5.22:** The estimated parameter values of Case 1 in x-direction.



**Figure 5.23:** The estimated parameter values of Case 1 in y-direction.

### 5.3. Main Findings

With the research, several studies were conducted to evaluate the influence of certain settings and uncertainties in model updating and to assess the effectiveness of the technique in estimating the structural properties of a bending-dominant building. The main findings were:

- **Setup Models:** Obtaining values for the bending stiffness of the segments using the FE model proved to be difficult. The Steiner approach, which calculated the bending stiffness per segment using the elastic modulus  $E$  and the second moment of inertia  $I$  (calculated with  $\frac{1}{12}bh^3$  and the Steiner rule), assumed fully rigid connections between elements when determining  $I$ , making it not possible to accurately account for the contributions of elements that were not rigidly connected. The FE approach, which used the relative displacement  $\Delta$  of the FE segment to calculate the bending stiffness, did not account for the intermediate relationships between segments. Segments 0 and I had column structures, which exhibited weaker behavior when considered individually compared to when they were part of the overall system. This was due to the additional mass of the other segments. Therefore, considering these segments individually led to an inaccurate assessment of their bending stiffnesses.
- **Influence of Parameters:** Parameter  $K_r$  showed primarily influence on the first natural frequency, and parameter  $K_t$  showed primarily influence on the higher bending modes. Parameters  $E$  and  $I$  exhibited similar influences. Parameter  $kG$  showed no influence on the modal properties.
- **Influence of Amount of Measured Modal Properties:** To estimate parameters  $K_t$  and  $EI$ , it was insufficient to use only one bending mode in both directions.
- **Influence of Measurement Uncertainties:** Parameter  $K_r$  showed limited sensitivity to the uncertainties in the measurements, with values obtained between 0.40 and 0.47, with a target value of 0.40. In contrast, parameters  $K_t$  and  $EI$  exhibited significant sensitivity to uncertainties. The values of parameter  $K_t$  ranged between 6.41 and 10.0, with a target value of 9.0. For  $EI$ , the values ranged between 10.29 and 12.28, with a target value of 10.5.
- **Estimating Structural Properties:** Despite incorporating additional features compared to the model used in the study by Moretti et al. [1], the discrete Timoshenko beam model was not sufficient for accurately describing the third bending mode. This was indicated by a MAC value of 0.87.

However, when only the lowest two bending modes were used to fit the model, the estimates of parameter  $K_{r,y}$  showed low uncertainty, with a Coefficient of Variation of 6.91%. This was an improvement compared to the study conducted by Moretti et al. [1], which obtained a Coefficient of Variation of 46.9% with a uniform Euler-Bernoulli beam model. Based on these results, it could be stated that estimating the structural parameters of the New Orleans using vibration-based model updating was effective.



# 6

## The office tower the Delftse Poort

To evaluate the effectiveness of vibration-based model updating in estimating the structural properties of a shear-dominant high-rise building, which has irregular stiffness across its heights, using a discrete Timoshenko beam model, this chapter applied the technique to the office tower the Delftse Poort in Rotterdam.

Section 6.1 explains the studies conducted on the office tower the Delftse Poort. Section 6.2 shows the results of these studies. Finally, the main findings of the research are summarized in Section 6.3.

### 6.1. Studies

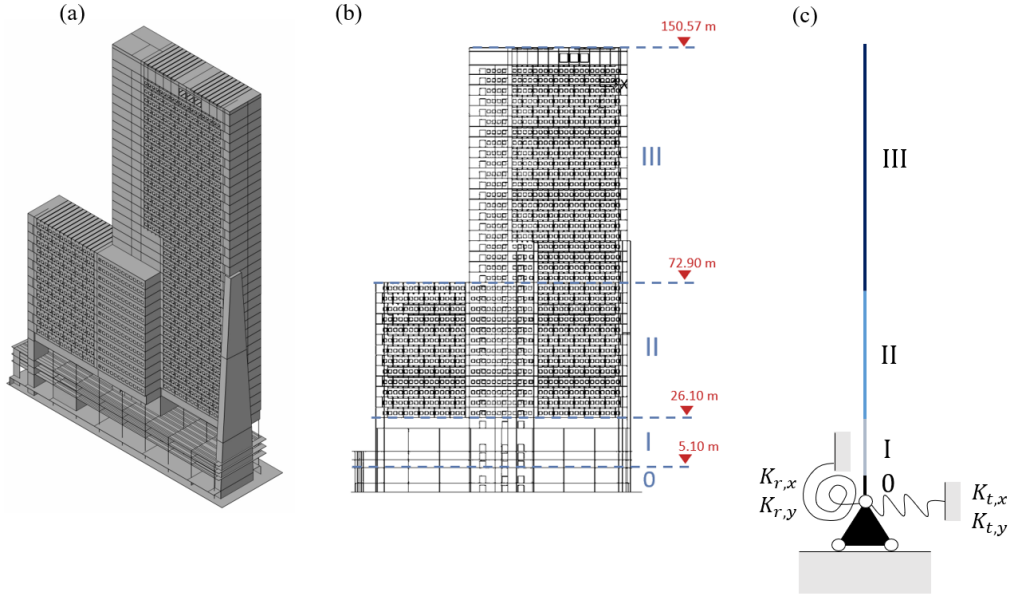
The research consists of six studies. The first five studies examine how different settings and uncertainties in the model updating process, using a discrete Timoshenko beam model, influence the results obtained. The final study applies model updating to estimate the structural properties of the Delftse Poort, using the discrete Timoshenko beam model and the measured modal properties. The details of the studies are explained in Sections 6.1.1 to 6.1.6. Appendix D describes how the measured modal properties were obtained.

#### 6.1.1. Influence Features Model

The first study is about the features of the Timoshenko beam model. The model incorporates shear deformations and accounts for irregular stiffness across its height. To assess the influence of these features in the beam model, the natural frequencies and mode shapes obtained with this model are compared with those obtained using the uniform Timoshenko beam and discrete Euler-Bernoulli beam model.

The discrete Timoshenko beam model used in the studies to approximate the dynamic behavior of the Delftse Poort is created using the Finite Element (FE) model of the building. This FE model, shown in Figure 6.1, is divided into four segments. The segments were defined based on the floorplans. The floors within a segment have the same floorplan in the FE model. Therefore, a four-segment Timoshenko beam model is used to represent the FE model in Figure 6.1.

The initial structural property values applied in this beam model are shown in Table 6.1. These values were obtained using the information of the FE model and the floor plans. Details of the approach with which the structural property values were obtained are described in Appendix C. The second moment of area around the x-axis is denoted as  $I_x$ . The rotational stiffness of the foundation around the x-axis is denoted as  $K_{r,x}$ . The translation stiffness of the foundation in x-direction is denoted as  $K_{t,x}$ .



**Figure 6.1:** Pictures of (a) the FE model of the Delftse Poort, (b) the 4 segments defined for the Delftse Poort model, and (c) the four-segment Timoshenko beam model.

Segment	0	I	II	III
$L$ [m]	3.15	22.65	46.8	81.27
$A$ [m <sup>2</sup> ]	1320	1320	1320	825
$I_x$ [m <sup>4</sup> ]	4312	4312	4744	1402
$I_y$ [m <sup>4</sup> ]	34350	34350	35105	10685
$\rho$ [kg/m <sup>3</sup> ]	5071	552	426	371
$E$ [N/m <sup>2</sup> ]	11e9	11e9	22e9	33e9
$\nu$ [-]	0.2	0.2	0.2	0.2
$G$ [N/m <sup>2</sup> ]	4.6e9	4.6e9	9.2e9	1.38e9
$k$ [-]	0.85	0.85	0.85	0.85
<b>Boundary</b>				
$K_{r,x}$ [GNm/rad]	6486	$K_{t,x}$ [GN/m]	9.22	
$K_{r,y}$ [GNm/rad]	50371	$K_{t,y}$ [GN/m]	9.22	

**Table 6.1:** The structural property values obtained with the floor plans and FE model of the Delftse Poort.

Obtaining these bending stiffnesses of the segments was challenging. Two approaches were applied to determine the bending stiffness  $EI_s$  (with  $s$  being  $x$  or  $y$ ) per segment. The first approach calculated the bending stiffness per segment  $EI_s$  using the elastic modulus  $E$  applied in the FE model and the second moment of area  $I$ , calculated using the term  $\frac{1}{12}bh^3$  and the Steiner rule:

$$EI_s = E \left( \frac{1}{12}bh^3 + bd^2 \right) \quad (6.1)$$

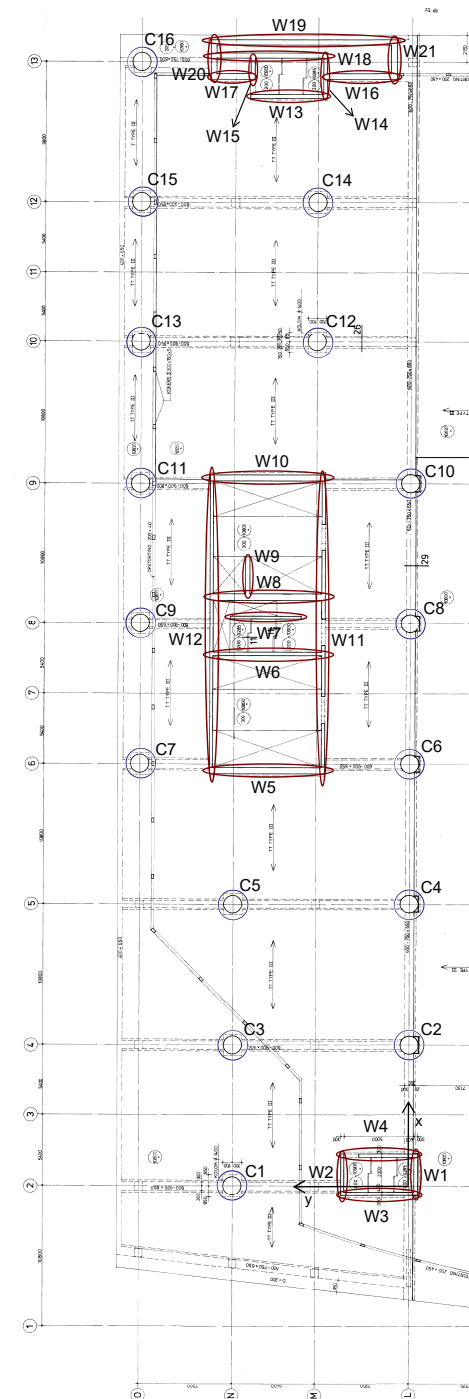
The second approach considered the segments of the FE model. The FE model of the segment was clamped at the bottom, and a force  $F$  was applied at the top, resulting in a relative displacement  $\Delta$ . The bending stiffness  $EI_s$  was then determined using this displacement  $\Delta$  and Equation 6.2:

$$EI_s = \frac{FL^3}{3\Delta} \quad (6.2)$$

More information about the applied approaches can be found in Appendix C. The values obtained from each approach are shown in Table 6.2. Additionally, the displacements at the top, using either a four-segment beam model with these bending stiffnesses or the FE model, are presented, when a force of 10,000 kN is applied and the base of the model is fixed.

	$\mathbf{EI}_y$ [Nm <sup>2</sup> ]		$\mathbf{EI}_x$ [Nm <sup>2</sup> ]		<b>FEM</b>	
	Steiner	FE segment	Steiner	FE segment		
0	614e12	1.04e12	40e12	0.26e12	<b>X</b> <b>Y</b>	
I	614e12	6.46e12	40e12	5.53e12		
II	1255e12	170.84e12	88e12	48.81e12		
III	573e12	255.56e12	39e12	41.60e12		
$\Delta \mathbf{u}$ [mm]	16	-	236	-	26	199

**Table 6.2:** The bending stiffnesses and the total displacements determined with the different approaches.



**Figure 6.2:** The floor plan of segment 0.

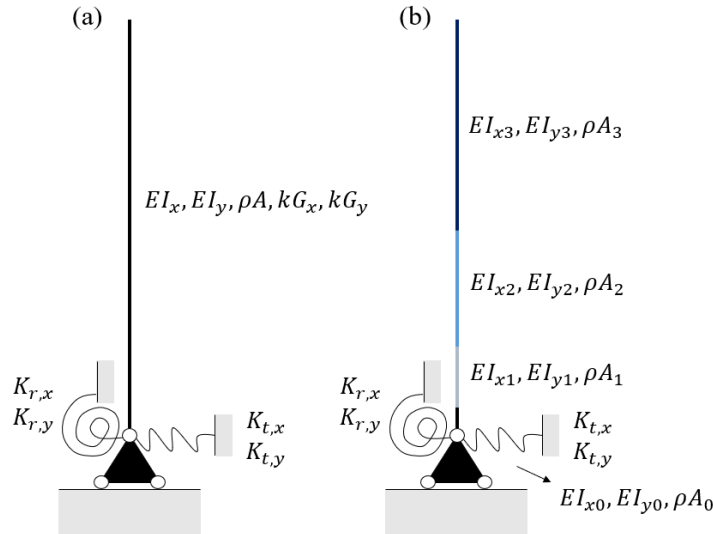
With the FE segment approach, as shown in Table 6.2, the bending stiffnesses in both x- and y-direction vary significantly. The top segment, which represent the smallest part of the building, shows the largest bending stiffness value. These differences are not observed with the bending stiffnesses obtained using the Steiner approach. The displacements obtained with the Steiner approach show similarities with the displacements obtained using the FEM model.

The FE segment approach fails to account for the intermediate relationships between the different segments. For example, segments 0 and I have column structures, which exhibit weaker behavior when considered individually than when integrated into the overall system, due the additional mass from other segments. The floor plan of segment 0 is given in Figure 6.2. The E-moduli values of the different segments in the FE model may also contribute to the differences, as the top segment has a E-modulus value of  $33 \times 10^9 \text{ N/m}^2$ , while the bottom segment has a E-modulus value of  $11 \times 10^9 \text{ N/m}^2$ ). However, it is not expected that this caused the factor of 100 difference between the segment stiffnesses.

Due to the differences in the segment stiffnesses obtained using the FE segment approach, the segment stiffnesses obtained using the Steiner approach are applied, with a correction factor included to account for the displacement differences between the FE model and four-segment beam using these segment stiffnesses. Since the FE model is weaker in the x-direction, the segment stiffnesses  $EI_y$  are multiplied by  $\frac{16}{26}$ . Since the FE model is stiffer in the y-direction, the segment stiffnesses  $EI_x$  are multiplied by  $\frac{236}{199}$ .

It is assumed that the ratios of the initial structural properties between the segments are well estimated with the Steiner approach. By maintaining these ratios, only one set of parameters needs to be updated, thereby reducing the total amount of parameters that needs to be updated.

The uniform Timoshenko beam model and the discrete Euler-Bernoulli beam model used in this study are shown in Figure 6.3. For the discrete Euler-Bernoulli beam, the same initial structural properties are applied as those of the four-segment Timoshenko beam model. For the uniform Timoshenko beam model, the volume weighted averages of these values are used.



**Figure 6.3:** Pictures of (a) the uniform Timoshenko beam model and (b) the discrete Euler-Bernoulli beam model.

### 6.1.2. Influence Parameters

The second study investigate the influence of the input parameters of the model. With model updating, a decision must be made about which parameters to update. To understand how each input parameter of the discrete Timoshenko beam model influences the modal properties of the model (i.e., natural frequencies and mode shapes), a sensitivity study is conducted where the values of each input parameter are varied within a specified range.

The range over which each input parameter is varied corresponds to their initial structural property value in Table 6.1, scaled by a factor of ten. Parameters  $A$  and  $\rho$ , representing the cross-section and density, respectively, are excluded from this study, as these parameters do not address model uncertainties.

### 6.1.3. Influence Optimization Algorithm

In the third study, the influence of the optimization algorithm on model updating with a discrete Timoshenko beam model is investigated through a convergence study. The study uses three different algorithms on the same case: Sequential Least Squares Quadratic Programming (SLSQP), Differential Evolution (DE), and Particle Swarm Optimization (PSO). For each algorithm, different amounts of candidate solutions are considered. The updating parameters and the value ranges considered in this study are chosen based on the study "Influence Input Parameters" in Section 6.1.2.

To conduct this study, measured modal properties of a building with known structural properties are required. Since such data is not available, as the structural properties of the Delftse Poort are not known, a numerical case is created using the structural property values from Table 6.1. Random scalars are applied to the initial values in Tables 6.1 in both x- and y-direction, creating another model with known structural and modal properties. The random scalars applied are listed in Table 6.3. The modal properties of this new model serve as the measured modal properties for the study. The amount of modal properties applied aligns the amount of measured modal properties of the Delftse Poort: one bending mode in the x-direction and two bending modes in the y-direction. The modal displacements considered correspond to the measured heights of the Delftse Poort (−3.0 m, 69.30 m, 108.90 m and 144.51 m).

It should be noted that the random scaling factors are applied exclusively to the parameters selected for updating. Otherwise, it would be impossible to completely match the model output with the measured modal properties after model updating.

	$K_r$	$K_t$	$E$	$I$	$\rho$	$kG$
<b>Scalar (-)</b>	0.4	9.0	7.0	1.5	2.0	0.2

**Table 6.3:** The random scalar applied on the initial structural properties values of Table 6.2.

### 6.1.4. Influence Amount of Measured Modal Properties

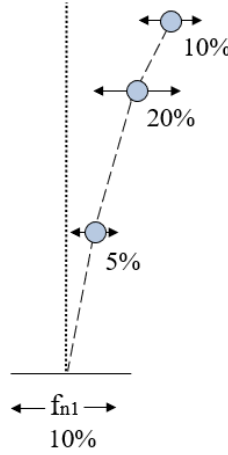
In the fourth study, the influence of the amount of measured modal properties on model updating with the discrete Timoshenko beam model is examined. A convergence study is conducted that considers an increasing number of modes on the same case:

- The first bending mode in the x- and y-direction
- The first bending mode in the x- and the first two bending modes in y-direction

Different amounts of candidate solutions are considered for each amount of modes. The updating parameters and the value ranges considered in this study are chosen based on the study "Influence Input Parameters" in Section 6.1.2. The optimization algorithm is chosen based on the study "Influence Optimization Algorithm" in Section 6.1.3. Since this study also requires measured modal properties of a building with known structural properties, the numerical case from the study "Influence Optimization Algorithm" is also applied here.

### 6.1.5. Influence Measurements Uncertainties

In the fifth study, the impact of errors in the measured modal properties (i.e., the measured natural frequencies and mode shapes) on model updating with the discrete Timoshenko beam model is investigated. The sensitivity study introduces random errors into the measured modal properties to understand how these uncertainties affect the results obtained. The errors applied to the natural frequencies range up to 10%, and the errors applied to the mode shapes range up to 20%. An example of an error distributions is illustrated in Figure 6.4 for the first natural frequency and mode shape.



**Figure 6.4:** A visualization of the random errors applied on the first natural frequency and mode shape.

The updating parameters and the value ranges considered in this study are chosen based on the study "Influence Input Parameters" in Section 6.1.2. The optimization algorithm is chosen based on the study "Influence Optimization Algorithm" in Section 6.1.3. The amount of measured properties from Table 6.4 used to fit the model and the amount of candidate solution considered are based on the study "Influence of the Amount of Measured Properties" in Section 6.1.4. Since this study also requires measured modal properties of a building with known structural properties, the numerical case from the study "Influence Optimization Algorithm" is applied here as well.

#### 6.1.6. Estimating Structural Properties

The last study applies model updating on the Delftse Poort to estimate its structural properties using the discrete Timoshenko beam model and the measured modal properties.

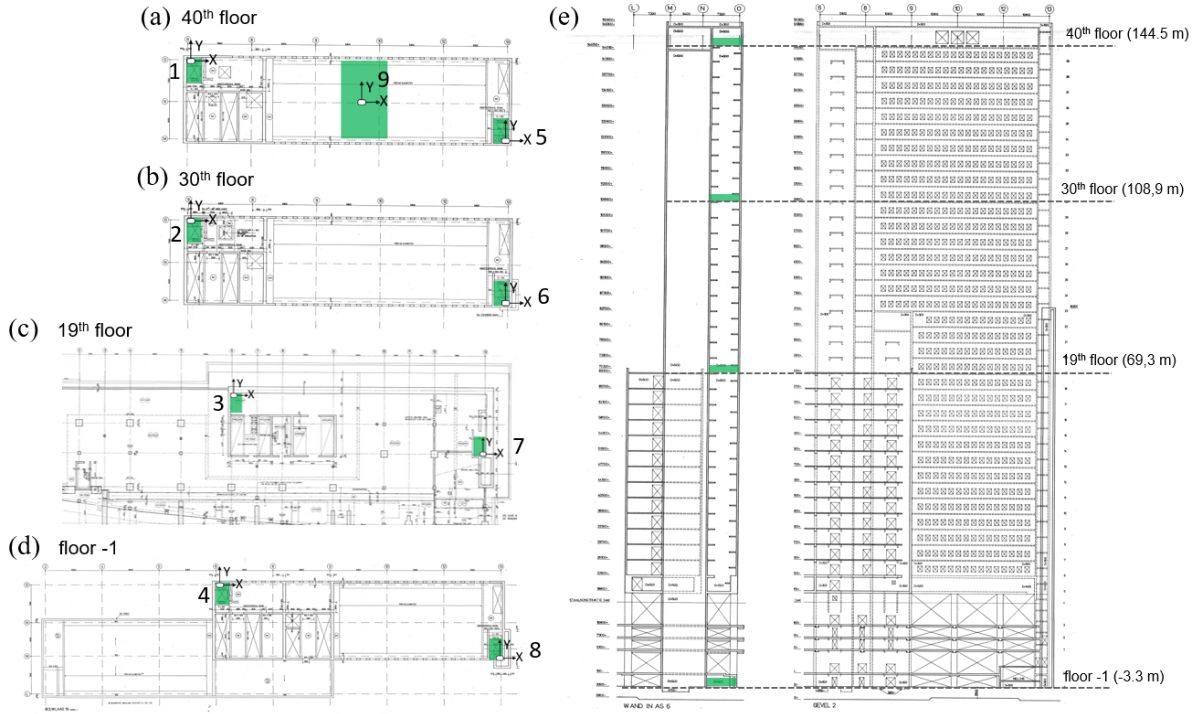
Mode	1	2	3
Dominant direction	Y	X	Y
Natural frequency [Hz]	0.403	0.861	1.624
<b>Modal Displacement</b>			
Height 1 (−3.30 m)	0.01	−0.06	0.01
Height 2 (69.30 m)	0.33	−0.49	0.49
Height 3 (108.90 m)	0.66	−0.76	0.12
Height 4 (144.51 m)	0.99	−1.00	−0.56

**Table 6.4:** The measured natural frequencies and modal displacements of the office tower the Delfste Poort.

The measured modal properties of the Delftse Poort are indicated in Table 6.4. To determine these modal properties, acceleration sensors were installed on the 1st, 19th, 30th, and 40th floors. The positions and orientations of the sensors are illustrated in Figure 6.5. Details of the data analysis applied to obtain the measured modal properties are described in Appendix D.

The updating parameters and the value ranges considered for these updating parameters are based on the study "Influence Parameters" in Section 6.1.2. The algorithm applied in the model updating process is determined using the study "Influence Optimization Algorithm" in Section 6.1.3. The amount of measured properties from Table 6.4 used to fit the model and the amount of candidate solution considered are based on the study "Influence of the Amount of Measured Properties" in Section 6.1.4.

The model updating in this study follows the approach described by Moretti et al. [1] and consists of two parts. In the first part, model updating is carried out using the specified settings above. However, unlike the method used by Moretti et al. [1], which retains the lowest quantile of the candidate solutions (i.e., the quantile that best fit the measured modal properties after updating), this study



**Figure 6.5:** Pictures of (e) the Delfste Poort and the positions of the acceleration sensors (green squares) on (d) floor 1, on (c) the 19th floor, (b) the 30th floor and (a) the 40th floor. The arrows indicate the directions of the acceleration sensors.

retains the lowest octile of the candidate solutions. This modification is done because the discrete Timoshenko beam model is expected to be more complex than the uniform Euler-Bernoulli beam model, resulting in a more complex solution space for identifying the global minimum. New structural property value ranges for the updating parameters are defined based on the minimum and maximum values of these retained solutions. Using these newly refined ranges, another selection and model updating process is carried out (the second part) in the same manner as the first part. The reasons behind certain choices made by Moretti et al. [1] in this procedure are unclear. This approach was chosen for the Delfste Poort to stay consistent with the research conducted on the New Orleans in Chapter 5.

## 6.2. Results

In this section, the results of the studies are presented. Sections 6.2.1 to 6.2.5 present the results of the studies where the influences of certain settings and uncertainties in model updating with the discrete Timoshenko beam were investigated. Section 6.2.6 shows the results of applying model updating to estimate the structural properties of the Delfste Poort.

### 6.2.1. Influence Model

To begin, the results of the first study are presented. The natural frequencies obtained with the different beam models are shown in Tables 6.5 and 6.6. Their mode shapes are illustrated in Figure 6.6. The modal properties of the Finite Element (FE) model and the measured modal properties are also shown as reference.

According to Table 6.5, the discrete Timoshenko beam model is closer to the measured natural frequencies than the FE model. This is unexpected, as the FE model incorporates more realistic boundary conditions and better captures complex geometries compared to the discrete Timoshenko model. Moreover, the discrete Timoshenko beam model is created using the FE model. Therefore, it is unlikely that the discrete Timoshenko beam model provides a more accurate approximation of the high-rise building.

The uniform Timoshenko beam model generally matches the measured natural frequencies less accurately than the discrete Timoshenko beam model. This is expected, as a uniform Timoshenko beam model is not well-suited for representing buildings with irregular stiffness. A discrete beam model is needed for accurate representation. In the y-direction, the discrete Euler-Bernoulli beam model shows closer resembles to the measured natural frequencies than the discrete Timoshenko beam model. This suggests that excluding shear contributions may be more accurate for this direction.

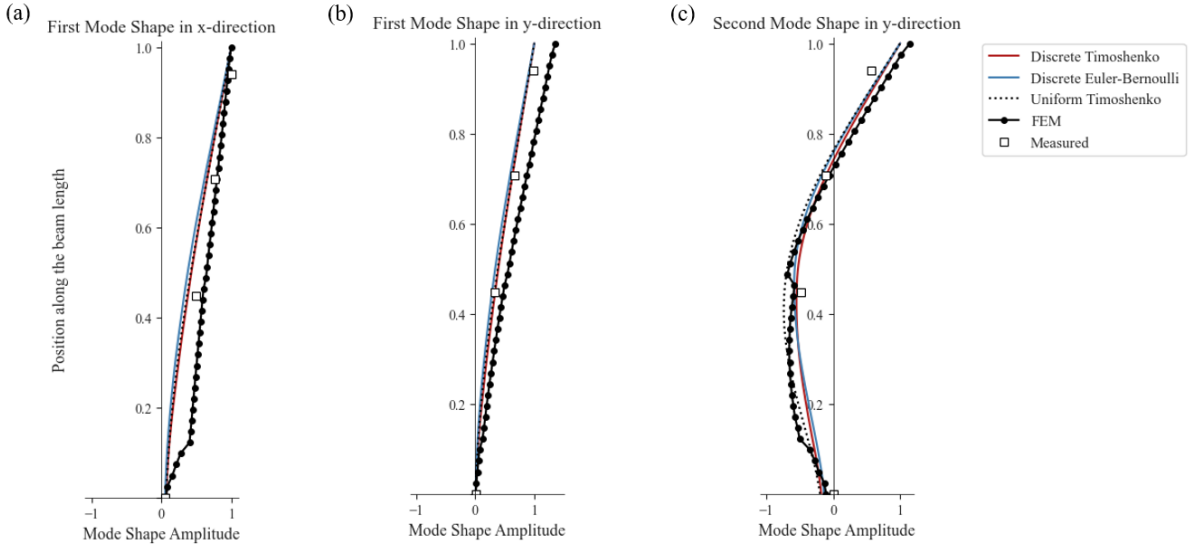
Figure 6.6 shows that the modes shapes of the beam models and the FE model are comparable to the measured mode shapes. However, a notable difference is observed in the first bending mode in the x-direction at lower heights, where the FE model displays more displacement compared to the beam models. The FE model shows a kink. A thorough analysis of the FE model could provide valuable insights into the obtained results.

Measured [Hz]		FEM ( $\Delta_{\text{measured}}$ %) [Hz]		Discrete Timoshenko beam model ( $\Delta_{\text{measured}}$ %) [Hz]	
X	Y	X	Y	X	Y
0.86	0.40	0.62 (-27.9%)	0.28 (-30%)	0.76 (-11.6)	0.29 (-27.5)
	1.62		1.13 (-28.4%)		1.45 (-10.5%)

**Table 6.5:** The measured natural frequencies, the natural frequencies of the FE model and the natural frequencies of the discrete Timoshenko beam model.

Uniform Timoshenko beam model ( $\Delta_{\text{measured}}$ %) [Hz]		Discrete Euler-Bernoulli beam model ( $\Delta_{\text{measured}}$ %) [Hz]	
X	Y	X	Y
0.61 (-29.1)	0.23 (-42.5)	0.96 (+11.6)	0.37 (-7.5)
	1.37 (-15.4%)		1.61 (-0.6%)

**Table 6.6:** The natural frequencies obtained with the uniform Timoshenko and discrete Euler-Bernoulli beam models.



**Figure 6.6:** The measured and model mode shapes.



### 6.2.2. Influence Parameters

This section presents the results of second study, which changed the input parameter values of the discrete Timoshenko beam model to see their influence on the modal properties. The results of parameters  $K_r$  and  $K_t$  are shown in Figures 6.7 and 6.8. The results of parameters  $E$  and  $I$  are shown in Figures 6.10 and 6.9. The results of parameter  $kG$  is shown in Figure 6.11. The change in the mode shape is quantified using the MAC (Equation 2.2).

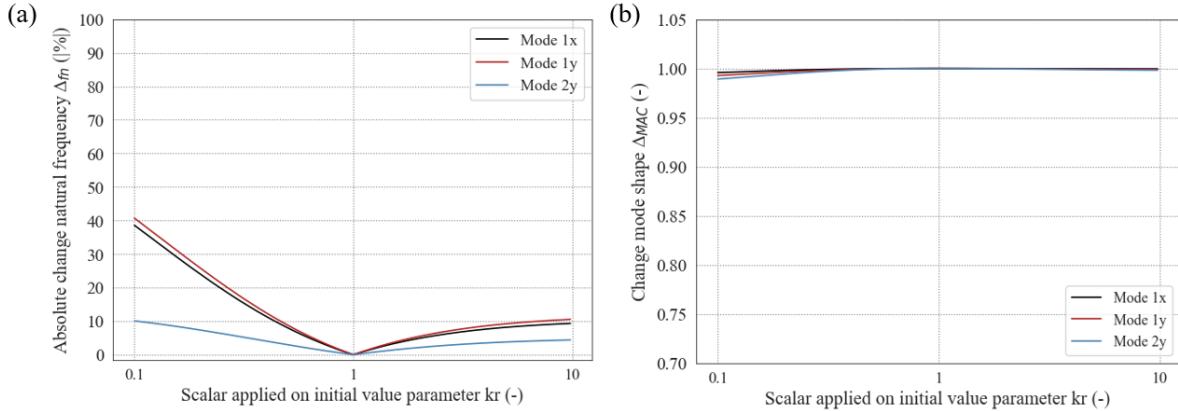
According to Figure 6.7, parameter  $K_r$  primarily has influence on the first natural frequency. It does not have influence on the mode shapes. Figure 6.8 shows that parameter  $K_t$  has influence on the first bending mode in the x-direction and the second bending mode in the y-direction, but have almost no influence on the first bending mode in the y-direction.

Parameters  $E$  and  $I$  have the same influence on the bending modes. As stated before with the New Orleans, these parameters often occupy similar positions within the model. Due to the similarity between parameters  $E$  and  $I$ , also for the Delftse Poort, it is chosen to consider the updating results of these parameters together as  $EI$  for the rest of the research.

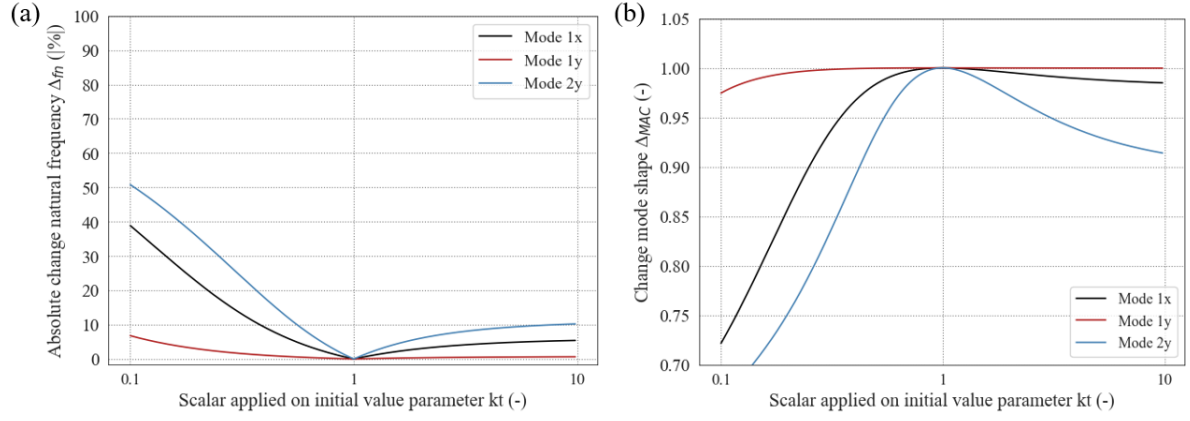
Figure 6.11 shows that adjusting the value of the parameter  $kG$  has limited effect on the model output. It is not selected as an updating parameter. This is unexpected, as for the x-direction, the wide side of the building, a shear-dominant behavior was expected of the high-rise building. However, it should be noted that only the first bending mode is considered in x-direction. It is suspected that with higher bending modes, parameter  $kG$  shows more influence.

Based on the results, it is chosen to update parameters  $K_r$ ,  $K_t$ ,  $E$ , and  $I$  in the following studies of the research.

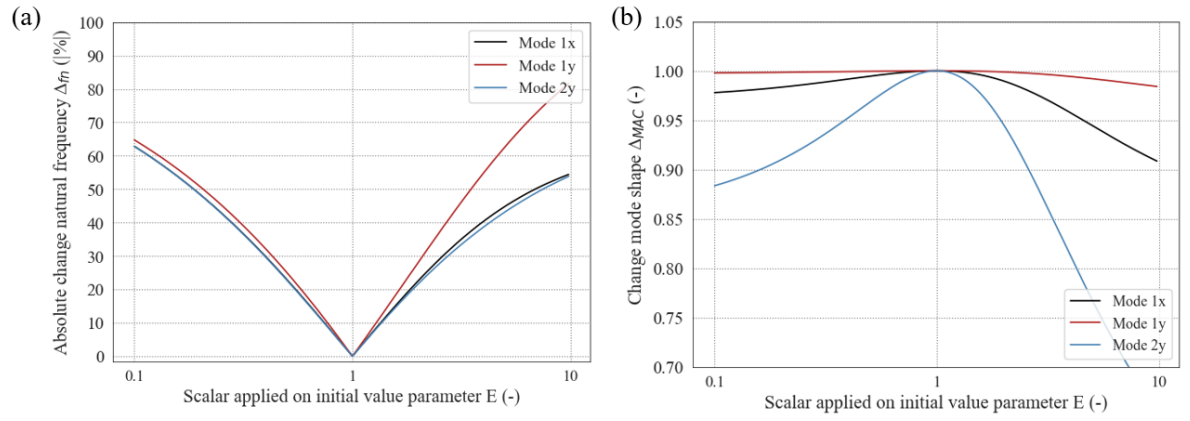
To summarize, it should be noted that parameter  $K_r$  primarily influence the first natural frequencies. Parameter  $K_t$  affects the first bending mode in the x-direction and the second bending mode in the y-direction, but does not have influence on the first bending mode in the y-direction. Parameters  $E$  and  $I$  exhibit similar influences. Therefore, the results for these parameters will be considered together as  $EI$  for the rest of the research. Parameter  $kG$  is not selected, as the modal properties of the model are insensitive to changes in this parameter.



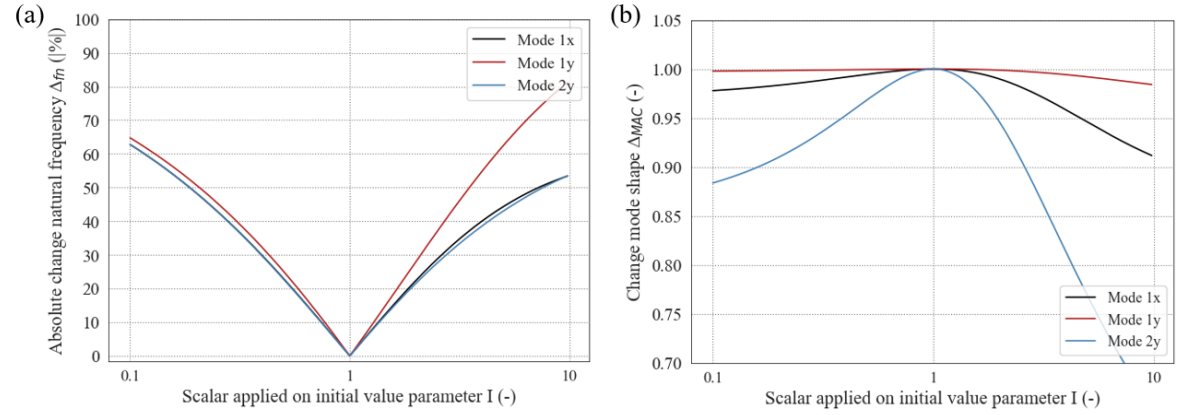
**Figure 6.7:** The influence of parameter  $K_r$  on the model output (a,b).



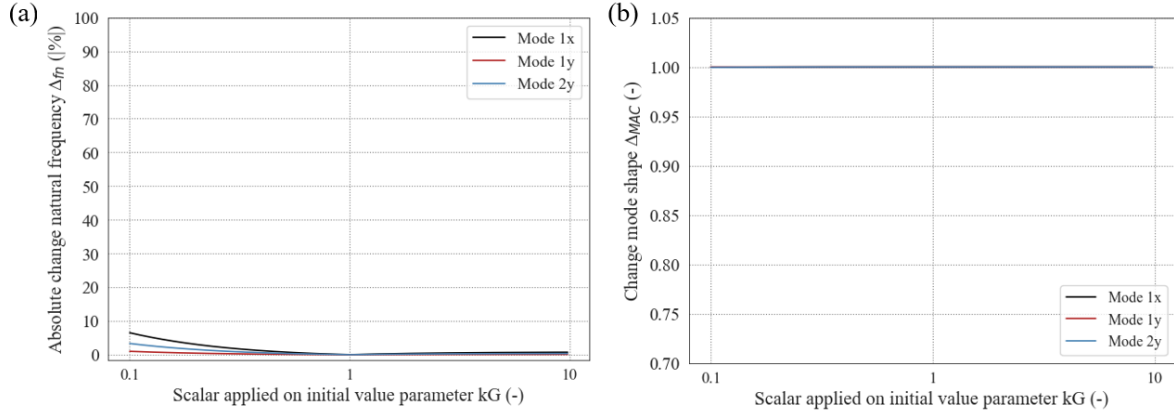
**Figure 6.8:** The influence of parameter  $K_t$  on the model output (a,b).



**Figure 6.9:** The influence of parameter  $E$  on the model output (a,b).



**Figure 6.10:** The influence of parameter  $I$  on the model output (a,b).



**Figure 6.11:** The influence of parameter  $kG$  on the model output (a,b).

### 6.2.3. Influence Optimization Algorithm

In this section, the results of the third study are shown, which applied different optimization algorithms on the same numerical case. The updating parameters were  $K_r$ ,  $K_t$ ,  $E$  and  $I$ , which were updated at the same time.

To create the numerical case, random scalars were applied to the initial values of the chosen updating parameters, indicated in Table 6.1, to generate another model with known structural properties and mode shapes. The scalars applied are indicated in Table 6.7. The modal properties of this new model, indicated in Table 6.8, were selected as measured modal properties for this study.

	$K_r$	$K_t$	$E$	$I$
<b>Scalar (-)</b>	0.4	9.0	7.0	1.5

**Table 6.7:** The random scalar applied on the initial structural properties values of Table 6.1.

<b>Mode</b>	<b>1</b>	<b>2</b>	<b>3</b>
Dominant direction	Y	X	Y
Natural frequency [Hz]	0.38	1.02	4.00
<b>Modal Displacement</b>			
Height 1 (-3.0 m)	0.00	0.02	-0.16
Height 2 (69.30 m)	0.45	0.47	-0.161
Height 2 (108.90 m)	0.74	0.74	-0.02
Height 3 (144.51 m)	1.00	1.00	1.00

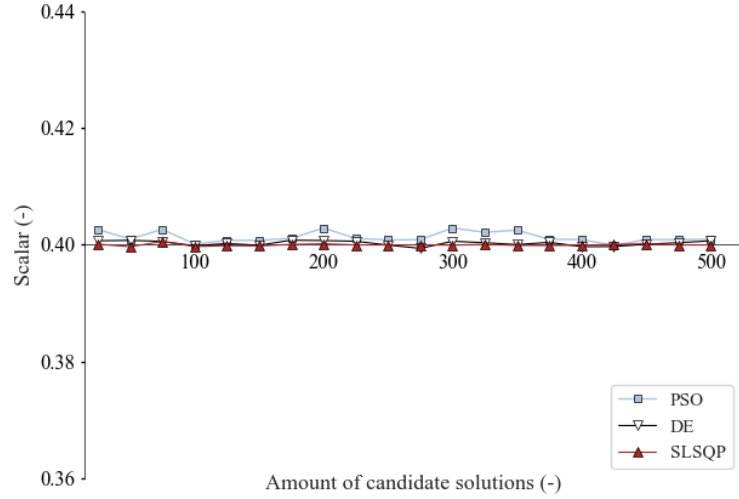
**Table 6.8:** The modal properties of the numerical case.

Different amounts of candidate solutions were considered, and for each amount, the candidate solution that best fitted the measured modal properties after model updating is shown. To maintain the compactness of the results, the results are presented as ratios relative to their initial values in Table 6.1. For example, a value of 0.8 indicates that after the model updating, the new value of the structural property is 80% of its initial value. Therefore, if the model updating was successful with the chosen algorithm, the results should match the scalars listed in Table 6.7. The results of parameters  $K_r$  and  $K_t$  are shown in Figures 6.12 and 6.13, respectively. The results of parameters  $E$  and  $I$  are combined and shown as  $EI$  in Figure 6.14.

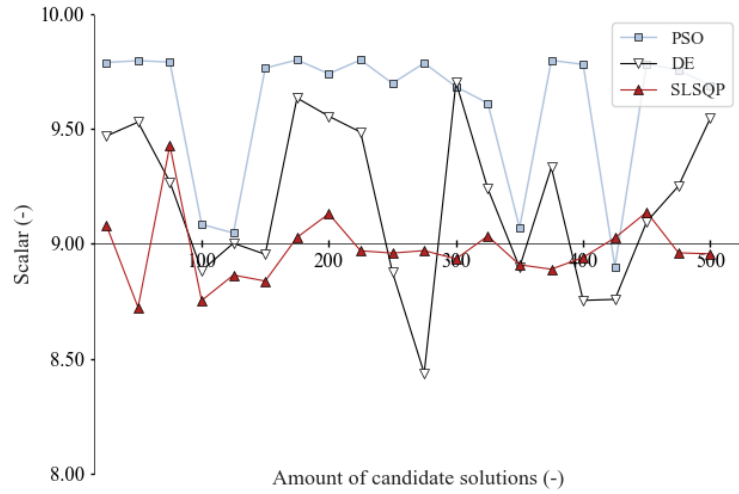
Figure 5.15 shows that  $K_r$  is well estimated by all algorithms. However, the optimization algorithms PSO and DE have difficulty with estimating parameter  $K_t$ . Increasing the amount of candidate solutions does not improve the estimates. This suggest that more candidate solutions need to be considered to sufficiently explore the solution space.

The SLSQP algorithm estimates parameter  $K_t$  well after 200 candidate solutions, however, small deviations are observed. The algorithm has more difficulty with estimating this parameter. Small deviation can be expected, as the model updating process stops once a certain convergence threshold is reached, which may occur earlier for some initiations than others. Therefore, this algorithm is selected for the following studies.

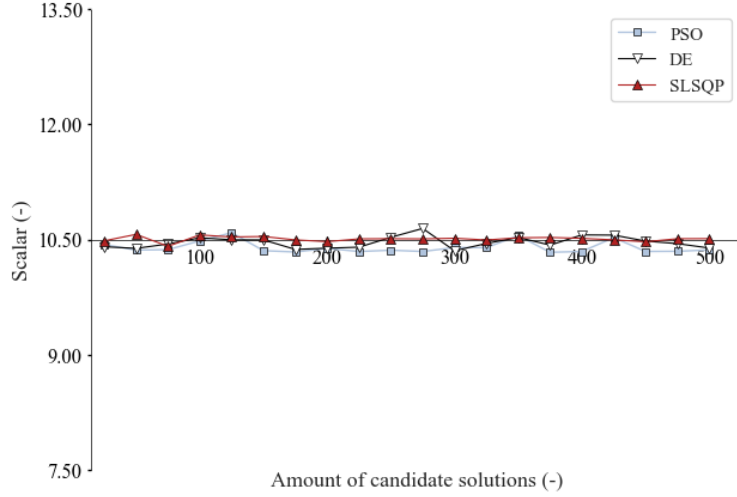
Figure 6.14 shows that  $EI$  is well estimated by all optimization algorithms.



**Figure 6.12:** The influence of the different algorithms on the model updating result of parameter  $K_r$ .



**Figure 6.13:** The influence of the different algorithms on the model updating result of parameter  $K_t$ .



**Figure 6.14:** The influence of the different algorithms on the model updating result of the product  $EI$ .

#### 6.2.4. Influence Amount of Measured Modal Properties

This section presents the model updating results of the fourth study, which applied different amount of measured modal properties on the same numerical case. The parameters chosen to update in this study were  $K_r$ ,  $K_t$ ,  $E$  and  $I$ , which were updated at the same time. The SLSQP was chosen as optimization algorithm.

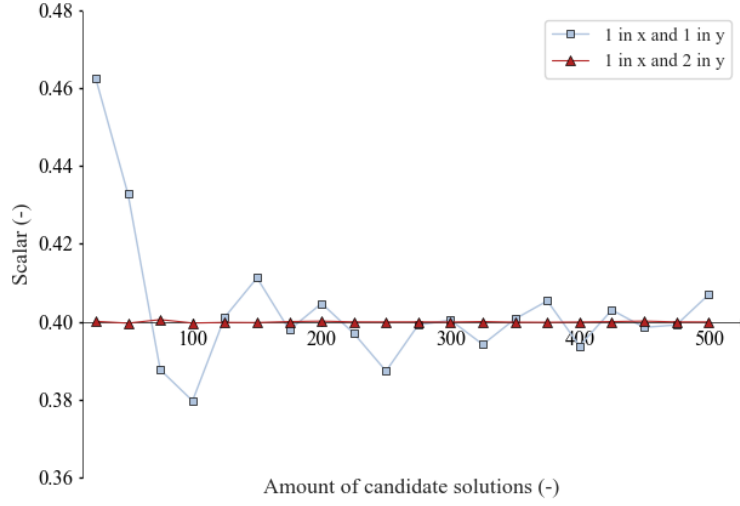
This study used the same numerical case described in the study "Influence Optimization Algorithm" in Section 6.2.3. Different amounts of candidate solutions were considered, and for each amount, the candidate solution that best fitted the measured modal properties after updating is shown. The results per parameter are shown as ratios relative to their initial values. Therefore, if the model updating was successful with the amount of measured modal properties, the results should match the scalars listed in Table 6.7. The results of the parameters  $K_r$  and  $K_t$  are shown in Figures 6.15 and 6.16, respectively. The results of the parameters  $E$  and  $I$  are shown together as  $EI$  in Figure 6.17.

Figure 6.15 shows that it is possible to estimate parameter  $K_r$  considering only the lowest bending mode in both x-and y-direction. This also in line with the influences obtained in Figure 6.7 in Section 6.2.2, as it showed that this parameter primarily influence the first natural frequency.

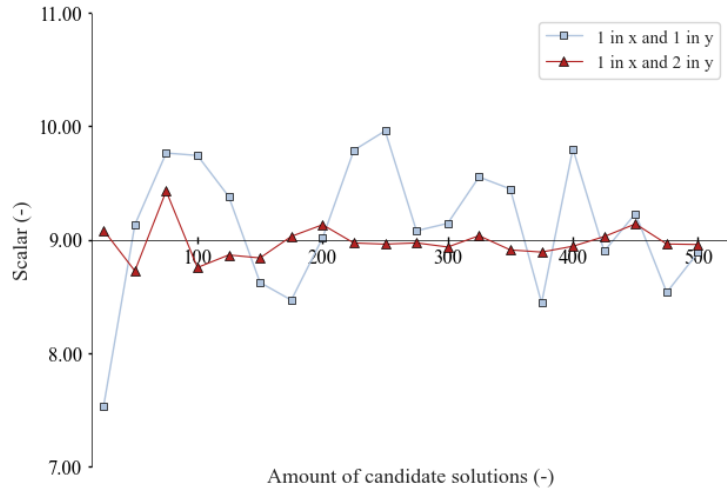
Figures 6.16 and 6.17 show, however, that it is insufficient to estimate  $K_t$  and  $EI$  considering only the lowest bending mode in the x- and y-direction. According to Figure 6.8 in Section 6.2.2, parameter  $K_t$  shows only influence on the first bending mode in the x-direction and the second bending mode in the y-direction. Excluding the second bending mode in the y-direction results in insufficient information to estimate this parameter.

However, according to 6.9 and 6.10, parameters  $E$  and  $I$  show sensitivity to the first bending mode. It is suspected that the first bending mode alone does not provide enough information to estimate these parameters. Higher bending modes, which are more sensitive to changes in bending stiffness, are needed to estimate these parameters.

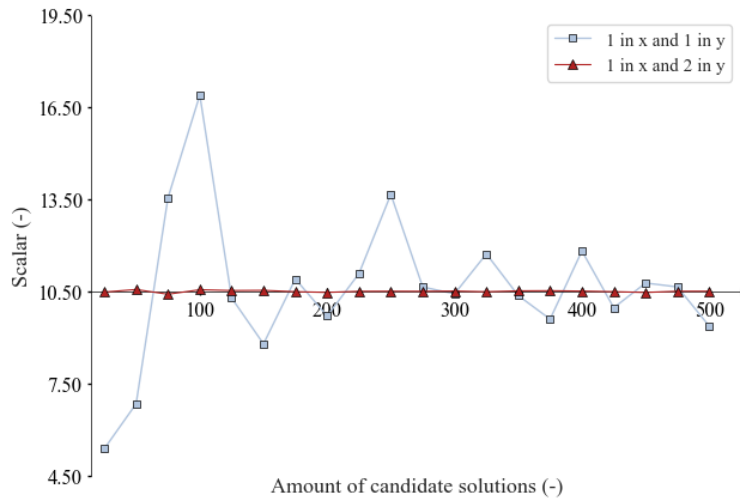
To estimate parameter  $K_t$ , a minimum of 200 candidate solutions is required. Based on the results, it is chosen to consider all the measured modal properties listed in Table 6.18 in the study "Estimating Structural Properties", using 400 candidate solutions. For the study "Influence of Measurement Uncertainties", 200 candidate solutions are used.



**Figure 6.15:** The influence of the amount of modal information on the model updating result of parameter  $K_r$ .



**Figure 6.16:** The influence of the amount of modal information on the model updating result of parameter  $K_t$ .

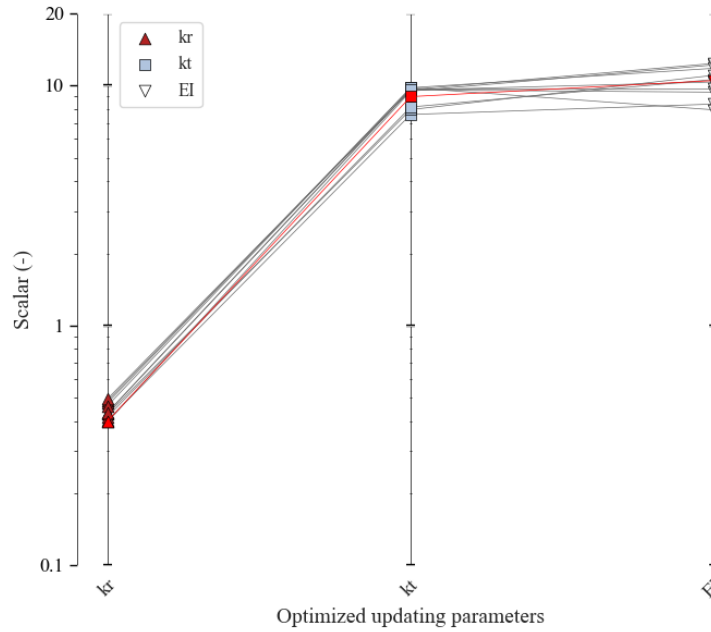


**Figure 6.17:** The influence of the amount of modal information on the model updating result of  $EI$ .

### 6.2.5. Measurement Uncertainties

This section shows the results of the fifth study, which applied errors to the measured modal properties to see their influence on the results obtained. The parameters updated in this study were  $K_r$ ,  $K_t$ ,  $E$  and  $I$ . These were updated at the same time, using the SLSQP optimization algorithm. The study used the same numerical case described in the study "Influence Optimization Algorithm" in Section 6.2.3. Two hundred candidate solution were considered, but only the candidate solution that best fitted the measured modal properties is shown. As the procedure of adding random errors, conducting model updating, and retaining the best-fitted solution was done ten times, ten solutions are shown in total.

The results for each parameter are presented as ratios relative to their initial values. Therefore, if the model updating was insensitive to the added errors in the measured modal properties, the results should match the scalars listed in Table 6.7. The results are shown in Figure 6.2.5. The values of the different parameters for each solution are connected by grey lines. The red solution indicates the scalar values of Table 6.7.



**Figure 6.18:** The influence of measurement uncertainties on the model updating results.

Parameter  $K_r$  shows limited sensitivity to the uncertainties in the measured model properties. The values of this parameter are between 0.39 and 0.49.  $K_t$  and  $EI$ , however, show significant sensitivity to the uncertainties. The values of parameter  $K_t$  range between 7.58 and 10.00. The values of  $EI$  range between 7.93 and 12.38.

### 6.2.6. Estimating Structural Properties

To end, the estimated structural properties values of the Delftse Poort are shown. The structural parameters updated in this study were  $K_r$ ,  $K_t$ ,  $E$  and  $I$ . The SLSQP algorithm was applied. Four hundred candidate solutions were considered, and the lowest octile of these solutions, i.e., the octile with the smallest objective function values, were retained.

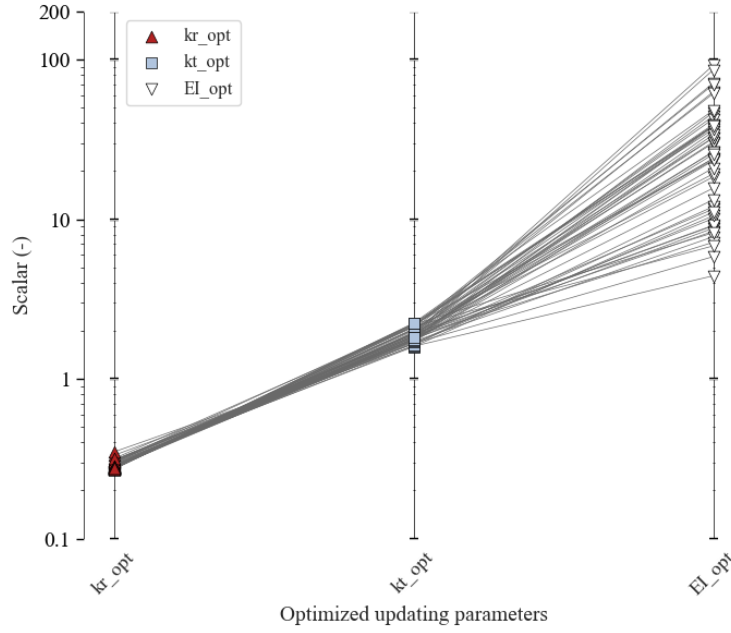
Table 6.9 presents the estimated and measured natural frequencies and the Modal Assurance Criterion (MAC) values between the estimated and measured mode shapes. Case 1 indicates the results of model updating using the measured modal properties of Table 6.4.

The median natural frequency for Case 1 in the x-direction is close to the measured natural frequencies. However, in the y-direction, the median of the second natural frequency shows differences compared to the measured second natural frequency.

Mode [-] (dominant direction)	Natural frequency [Hz]			MAC [-]
	Design	Measured	Case 1 ( $\Delta_{\text{measured}}$ %)	Measured vs. Case 1
1 (x)	0.620	0.861	0.861 (0%)	0.999
1 (y)	0.280	0.403	0.403 (0%)	0.999
2 (y)	1.130	1.624	1.803 (+11%)	0.910

**Table 6.9:** The measured and estimated natural frequencies and MAC values between the measured and estimated mode shapes.

Additionally, the MAC values indicate that the second bending mode shape in the y-direction is not accurately estimated. Therefore, the obtained structural properties values in y-direction are not considered.



**Figure 6.19:** The estimated parameter values of Case 1 in x-direction.

Figure 6.19 shows the estimated structural property values in the x-direction. The values of the different parameters that belong together are connected with grey lines. For the estimates per parameter, the Coefficient of Variation (CV) is calculated. The CV shows the relative variability of the estimates:

$$CV = \left( \frac{\sigma}{\mu} \right) \times 100\% \quad (6.3)$$

where  $\sigma$  is the standard deviation and  $\mu$  is the mean. A lower CV indicates less variability relative to the mean, suggesting more consistency in the estimates. A higher CV indicates greater dispersion in the estimates. The variability in the distribution of  $K_{r,y}$  is low, indicated by a coefficient of variation (CV) of 5.58%. The distribution suggests that the value should be lower than the design value. The variability of the distribution of  $K_{t,x}$  is also low, indicated by a coefficient of variation (CV) of 9.49%. The distribution suggests that the value should be higher than the design value.

However, the distribution of the parameter  $EI_y$  exhibit high uncertainty, as the estimates are scattered across the range. The coefficients of variation is 71.17%. Based on these results, it can be stated that estimating the structural parameters of the Delftse Poort using vibration-based model updating was not effective.



### 6.3. Main Findings

With the research, several studies were conducted to evaluate the influence of certain settings and uncertainties in model updating and to assess the effectiveness of the technique in estimating the structural properties of a shear-dominant building with irregular stiffness across its height. The main findings were:

- **Setup model:** Obtaining accurate values for bending stiffness using the FE model proved difficult. The FE approach, which used the relative displacement  $\Delta$  of the FE segment to calculate the bending stiffness, did not account for the intermediate relationships between segments. Segments 0 and I had column structures, which exhibited weaker behavior when considered individually compared to when they were part of the overall system. This was due to the additional mass of the other segments. Therefore, considering these segments individually led to an inaccurate assessment of their bending stiffness. The FE segment approach resulted in bending stiffness values that differed by a factor of 100.
- **Influence of Parameters:** Parameter  $K_r$  showed primarily influence on the first natural frequencies. Parameter  $K_t$  showed influence on the first bending mode in the x-direction and the second bending mode in the y-direction, but did not show influence on the first bending mode in the y-direction. Parameters  $E$  and  $I$  exhibited similar influences. Parameter  $kG$  showed no influence on the modal output.
- **Influence of Amount of Modal Properties:** It was possible to estimate parameter  $K_r$  considering only the lowest bending mode in both x- and y-directions. However, this was insufficient to estimate  $K_t$  and  $EI$ .
- **Influence of Measurement Uncertainties:** Parameter  $K_r$  showed limited sensitivity to the uncertainties in the measurements, with values obtained between 0.39 and 0.49, closely aligning with the expected value of 0.40. In contrast, parameters  $K_t$  and  $EI$  exhibited significant sensitivity to uncertainties. The values of parameter  $K_t$  ranged between 7.58 and 10.00, with the expected value being 9.0. For  $EI$ , the values ranged between 7.93 and 12.38, while the expected value was 10.5.
- **Estimating Structural Properties:** The discrete Timoshenko beam model was not able to match the natural frequency of the measured second bending mode in the y-direction after updating. Additionally, the MAC values indicated that the second bending mode shape in the y-direction was not accurately estimated. Moreover, the distribution of the parameters  $EI_y$  exhibited high uncertainty, as the estimates were scattered across the range. The coefficient of variation was 71.17%. Based on these results, it can be stated that estimating the structural parameters of the Delftse Poort using vibration-based model updating was not effective.

# Discussion

In this research, various studies were conducted on the New Orleans and the Delftse Poort to investigate the effectiveness of vibration-based model updating with a discrete Timoshenko beam model in estimating the structural properties of high-rise buildings. Their findings require explanation, which is provided in this chapter.

First, the setup of the models need to be discussed. The discrete Timoshenko beam models used to approximate the dynamic behavior of the New Orleans and the Delftse Poort were created using their Finite Element (FE) models. However, obtaining values for the bending stiffnesses using the FE model was challenging.

The Steiner approach, which calculated the bending stiffness per segment using the elastic modulus  $E$  and the second moment of inertia  $I$  (calculated with  $\frac{1}{12}bh^3$  and the Steiner rule), assumed fully rigid connections between elements when determining  $I$ , making it not possible to accurately account for the contributions of elements that were not rigidly connected.

Moreover, the FE approach, which used the relative displacement  $\Delta$  of the FE segment to calculate the bending stiffness per segment, also posed problems, as it did not account for the intermediate relationships between the segments. For both the New Orleans and the Delftse Poort, segments 0 and I had column structures, which exhibited weaker behavior when considered individually compared to when they were part of the overall system, due to the additional mass of the other segments. Therefore, considering these segments individually led to an inaccurate assessment of their bending stiffnesses. This was particularly evident in the case of the Delftse Poort, where the FE segment approach resulted in bending stiffness values differing by a factor of 100.

These difficulties in determining the bending stiffness values per segment pose a problem for model updating, as the ratios of the initial structural properties were maintained for both high-rise building applications. Due to the superstructure of the discrete Timoshenko beam model, a separate set of parameters needs to be defined for each segment. By maintaining the ratios of the initial structural property values, only one set of parameters needs to be updated, thereby reducing the total number of parameters that need to be updated. However, maintaining these ratios gives weight to the initial structural property values. These values must be correct. Otherwise, model updating is limited by the incorrect ratios and will not be able to accurately match the measured modal properties. Therefore, to obtain a more accurate estimation of the structural properties, it is crucial to obtain accurate values for the bending stiffness.

Despite these challenges with the bending stiffnesses, applying the discrete Timoshenko beam model for model updating to estimate the structural properties of the New Orleans has proven to be effective. When only the lowest two bending modes in both directions were used to fit the model, the estimates of parameter  $K_{r,y}$  showed low uncertainty, with a Coefficient of Variation of 6.91%. This is an improvement compared to the study conducted by Moretti et al. [1], which obtained a Coefficient of Variation of 46.9% with the uniform Euler-Bernoulli beam model. Therefore, to obtain a more accurate estimation of parameter  $K_{r,y}$ , it is crucial to incorporate shear deformations into the model and account for irregular stiffness along the height. However, further investigation is needed to clarify whether the improved results of the structural property  $K_{r,y}$  are primarily attributed to the discreteness of the beam, the added shear or a combination of both.

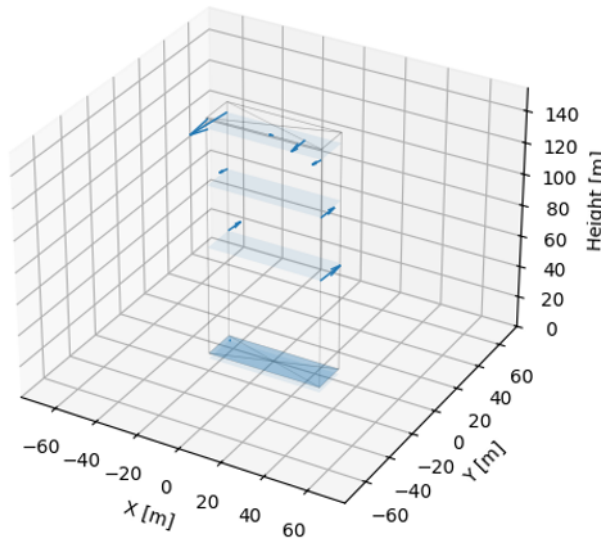
Despite incorporating additional features compared to the uniform Euler-Bernoulli beam model used in the study by Moretti et al. [1], the model was not sufficient for accurately describing the third bending mode.

This is indicated by a Modal Assurance Criterion (MAC) value of 0.87. This suggests that essential aspects are missing from the model that are crucial for matching the mode shape of the third bending mode.

Applying model updating with the discrete Timoshenko beam model to estimate the structural properties of the Delftse Poort was not effective. Parameter  $EI_y$  exhibits large uncertainty, which is indicated by a Coefficient of Variation of 71.17%. Several factors could contribute to this discrepancy. One possible cause might be that the value ranges considered for parameters  $E$  and  $I$  are too narrow. Errors in the initial values of these parameters could also be the cause. As previously mentioned, accurately estimating the values for bending stiffness was challenging. As the ratios of the initial structural properties values were maintained, the model updating may be constrained by incorrect ratios, which prevents it from accurately matching the measured modal properties. Moreover, only one bending mode in the  $x$ -direction was used to fit the discrete Timoshenko model. It is suspected that the first bending mode alone does not provide enough information to estimate parameters  $E$  and  $I$ . Higher bending modes, which are more sensitive to the bending stiffness, are needed, as they provide more comprehensive information.

However, the need for higher bending modes poses challenges for model updating, as higher bending modes are more likely to contain errors. For both the New Orleans and the Delftse Poort,  $K_t$  and  $EI$  show sensitivity to the measurement uncertainties. When errors in the natural frequency of up to 10% and in the mode shapes of up to 20% were applied, the values of  $K_t$  and  $EI$  for the New Orleans ranged from 6.41 to 10.0 and from 10.29 to 12.28, respectively. For the Delftse Poort, the values of  $K_t$  and  $EI$  ranged from 7.58 to 10.0 and from 7.93 to 12.38, respectively. The target values for  $K_t$  and  $EI$  for both cases were 9.0 and 10.5, respectively. Therefore, to achieve a more accurate estimation of these parameters, it is essential to minimize the errors of the measured modal properties to below these thresholds. This is not the case for parameter  $K_r$ . For both applications, parameter  $K_r$  did not exhibit significant sensitivities to measurement uncertainties. For the New Orleans, parameter  $K_r$  showed values between 0.40 and 0.47. For the Delftse Poort, parameter  $K_r$  showed values between 0.39 and 0.49. The target value of this parameter was 0.4.

Furthermore, for the Delftse Poort, the discrete Timoshenko beam model was unable to match the natural frequency of the measured second bending mode in the  $y$ -direction after updating. Additionally, the Modal Assurance Criterion (MAC) values indicated that the second bending mode shape in the  $y$ -direction was not accurately estimated. This result was unexpected, as the discrete Timoshenko beam model was able to successfully match the second measured bending mode for the New Orleans in both the  $x$ - and  $y$ -directions.



**Figure 6.20:** The mode shape of the second bending mode in  $y$ -direction of the Delftse Poort.

The measured mode shape is shown in 3D in Figure 6.20. The differences in the magnitude and direction of the arrows indicate twisting of the building. This twisting cannot be modeled with the discrete Timoshenko beam model, as it only can model pure bending modes. Furthermore, since the discrete Timoshenko beam model requires a single value per height, while multiple sensors were used to measure the displacements, the measurements had to be averaged. This averaging process may have led to a mode shape that deviates from the actual behavior, resulting in an inaccurate representation of reality. These issues pose a challenge for model updating with the Delftse Poort, as now only the first measured bending modes could be accurately applied, which provide alone insufficient information to estimate the parameters  $EI$  and  $K_t$ . Therefore, more pure bending modes are needed to obtain an accurate estimation of these parameters.

# Conclusion

This research focused on applying vibration-based model updating to estimate the structural properties of high-rise buildings using a discrete Timoshenko beam model. It provided valuable insights into how the model affects the accuracy of estimating structural properties for high-rise buildings and expanded the use of vibration-based model updating to buildings that are shear-dominant in behavior and have irregular stiffness distributions across their height. The main objective of the research was to achieve a more accurate estimation of the structural properties of high-rise buildings using vibration-based model updating. Several key aspects were highlighted as essential to obtaining a more accurate estimation of these properties using model updating with a discrete Timoshenko beam model :

- **Discrete superstructure and shear:** For a more accurate estimation of parameter  $K_{r,y}$ , it is of importance to incorporate shear deformations into the model and account for irregular stiffness along the height. For the New Orleans, when only the lowest two bending modes in both directions were used to fit the model, the estimates of parameter  $K_{r,y}$  showed low uncertainty, with a Coefficient of Variation of 6.91%. This is an improvement compared to the study conducted by Moretti et al. [1], which obtained a Coefficient of Variation of 46.9% with the uniform Euler-Bernoulli beam model. Therefore, it is of importance to incorporate shear deformations into the model and account for irregular stiffness along the height.
- **Bending stiffness values:** To obtain a more accurate estimation of the structural properties, it is crucial to obtain accurate values of the bending stiffness. The discrete Timoshenko beam models used to approximate the dynamic behavior of the New Orleans and the Delftse Poort were created using their Finite Element (FE) models. However, obtaining values for the bending stiffnesses using the FE models proved challenging. The Steiner approach, which calculated the bending stiffness per segment using the elastic modulus  $E$  and the second moment of inertia  $I$  (calculated with  $\frac{1}{12}bh^3$  and the Steiner rule), assumed fully rigid connections between elements when determining  $I$ , making it impossible to accurately account for the contributions of elements that were not rigidly connected. The FE approach, which used the relative displacement  $\Delta$  of the FE segment to calculate the bending stiffness, did not account for the intermediate relationships between segments, leading to inaccurate estimation of the segments with column structures. These segments exhibit weaker behavior when considered individually compared to when they are part of the overall system due to the additional mass of the other segments.

These challenges in determining bending stiffness values per segment posed a problem for model updating, as the initial structural property ratios were maintained for both high-rise buildings. Maintaining these ratios gave weight to the initial structural property values. These values must be correct. Otherwise, model updating is limited by the incorrect ratios and will not be able to accurately match the measured modal properties. Therefore, it is crucial to obtain accurate values of the bending stiffness.

- **High-quality measured modal properties:** To achieve a more accurate estimation of the parameters  $K_t$  and  $EI$ , it is of importance to obtain measured natural frequencies with errors below 10% and measured mode shapes with errors below 20%. For both the New Orleans and the Delftse Poort,  $K_t$  and  $EI$  showed sensitivity to measurement uncertainties. When errors in the measured natural frequencies of up to 10% and in the measured mode shapes of up to 20% were applied, the values of  $K_t$  and  $EI$  for the New Orleans ranged from 6.41 to 10.0 and from 10.29 to 12.28, respectively. For the Delftse Poort, the values of  $K_t$  and  $EI$  ranged from 7.58 to 10.0 and from 7.93 to 12.38, respectively. The target values for  $K_t$  and  $EI$  were for both cases 9.0 and 10.5, respectively. This posed challenges for model updating, as both parameters required higher bending modes for accurate estimation, which are more prone to errors. Therefore, it is essential to minimize the errors of the measured modal properties to below these thresholds.
- **Lack of pure bending modes:** To obtain a more accurate estimation of the parameters  $EI$  and  $K_t$  for the Delftse Poort, more measured pure bending modes are needed.

The discrete Timoshenko beam model was unable to match the natural frequency of the measured second bending mode in the y-direction after updating. Additionally, the Modal Assurance Criterion (MAC) values indicated that the second bending mode shape in the y-direction was not accurately estimated. The mode shape indicated twisting of the building, which could not be captured by the discrete Timoshenko beam model, as it is only capable of representing pure bending modes.

Furthermore, since the discrete Timoshenko beam model requires a single displacement value per height but multiple sensors were used to measure displacements, the measurements had to be averaged. This averaging process may have led to a mode shape that deviated from the actual behavior, resulting in an inaccurate representation of reality. These issues posed a challenge for model updating with the Delftse Poort, as now only the first measured bending modes in both direction could be accurately applied, which provided alone insufficient information to estimate the parameters  $EI$  and  $K_t$ . More pure bending modes are needed to obtain an accurate estimation of these parameters.

Based on these findings, several recommendations can be made to improve the accuracy and broaden the application of vibration-based model updating in estimating the structural properties of high-rise buildings. The first recommendation addresses the difficulties observed with the discrete superstructure. Maintaining the ratios gave weight to the initial structural property values, which needed to be correct for accurate estimations of the structural properties. It is recommended to allow some flexibility in these ratios, acknowledging that they might not be entirely accurate. Additionally, exploring model updating without keeping these ratios constant would be valuable. However, given the large number of parameters involved, this could potentially lead to an overdetermined problem, as independently adjusting the parameters of a discrete superstructure might result in multiple combinations that fit the measured modal properties equally well.

The second recommendation involves the improvement of the estimates of parameter  $K_{r,y}$ , indicated with a Coefficient of Variation of 6.91%. It remains uncertain whether the improved results of  $K_{r,y}$  are primarily due to the discreteness of the beam, the added shear, or a combination of both. Therefore, it is recommended to perform model updating using both the uniform Timoshenko and the discrete Euler-Bernoulli beam model to compare the results obtained with those obtained with the discrete Timoshenko beam model.

The third recommendation involves torsion, a feature not currently accounted for in the discrete Timoshenko beam model. The discrete Timoshenko beam model was unable to accurately capture the second bending mode in the y-direction for the Delftse Poort, where twisting of the building was observed. This limitation suggests that torsion, which was not included in the current model, could be significant. Incorporating torsion into the model could address this issue by allowing the model to capture the twisting of the building more accurately. Moreover, including torsion in the model would enable the use of measured torsional modes, which could reveal new relationships between input parameters and the model output.

The last recommendation emphasizes the importance of obtaining measured data with minimal errors. It is advised to install additional sensors at various heights to reduce measurement errors. Increasing the number of sensors helps average out errors, leading to more accurate data.

# References

- [1] D. Moretti, A. J. Bronkhorst, and C. P. W. Geurts. “Identification of the structural properties of a high-rise building”. In: (May 2023).
- [2] E. Taciroglu, S.F. Ghahari, and F. Abazarsa. “Efficient model updating of a multi-story frame and its foundation stiffness from earthquake records using a timoshenko beam model”. In: *Soil Dynamics and Earthquake Engineering* 92 (Jan. 2017), pp. 25–35.
- [3] Dr. Garth N. Wells. *The Finite Element Method: An Introduction*. University of Cambridge and Delft University of technology, 2009.
- [4] Olek C. Zienkiewicz, Robert L. Taylor, and J. Z. Zhu. *The Finite Element Method: Its Basis and Fundamentals*. Elsevier, May 2005.
- [5] J. E. Mottershead and M. I. Friswell. “Model Updating In Structural Dynamics: A Survey”. In: *Journal of Sound and Vibration* 167.2 (Oct. 1993), pp. 347–375.
- [6] Sergey E. Lyshevski. *Control Systems Theory with Engineering Applications*. Springer Science & Business Media, June 2001.
- [7] M. I. Friswell and J. E. Mottershead. *Finite Element Model Updating in Structural Dynamics*. Ed. by G. M. L. Gladwell. Vol. 38. Solid Mechanics and its Applications. Springer Netherlands, 1995.
- [8] H. G. Natke, ed. *Identification of Vibrating Structures*. Springer Vienna, 1982.
- [9] B. A. H. Abbas and J. Thomas. “The second frequency spectrum of timoshenko beams”. In: *Journal of Sound and Vibration* 51.1 (Mar. 1977), pp. 123–137.
- [10] Seyedali Mirjalili. *Genetic Algorithm / SpringerLink*. Vol. 780. Springer, June 2018.
- [11] N. A. Z. Abdullah et al. “A review on model updating in structural dynamics”. In: *IOP Conference Series: Materials Science and Engineering* 100.1 (Nov. 2015).
- [12] A. J. Bronkhorst, D. Moretti, and C. P. W. Geurts. “Identification of the Dynamic Properties of the Residential Tower New Orleans”. In: *Experimental Vibration Analysis for Civil Engineering Structures*. Ed. by Zhishen Wu et al. Vol. 224. Springer International Publishing, 2023, pp. 469–479.
- [13] S. F. Ghahari et al. “Blind identification of the Millikan Library from earthquake data considering soil–structure interaction”. In: *Structural Control and Health Monitoring* 23.4 (2016), pp. 684–706.
- [14] R. W. TRAILL-NASH and A. R. COLLAR. “THE EFFECTS OF SHEAR FLEXIBILITY AND ROTATORY INERTIA ON THE BENDING VIBRATIONS OF BEAMS”. In: *The Quarterly Journal of Mechanics and Applied Mathematics* 6.2 (Mar. 1953), pp. 186–222.
- [15] G. R. Bhashyam and G. Prathap. “The second frequency spectrum of Timoshenko beams”. In: *Journal of Sound and Vibration* 76.3 (June 1981), pp. 407–420.
- [16] M. Levinson and D. W. Cooke. “On the two frequency spectra of Timoshenko beams”. In: *Journal of Sound and Vibration* 84.3 (Oct. 1982), pp. 319–326.
- [17] A. J. Bronkhorst and C. P. W. Geurts. “Long-term vibration and wind load monitoring on a high-rise building”. In: *Proceedings ISMA2020, Leuven, Belgium* (2020).
- [18] S.P. Timoshenko. “LXVI. On the correction for shear of the differential equation for transverse vibrations of prismatic bars”. In: *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science* 41.245 (May 1921), pp. 744–746.
- [19] Antonio Cazzani, Flavio Stochino, and Emilio Turco. “On the whole spectrum of Timoshenko beams. Part I: a theoretical revisitation”. In: *Zeitschrift für angewandte Mathematik und Physik* 67.2 (Apr. 2016), p. 24.

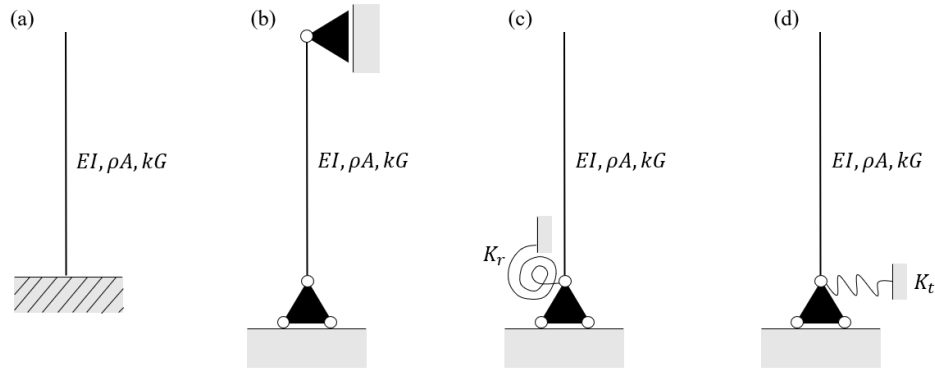
- [20] Karl F. Graff. *Wave motion in Elastic Solids*. Oxford: Dover publications (New York), 1991.
- [21] J.M.J. Spijkers, A.W.C.M. Vrouwenvelder, and E.C. Klaver. *Structural Dynamics CT4140 - Part 1- Structural Vibrations*. Delft University of Technology - Faculty of Civil Engineering and Geosciences, Jan. 2005.
- [22] Angelo Simone. “An Introduction to the Analysis of Slender Structures”. In: *Delft University of Technology, Faculty of Civil Engineering and Geosciences - Structural Mechanics Section - Computational Mechanics Group* (Sept. 2011).
- [23] Rune Brincker, Lingmi Zhang, and Palle Andersen. “Modal identification of output-only systems using frequency domain decomposition”. In: *Smart Materials and Structures* 10.3 (June 2001), p. 441.
- [24] J. R. Wu and Q. S. Li. “Finite element model updating for a high-rise structure based on ambient vibration measurements”. In: *Engineering Structures* 26.7 (June 2004), pp. 979–990.
- [25] *ChatGPT*.
- [26] *Vertalen met DeepL Translate*. nl.



# A

## Timoshenko beam model

The Timoshenko beam, which was first introduced by Timoshenko in 1921 [18], is a model used to describe the dynamic behavior of a beam. The beam model, illustrated in Figure A.1, accounts for both bending and shear deformation [19], and is utilized in many papers published after 1921. Despite the amount of papers published after 1921, however, discussion regarding its exact definition remain, especially due to the existence of a so-called 'second spectrum of modes'.



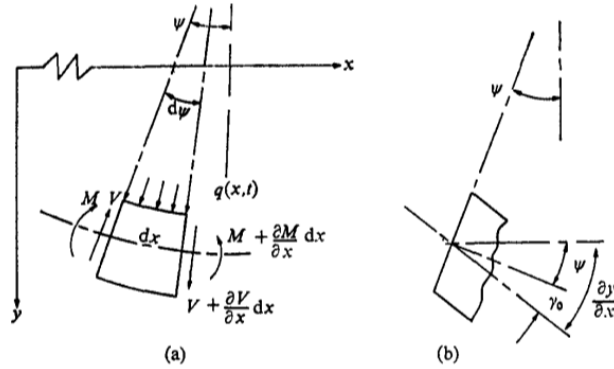
**Figure A.1:** The Timoshenko beam model (1D) with various boundary conditions: (a.) a clamped beam, (b.) a simply supported beam, (c.) a beam supported by a rotational spring and (d.) a beam supported by a translational spring.

Traill-Nash and Collar were the first to indicate the presence of a second spectrum of modes [14]. Their findings caused various responses: Abbas and Thomas argued that, aside from a simply supported beam, no second spectrum of modes exists, stating that earlier conclusions about its existence were misinterpretations [9]. Bhashyam and Prathap provided proof of its existence for various boundary conditions, and developed a methodology to categorize the different modes [15]. Levinson and Cooke claimed that, even for the simply supported Timoshenko beam, only one spectrum of modes exists [16].

The ongoing debate may explain why many papers published since 1921 do not provide a complete solution and often fail to address the potential existence of a second spectrum of modes. To address this gap and to provide a more detailed explanation of the model, this appendix presents the complete derivation of a piece-wise Timoshenko beam. It begins with the governing equations of dynamics, followed by an eigenvalue analysis of the beam model. Additionally, it outlines the boundary and continuity conditions applied. The appendix concludes with a numerical example to verify the procedure and illustrate how the phenomenon of the second spectrum of modes appears.

## A.1. The governing equations of dynamics

To start the derivation of the Timoshenko beam, a reference system is needed. The applied reference system is illustrated in Figure A.2.



**Figure A.2:** The sign convention of the Timoshenko beam of the publication 'Wave motion in Elastic Solids' of Karl F. Graff [20]. This sign convention is used in this Appendix.

In the figure,  $M$  represents the bending moment,  $V$  represents the shear force,  $q(x,t)$  represents the distributed force,  $\frac{\partial y}{\partial x}$  represents the slope of the centroidal axis,  $\gamma_0$  represents the shear strain at centroidal axis, and  $\psi$  represents the rotation of the plane. The curvature of the plane, denoted as  $\frac{1}{R}$ , equals to the first derivative of  $\psi$ . The kinematic equations are written as [20]:

$$\begin{aligned}\gamma_0 &= \frac{\partial y}{\partial x} - \psi \\ \frac{1}{R} &= -\frac{d\psi}{dx}\end{aligned}\tag{A.1}$$

To relate the kinematic equations with the applied loads, constitutive relations are applied. These equations are written as [20]:

$$\begin{aligned}Q &= G \int \gamma dA = kG\gamma_0 A \\ M &= \int y\sigma(y)dA = E \frac{1}{R} \int y^2 dA = -EI \frac{\partial \psi}{\partial x}\end{aligned}\tag{A.2}$$

With these relations,  $\gamma_0$  is used for the entire cross-section. This, however, may result in differences compared to the situation when the variable shear stress is integrated across the section. To correct this difference, a coefficient  $k$ , known as the Timoshenko shear coefficient, is introduced. This coefficient depends on the shape of the cross-section [20].

To connect the applied loads with the internal forces and moments, equilibrium equations are applied. The equilibrium equations in the  $\psi$ - and  $y$ - directions are written as [20]:

$$\begin{aligned}\sum T &= dM - Vdx = \rho I dx \frac{\partial^2 \psi}{\partial t^2} \\ \sum F_y &= dV + q(x,t)dx = ma_y = \rho A dx \frac{\partial^2 y}{\partial t^2}\end{aligned}\tag{A.3}$$

Substituting the kinematic and constitutive relations in these equilibrium equations gives the governing equations for the dynamics of a Timoshenko beam model [20]:

$$\begin{aligned}EI \frac{\partial^2 \psi(x,t)}{\partial x^2} + kGA \left( \frac{\partial y(x,t)}{\partial x} - \psi(x,t) \right) &= \rho I \frac{\partial^2 \psi(x,t)}{\partial t^2} \\ kGA \left( \frac{\partial^2 y(x,t)}{\partial x^2} - \frac{\partial \psi(x,t)}{\partial x} \right) + q(x,t) &= \rho A \frac{\partial^2 y(x,t)}{\partial t^2}\end{aligned}\tag{A.4}$$

The equations are a system of second-order partial differential equations (PDEs).

## A.2. Eigenvalue problem

To find the eigenvalue problem, the system of second-order partial differential equations is reduced to one fourth-order partial differential equation using the method separation of variables. To start this procedure, the governing equations of dynamics are given as a set of homogeneous equations [21]:

$$\begin{aligned} EI \frac{\partial^2 \psi(x, t)}{\partial x^2} + kGA \left( \frac{\partial y(x, t)}{\partial x} - \psi(x, t) \right) - \rho I \frac{\partial^2 \psi(x, t)}{\partial t^2} &= 0 \\ kGA \left( \frac{\partial^2 y(x, t)}{\partial x^2} - \frac{\partial \psi(x, t)}{\partial x} \right) - \rho A \frac{\partial^2 y(x, t)}{\partial t^2} &= 0 \end{aligned} \quad (\text{A.5})$$

In these equations, the term  $q(x, t)$  is set to zero, indicating a non-loaded beam. To decouple the set of equations, the variables  $(x, t)$  and  $\phi(x, t)$  are decomposed into the product of two unknown functions  $x$  and  $t$  [21]:

$$\begin{aligned} y(x, t) &= Y(x)T(t) = Y(x) \sin(\omega t + \phi) \\ \psi(x, t) &= \Psi(x)T(t) = \Psi(x) \sin(\omega t + \phi) \end{aligned} \quad (\text{A.6})$$

The unknown function  $T(t)$  is assumed to have a sinusoidal shape  $\sin(\omega t + \phi)$ . Substituting equations A.6 into equations A.5 yields [21]:

$$\begin{aligned} \left( \frac{d^2 \Psi(x)}{dx^2} + \frac{kGA}{EI} \left( \frac{dY(x)}{dx} - \Psi(x) \right) + \frac{\rho I \omega^2}{EI} \Psi(x) \right) \sin(\omega t + \phi) &= 0 \\ \left( \frac{d^2 Y(x)}{dx^2} - \frac{d\Psi(x)}{dx} + \frac{\rho A \omega^2}{kGA} Y(x) \right) \sin(\omega t + \phi) &= 0 \end{aligned} \quad (\text{A.7})$$

The time function  $\sin(\omega t + \phi)$  is not equal to zero at all times. Therefore, it is required that the term within brackets is independent of time [21]:

$$\begin{aligned} \frac{d^2 \Psi(x)}{dx^2} + \frac{kGA}{EI} \left( \frac{dY(x)}{dx} - \Psi(x) \right) + \frac{\rho I \omega^2}{EI} \Psi(x) &= 0 \\ \frac{d^2 Y(x)}{dx^2} - \frac{d\Psi(x)}{dx} + \frac{\rho A \omega^2}{kGA} Y(x) &= 0 \end{aligned} \quad (\text{A.8})$$

Substituting these equations into each other gives the fourth-order partial differential equation of the Timoshenko beam model [21], known as the eigenvalue problem:

$$EI \frac{d^4 Y(x)}{dx^4} + \frac{\rho \omega^2}{kGA} (EIA + kGAI) \frac{d^2 Y(x)}{dx^2} + \frac{\rho \omega^2}{kGA} (\rho IA \omega^2 - kGA^2) Y(x) = 0 \quad (\text{A.9})$$

## A.3. Eigenvalue analysis

To solve the eigenvalue problem indicated above, an eigenvalue analysis is performed. For convenience, dimensionless parameters are defined to rewrite the equation [2]:

$$\begin{aligned} s^2 &= \frac{EI}{kGAL^2} \\ b^2 &= \frac{\rho A \omega^2 L^4}{EI} \\ R^2 &= \frac{r^2}{L^2} \end{aligned} \quad (\text{A.10})$$

Rewriting equation A.9 with these parameters give A.11:

$$\frac{d^4 \tilde{Y}(\tilde{x})}{d\tilde{x}^4} + (b^2 s^2 + b^2 R^2) \frac{d^2 \tilde{Y}(\tilde{x})}{d\tilde{x}^2} + (b^4 s^2 R^2 - b^2) \tilde{Y}(\tilde{x}) = 0 \quad (\text{A.11})$$

To solve this eigenvalue problem, a general solution is assumed in the form of an exponential function  $\tilde{Y}(\tilde{x}) = \exp \lambda \tilde{x}$ . Implementing this assumption results in the characteristic equation of the system [2]:

$$\lambda^4 + (b^2 s^2 + b^2 R^2) \lambda^2 + (b^4 s^2 R^2 - b^2) = 0 \quad (\text{A.12})$$

Solving this characteristic equation gives four solutions, referred to as the eigensolutions of the problem [2]:

$$\begin{aligned}\lambda_{1,2} &= \pm \sqrt{\frac{-(b^2 s^2 + b^2 R^2) - \sqrt{(b^2 s^2 + b^2 R^2)^2 - 4(b^4 s^2 R^2 - b^2)}}{2}} \\ \lambda_{3,4} &= \pm \sqrt{\frac{-(b^2 s^2 + b^2 R^2) + \sqrt{(b^2 s^2 + b^2 R^2)^2 - 4(b^4 s^2 R^2 - b^2)}}{2}}\end{aligned}\quad (\text{A.13})$$

The roots  $\lambda_{1,2}$  are always imaginary. The roots  $\lambda_{3,4}$ , however, can be real or imaginary, depending on the frequency value. The change from real to imaginary occurs at a certain point in the frequency spectrum, known as the *transition frequency*  $\omega_c$ . The transition frequency is equal to  $\omega_c = \sqrt{\frac{kGA}{\rho I}}$ . Consequently, the frequency spectrum can be separated into two parts, with a special case at the transition frequency  $\omega_c$  [2]:

**Case 1:**  $\omega < \sqrt{\frac{kGA}{\rho I}}$

In this case, the roots  $\lambda_{3,4}$  are real, giving the real eigenfunctions as [2]:

$$\begin{aligned}\tilde{Y}(\tilde{x}) &= A_1 \sin(p\tilde{x}) + A_2 \cos(p\tilde{x}) + A_3 \sinh(q\tilde{x}) + A_4 \cosh(q\tilde{x}) \\ \tilde{\Phi}(\tilde{x}) &= B_1 \sin(p\tilde{x}) + B_2 \cos(p\tilde{x}) + B_3 \sinh(q\tilde{x}) + B_4 \cosh(q\tilde{x})\end{aligned}\quad (\text{A.14})$$

where the  $p$  is equal to  $|\text{Im}(\lambda_1)|$  and the  $q$  is equal to  $\lambda_3$ . The coefficients  $B_1$ ,  $B_2$ ,  $B_3$  and  $B_4$  are related to the coefficients  $A_1$ ,  $A_2$ ,  $A_3$  and  $A_4$  through the equations of A.8 [2]:

$$\begin{aligned}B_1 &= \left( \frac{b^2 s^2}{p} - p \right) A_2 \\ B_2 &= \left( -\frac{b^2 s^2}{p} + p \right) A_1 \\ B_3 &= \left( \frac{b^2 s^2}{q} + q \right) A_4 \\ B_4 &= \left( \frac{b^2 s^2}{q} - q \right) A_3\end{aligned}\quad (\text{A.15})$$

It should be noted that  $\tilde{Y}(\tilde{x}) = Y(x)/L$ ,  $\frac{d\tilde{Y}(\tilde{x})}{d\tilde{x}} = \frac{dY(x)}{dx}$ ,  $\frac{d^2\tilde{Y}(\tilde{x})}{d\tilde{x}^2}/L = \frac{d^2Y(x)}{dx^2}$  and  $\frac{d\tilde{Y}(\tilde{x})}{d\tilde{x}}/L = \frac{dY(x)}{dx}$  [2].

**Case 2:**  $\omega = \sqrt{\frac{kGA}{\rho I}}$

In this case, the roots  $\lambda_{3,4}$  are equal to zero, giving the real eigenfunctions A.16 as [2]:

$$\begin{aligned}\tilde{Y}(\tilde{x}) &= C_1 \sin(p\tilde{x}) + C_2 \cos(p\tilde{x}) + C_3 \tilde{x} + C_4 \\ \tilde{\Phi}(\tilde{x}) &= D_1 \sin(p\tilde{x}) + D_2 \cos(p\tilde{x}) + b^2 s^2 D_3 \tilde{x} + D_4\end{aligned}\quad (\text{A.16})$$

where the  $p$  is equal to  $|\text{Im}(\lambda_1)|$  and  $C_3 = 0$ . The coefficients  $D_1$ ,  $D_2$ ,  $D_3$  and  $D_4$  are related to  $C_1$ ,  $C_2$ ,  $C_3$  and  $C_4$  through the equations of A.8 [2]:

$$\begin{aligned}D_1 &= \left( \frac{b^2 s^2}{p} - p \right) C_2 \\ D_2 &= \left( -\frac{b^2 s^2}{p} + p \right) C_1 \\ D_3 &= C_4\end{aligned}\quad (\text{A.17})$$

**Case 3:**  $\omega > \sqrt{\frac{kGA}{\rho I}}$

In this case, the roots  $\lambda_{3,4}$  are imaginary, giving the real eigenfunctions A.18 as [2]:

$$\begin{aligned}\tilde{Y}(\tilde{x}) &= E_1 \sin(p\tilde{x}) + E_2 \cos(p\tilde{x}) + E_3 \sin(q\tilde{x}) + E_4 \cos(q\tilde{x}) \\ \tilde{\Phi}(\tilde{x}) &= F_1 \sin(p\tilde{x}) + F_2 \cos(p\tilde{x}) + F_3 \sin(q\tilde{x}) + F_4 \cos(q\tilde{x})\end{aligned}\quad (\text{A.18})$$

where the  $p$  is equal to  $|\text{Im}(\lambda_1)|$  and the  $q$  is equal to  $|\text{Im}(\lambda_3)|$ . The coefficients  $F_1$ ,  $F_2$ ,  $F_3$  and  $F_4$  are related to  $E_1$ ,  $E_2$ ,  $E_3$  and  $E_4$  through the equations of A.8 [2]:

$$\begin{aligned} F_1 &= \left( \frac{b^2 s^2}{p} - p \right) E_2 \\ F_2 &= \left( -\frac{b^2 s^2}{p} + p \right) E_1 \\ F_3 &= \left( \frac{b^2 s^2}{p} - p \right) E_4 \\ F_4 &= \left( -\frac{b^2 s^2}{p} + p \right) E_3 \end{aligned} \tag{A.19}$$

## A.4. Boundary and continuity conditions

To fit the eigenfunctions to the actual problem, boundary conditions are applied [22]. These conditions, listed in Table A.1, give information about the geometric character (*kinematic boundary conditions*) or the forces (*dynamic boundary conditions*) of the beam.

Boundary	Conditions	
Hinged	$\tilde{Y}(\tilde{x}) = 0$	$\frac{d\tilde{\Psi}(\tilde{x})}{d\tilde{x}} = 0$
Fixed	$\tilde{Y}(\tilde{x}) = 0$	$\tilde{\Psi}(\tilde{x}) = 0$
Free	$\frac{d\tilde{\Psi}(\tilde{x})}{d\tilde{x}} = 0$	$\tilde{\Psi}(\tilde{x}) - \frac{d\tilde{Y}(\tilde{x})}{d\tilde{x}} = 0$
Rotational	$\tilde{Y}(\tilde{x}) = 0$	$\frac{d\tilde{\Psi}(\tilde{x})}{d\tilde{x}} - \frac{k_r}{\frac{EI}{L}} \tilde{\Psi}(\tilde{x}) = 0$
Translational	$\tilde{\Psi}(\tilde{x}) - \frac{d\tilde{Y}(\tilde{x})}{d\tilde{x}} + \frac{k_t}{\frac{kGA}{L}} \tilde{Y}(\tilde{x}) = 0$	$\tilde{\Psi}(\tilde{x}) = 0$
Rotational + Translational	$\tilde{\Psi}(\tilde{x}) - \frac{d\tilde{Y}(\tilde{x})}{d\tilde{x}} + \frac{k_t}{\frac{kGA}{L}} \tilde{Y}(\tilde{x}) = 0$	$\frac{d\tilde{\Psi}(\tilde{x})}{d\tilde{x}} - \frac{k_r}{\frac{EI}{L}} \tilde{\Psi}(\tilde{x}) = 0$

**Table A.1:** The boundary conditions of a Timoshenko beam [21, 2, 19].

With a piece-wise beam model, it is also necessary to consider continuity conditions. These conditions, listed in Table A.2, explain the connection between the intermediate parts of the beam [22]. They represent the continuity of displacement, forces, and the slope of the beam between the parts.

Boundary	Conditions
Continuity	$\tilde{Y}_{left}(\tilde{x}) = \tilde{Y}_{right}(\tilde{x})$
	$\frac{d\tilde{Y}_{left}(\tilde{x})}{d\tilde{x}} = \frac{d\tilde{Y}_{right}(\tilde{x})}{d\tilde{x}}$
	$\frac{d\tilde{\Psi}_{left}(\tilde{x})}{d\tilde{x}} = \frac{EI_{right}}{EI_{left}} \frac{d\tilde{\Psi}_{right}(\tilde{x})}{d\tilde{x}}$
	$\tilde{\Psi}_{left}(\tilde{x}) - \frac{d\tilde{Y}_{left}(\tilde{x})}{d\tilde{x}} = \frac{kGA_{right}}{kGA_{left}} (\tilde{\Psi}_{right}(\tilde{x}) - \frac{d\tilde{Y}_{right}(\tilde{x})}{d\tilde{x}})$

**Table A.2:** The continuity conditions of a piece-wise Timoshenko beam [22].



the relations between the coefficients using  $\mathbf{A}\mathbf{v} = 0$ . This procedure concludes the derivation of the Timoshenko beam model [2].

## A.6. Numerical example

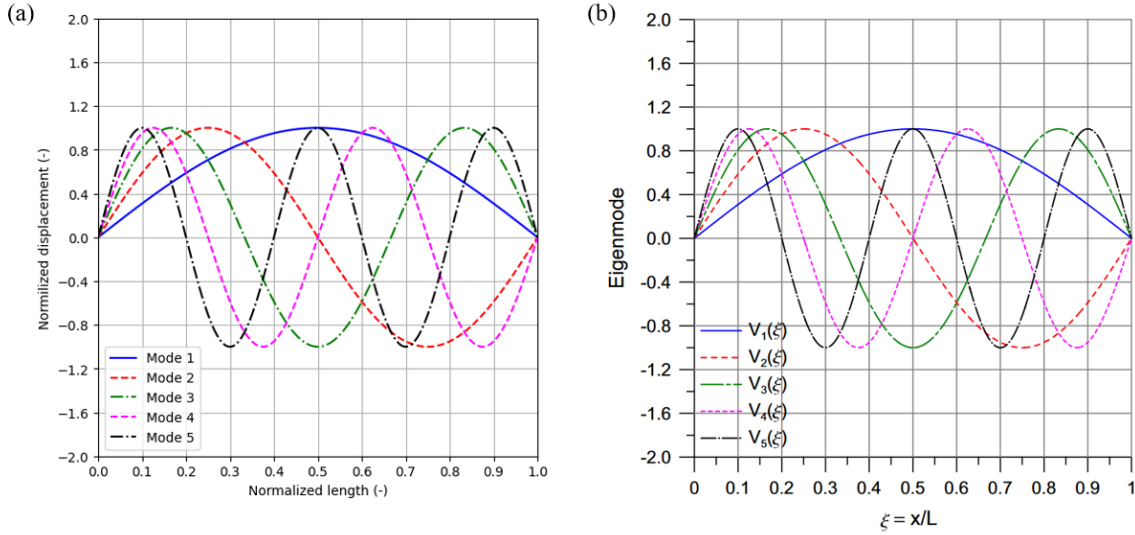
To verify the procedure and to show the phenomenon of the second spectrum of modes, a numerical example is applied. The parameters of this examples are shown in Table A.3. The chosen example is an uniform simply supported beam [19]. A two-piece Timoshenko beam model is used to replicate the uniform beam.

Segment	I / II
$L$ [m]	2
$B$ [m]	0.1
$I$ [m <sup>4</sup> ]	$\frac{1}{120000}$
$E$ [N/m <sup>2</sup> ]	$260e9$
$\nu$ [-]	0.3
$G$ [N/m <sup>2</sup> ]	$100e9$
$\rho$ [kg/m <sup>3</sup> ]	8000

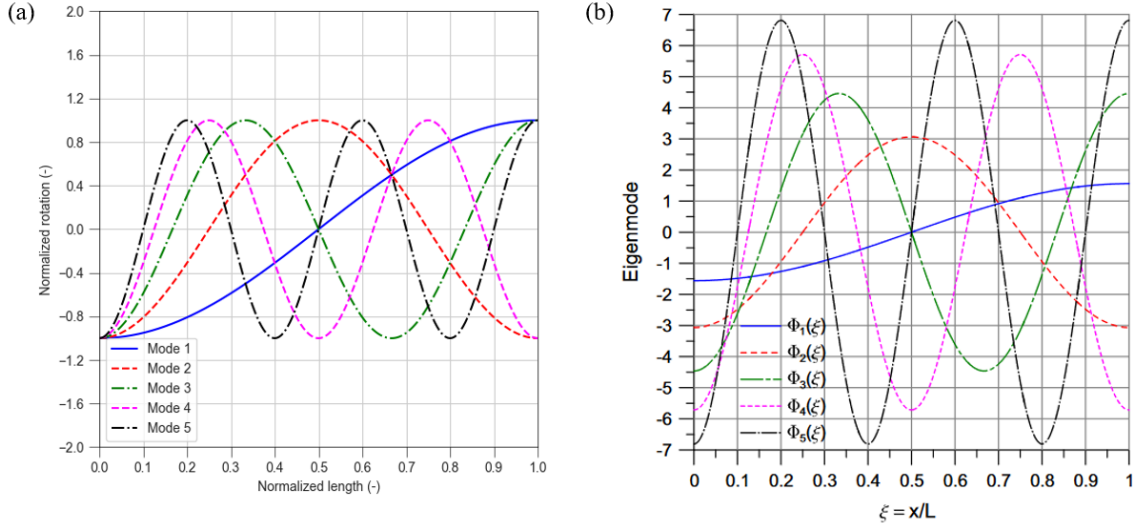
**Table A.3:** The parameters (with units) of a uniform simply supported beam [19]. To verify the code, the results of this beam are reproduced using a two-piece Timoshenko beam. With that beam, each discrete element is given the same parameters as the uniform beam to obtain the same results as for the uniform beam.

**Case 1:**  $\omega < \sqrt{\frac{kGA}{\rho I}}$

In the first part of the spectrum, 25 eigenvalues are obtained, shown in table A.4. The obtained eigenvalues with the two-piece Timoshenko beam model show no differences with the article, confirming the accuracy of the Python code for the first part of the spectrum. In Figures A.3 and A.4, the first five eigenmodes of the spectrum are shown.



**Figure A.3:** A illustration of the first five mode shapes : (a.) shows the results of  $Y(x)$  obtained with the two-piece Timoshenko beam model and (b.) shows the results of the article [19]. The functions  $V_1$ ,  $V_2$ ,  $V_3$ ,  $V_4$  and  $V_5$  are defined in article [19].



**Figure A.4:** A illustration of the first five mode shapes : (a.) shows the results of  $\phi(x)$  obtained with the two-piece Timoshenko beam model and (b.) shows the results of the article [19]. The functions  $\Phi_1$ ,  $\Phi_2$ ,  $\Phi_3$ ,  $\Phi_4$  and  $\Phi_5$  are defined in article [19].

n	Eigenvalue code	Eigenvalue [19]
1	404.35408286	404.3540829
2	1597.56095701	1597.560957
3	3524.34808222	3524.348082
4	6104.92032031	6104.920320
5	9247.9937426	9247.993743
6	12861.93645177	1286.193645
7	16862.12382933	16862.12383
8	21174.58318127	21174.58318
9	25736.94980872	25736.94981
10	30497.85748609	30497.85749
11	35415.60970542	335415.60971
12	40456.65008636	40456.65009
13	45594.10053964	45594.10054
14	50806.48034954	50806.48035
15	56076.63514005	56076.63514
16	61390.86478017	61390.86478
17	66738.22380619	66738.22381
18	72109.96464699	72109.96465
19	77499.09603988	77499.09604
20	82900.0330413	82900.03304
21	88308.3193316	88308.31933
22	93720.40640408	93720.40640
23	99133.47749818	99133.47750
24	104545.30677815	104545.3068
25	109954.14634516	109954.1463

**Table A.4:** The eigenvalues (in  $rad/s$ ) of the first part of the spectrum. The first column contains the results using a two-piece Timoshenko beam model. The second column contains the results of article [19].

**Case 2:**  $\omega = \sqrt{\frac{kGA}{\rho I}}$

The transition frequency has with this example a value of  $\omega_c = 111803.399$  ( $rad/s$ ). The frequency is imposed, not solved. To obtain a non-trivial solution at the transition frequency, the determinant of the coefficient matrix at the transition frequency has to be equal to zero. In the case of the simply

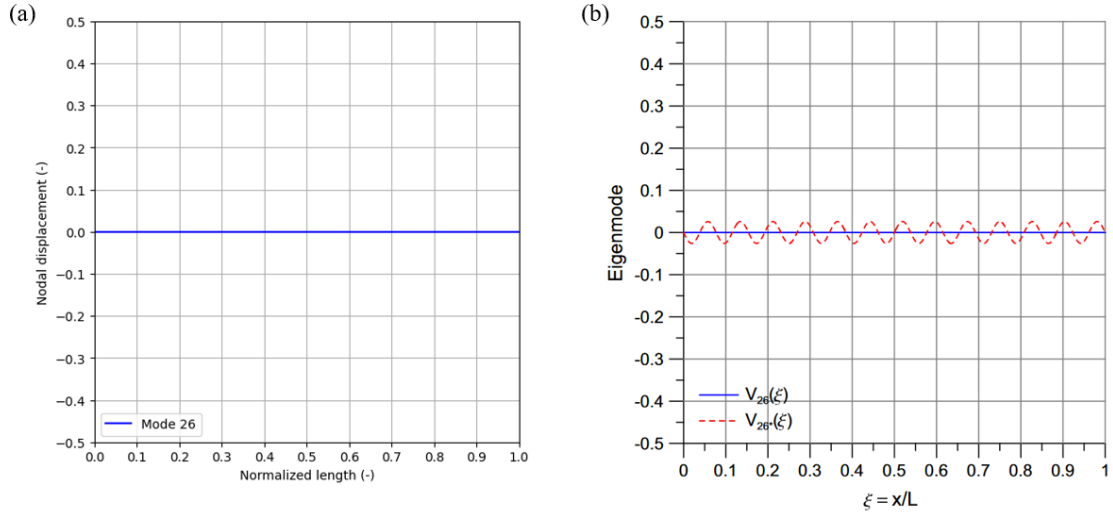


supported beam, the determinant of the coefficient matrix is always zero at the transition frequency:

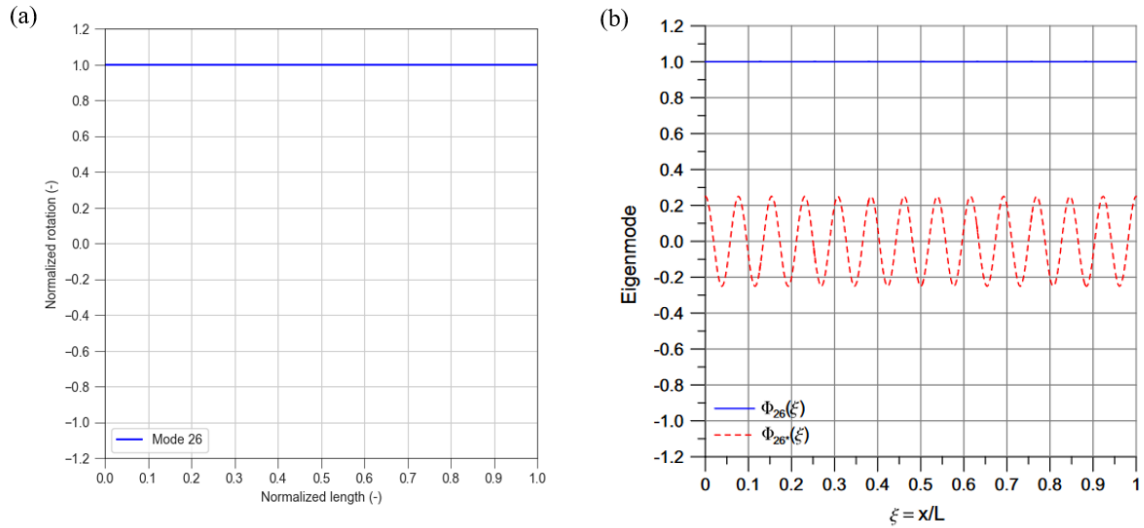
n	Eigenvalue code	Eigenvalue [19]
26	111803.398874989	111803.3989

**Table A.5:** The eigenvalues (in  $rad/s$ ) at the transition frequency. The first column contains the results using a two-piece Timoshenko beam model. The second column contains the results of article [19].

The mode obtained is a *pure shear vibration mode*, as shown in blue in Figure A.5 and A.6, where the transversal displacement is zero. Dependent on the rank of the coefficient matrix, a double eigenvalue can occur, indicated in red in Figures A.5 and A.6 as an example. This is not a pure shear vibration mode.



**Figure A.5:** The plotted mode shape at the transition frequency: (a.) shows the result of  $Y(x)$  obtained with the two-piece Timoshenko beam model and (b.) shows the result of the article [19]. A double eigenvalue can occur at the transition frequency (an example of that is shown in red by the article) [19]. The functions  $V_{28}$  and  $V_{28}^*$  are defined in article [19].



**Figure A.6:** The plotted mode shape at the transition frequency: (a.) shows the result of  $\Psi(x)$  obtained with the two-piece Timoshenko beam model and (b.) shows the result of the article. A double eigenvalue can occur at the transition frequency (an example of that is shown in red by the article) [19]. The functions  $\Phi_{28}$  and  $\Phi_{28}^*$  are defined in article [19].

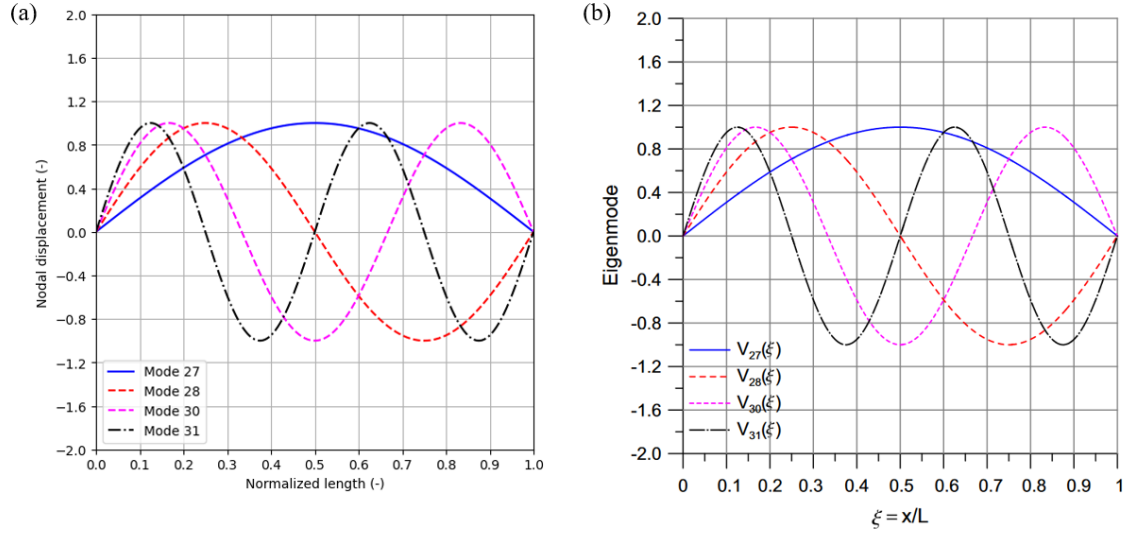
**Case 3:**  $\omega > \sqrt{\frac{kGA}{\rho I}}$

In the second part of the spectrum, the first 24 eigenvalues are obtained and given in table A.6. The obtained eigenvalues with the two-piece Timoshenko beam model show no differences with article [19], confirming the accuracy of the Python code for the second part of the spectrum.

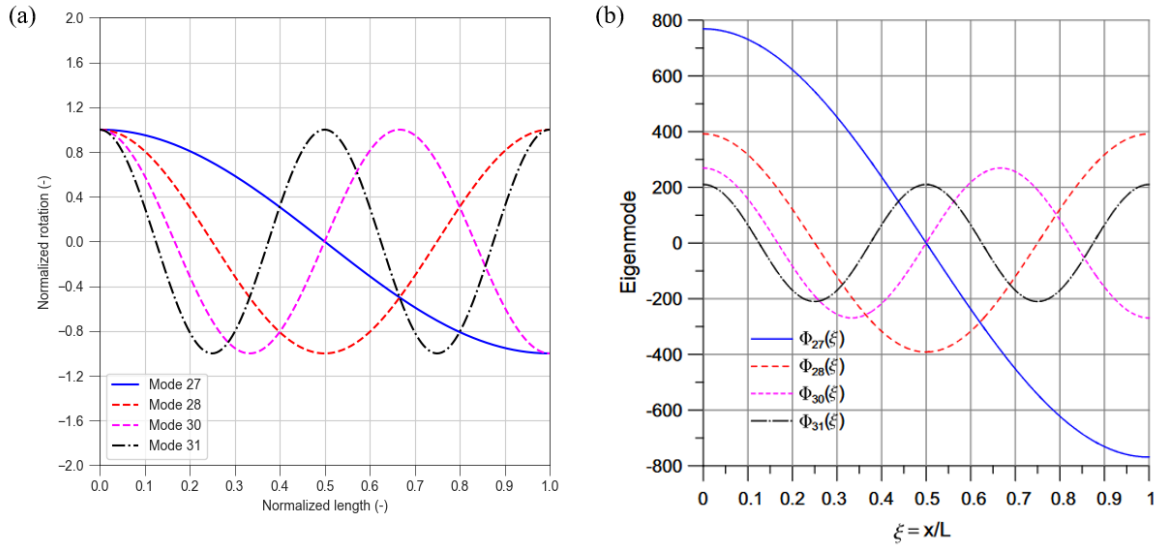
n	Eigenvalue code	Eigenvalue [19]
27	112275.23826886	112275.2383
28	113670.65725684	113670.6573
29	115358.63529931	115358.6353
30	115933.65622448	115933.6562
31	118983.24267197	118983.2427
32	120757.72633166	120757.7263
33	122726.48603937	122726.4860
34	126150.62630367	126150.6263
35	127069.68674956	127069.6867
36	131536.74802825	131536.7480
37	131925.76578166	131925.7658
38	136915.67105325	136915.6711
39	137217.94847072	137217.9485
40	142287.10970443	142287.1097
41	142880.76318863	142880.7632
42	147650.88700103	147650.8870
43	148859.47650081	148859.4765
44	153006.91333603	153006.9133
45	155108.80982765	155108.8098
46	158355.1690327	158355.1690
47	161591.45479719	161591.4548
48	163695.6900625	163695.6901
49	168276.65482621	168276.6548
50	169028.556347	169028.5563

**Table A.6:** The first 24 eigenvalues (in *rad/s*) of the second spectrum. The first column contains the results using a two-piece Timoshenko beam model. The second column contains the results of article [19].

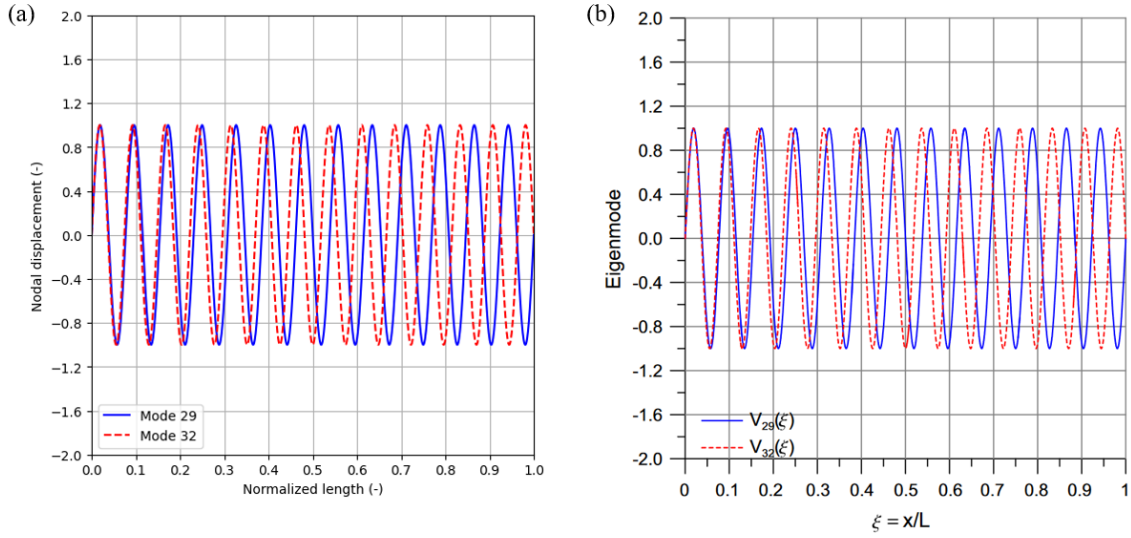
When the first six eigenmodes of the second spectrum are plotted in Figures A.7, A.8, A.9 and A.10, both low and high wavelengths appear in arbitrary order. The presence of lower wavelengths in the second part of the spectrum was initially observed by Traill-Nash and Collar [14], and was referred to as the second spectrum of modes. The shapes of these eigenmodes are identical to those appearing in the first part of the spectrum, however, they are associated with higher frequencies.



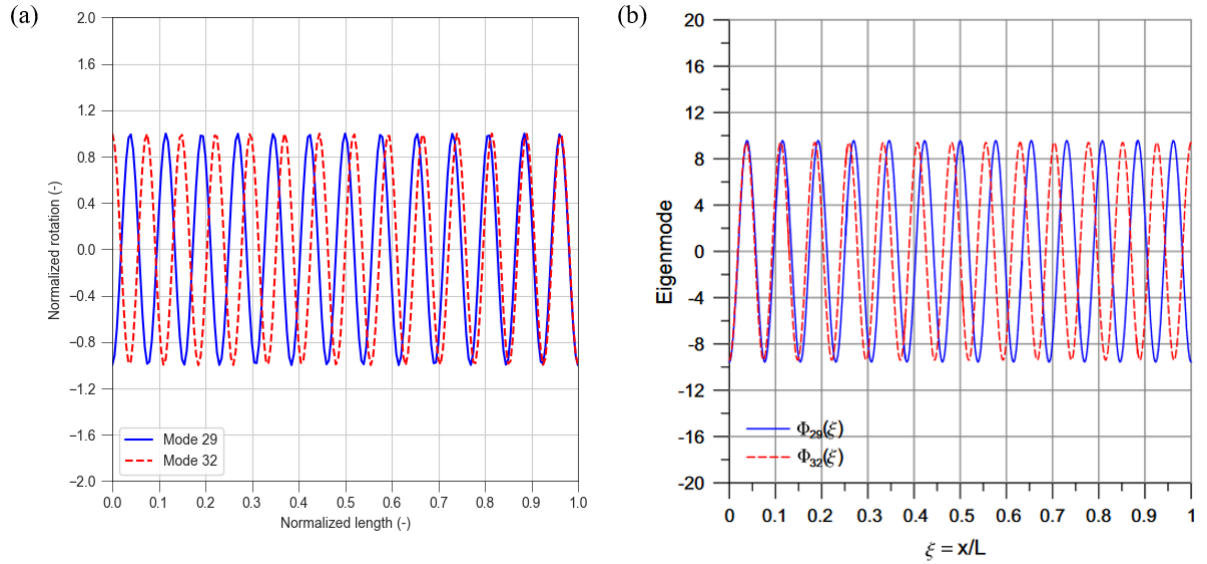
**Figure A.7:** The plotted mode shapes of modes 27, 28, 30 and 31: (a.) shows the results of  $Y(x)$  obtained with the two-piece Timoshenko beam model and (b.) shows the results of the article [19]. The functions  $V_{27}$ ,  $V_{28}$ ,  $V_{30}$ ,  $V_{31}$  are defined in article [19]. These mode shapes are part of the so-called second spectrum of modes, where the shape of the eigemode is identical to those appearing at the first part of the spectrum, however, they are associated with higher wave numbers.



**Figure A.8:** The plotted mode shapes of modes 27, 28, 30 and 31: (a.) shows the results of  $\Psi(x)$  obtained with the two-piece Timoshenko beam model and (b.) shows the results of the article [19]. The functions  $\Phi_{27}$ ,  $\Phi_{28}$ ,  $\Phi_{30}$ ,  $\Phi_{31}$  are defined in article [19]. These mode shapes are part of the so-called second spectrum of modes, where the shape of the eigemode is identical to those appearing at the first part of the spectrum, however, they are associated with higher wave numbers.



**Figure A.9:** The plotted mode shapes of modes 29 and 32: (a.) shows the results obtained with the two-piece Timoshenko beam model and (b.) shows the results of the article [19]. They are not part of the second spectrum of modes as these modes have no equivalent at lower frequencies. The functions  $V_{29}$ , and  $V_{32}$  are defined in article [19].



**Figure A.10:** The plotted mode shapes of modes 29 and 32: (a.) shows the results of  $\Psi(x)$  obtained with the two-piece Timoshenko beam model and (b.) shows the results of the article [19]. They are not part of the second spectrum of modes as these modes have no equivalent at lower frequencies. The functions  $\Phi_{29}$ , and  $\Phi_{32}$  are defined in article [19].

## A.7. Conclusion

With the Appendix, a thorough analysis of the discrete Timoshenko beam model is provided in the context of free vibrations, with a particular focus on the simply supported beam. This analysis brought insights into the nature of the vibration spectrum of a Timoshenko beam model. Despite numerous papers published since 1921, a complete solution has often been lacking, therefore failing to address the potential existence of a second spectrum of modes.

The analysis indicates the presence of a transition frequency, which separate the frequency spectrum into two parts.

The eigenmodes below and above this transition frequency exhibit difference in behavior.

To end, it should be noted that the second spectrum of modes is physically significant. The modes in this spectrum share the same wavelength, even if their frequencies differ. This means, for example, that a high-oscillating force can excite and potentially resonate with these modes. Ignoring these modes could lead to overlooking such phenomena. Only experimentation or possibly 3D modeling can validate the physical significance of these modes.

# B

## Verification Timoshenko beam model

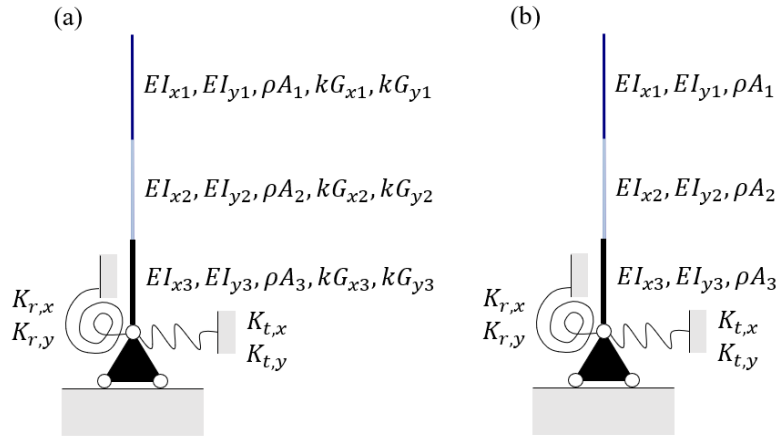
In this research, the discrete Timoshenko beam model is chosen to approximate the dynamic behavior of the residential tower New Orleans and the office tower the Delftse Poort. To verify the code written (shown in Appendix F) and the behavior of the model, in this chapter, two verification studies are conducted. Section B.1 explains the two studies conducted. Section B.2 discusses the results obtained. The chapter ends with the conclusion of the results obtained.

### B.1. Studies

Two studies are conducted. The first study verifies the code written for the discrete Timoshenko beam model and is explained in Section B.1.1. The second study verifies the model behavior and is explained in Section B.1.2.

#### B.1.1. Verification code model

To verify the discrete Timoshenko beam model, the uniform Timoshenko beam model of the study conducted by Taciroglu et al. [2] is replicated using the three-piece Timoshenko beam model. The beam model is shown in Figure B.1. The input parameter values of the uniform Timoshenko beam model are listed in Table B.1. Taciroglu et al. [2] consideration of the frequency dependency in the foundation is taken into account for this verification.



**Figure B.1:** The three-piece Timoshenko beam model (a) and the three-piece Euler-Bernoulli beam model (c).

#### B.1.2. Verification behavior model

To verify the behavior of the chosen model, the natural frequencies obtained with this three-piece Timoshenko beam model are compared with those obtained with a three-piece Euler-Bernoulli beam model, considering different values of the shear modulus  $G$ . The three-piece Euler-Bernoulli is shown

Properties segments	
$L$ [m]	44
$A$ [m <sup>2</sup> ]	843
$I$ [m <sup>4</sup> ]	17750.25
$\rho$ [kg/m <sup>3</sup> ]	400
$E$ [N/m <sup>2</sup> ]	6.60e8
$G$ [N/m <sup>2</sup> ]	2.75e8
$k$ [-]	0.85
Properties boundary	
$K_r$ [Nm/rad]	2.05e12
$k_t$ [N/m]	1.84e10
Properties ground	
$G_s$ [N/m]	2.68e8
$\rho_s$ [kg/m <sup>3</sup> ]	2.7e3

**Table B.1:** The input parameter values applied by Taciroglu et al. [2].

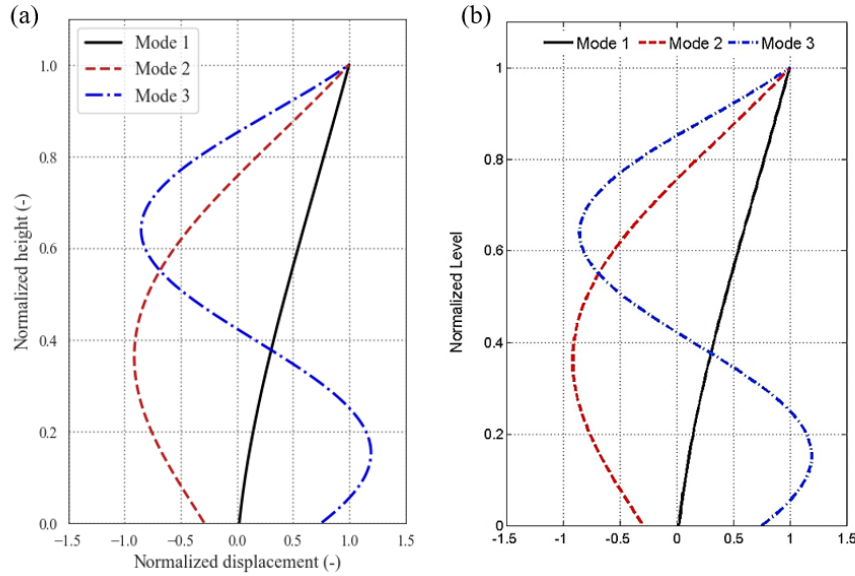
in Figure B.1. For this model, the same input parameter values (Table B.1) are applied. Taciroglu et al. [2] consideration of the frequency dependency is not taken into account for this verification.

## B.2. Results

The results of the two studies conducted are shown below.

### B.2.1. Verification code model

The mode shapes obtained with the three-piece Timoshenko beam model and by Taciroglu et al. [2] are shown in Figure B.2. The natural frequencies obtained with the three-piece Timoshenko beam model and by Taciroglu et al. [2] are listed in Table B.2. The modal properties obtained with the three-piece Timoshenko beam model show similarity to the properties obtained by Taciroglu et al. [2].



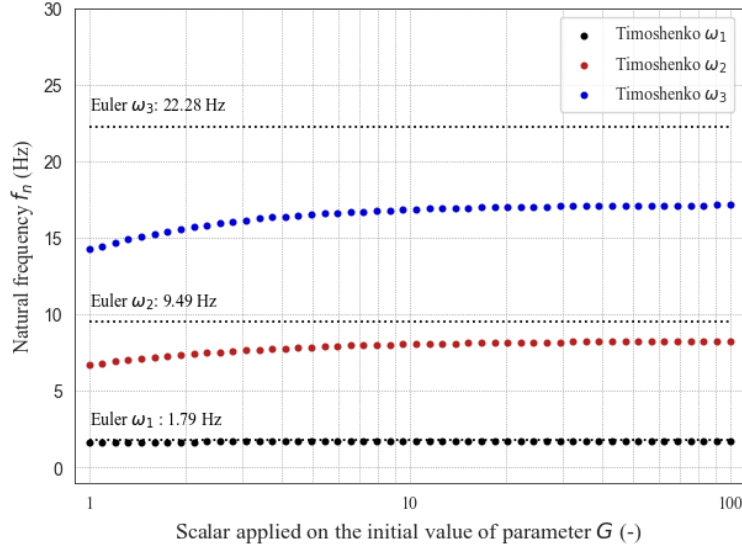
**Figure B.2:** An illustration of the first three mode shapes: (a) shows the results of transverse displacement obtained with the three-piece Timoshenko beam model, and (b) shows the results of the transverse displacement obtained by Taciroglu et al. [2].

Mode	Identified [Hz]	Values of article [2] [Hz]
1	1.608	1.61
2	6.687	6.68
3	14.234	14.22

**Table B.2:** The natural frequencies of the first three modes: (a) shows the results obtained with the three-piece Timoshenko beam model, and (b) shows the results from the article by Taciroglu et al. [2].

### B.2.2. Verification behavior model

The natural frequencies obtained with the Timoshenko and Euler-Bernoulli beam models are shown in Figure B.3.



**Figure B.3:** The natural frequencies of the two models in question. The black dotted lines represent the natural frequencies of the Euler-Bernoulli beam model. The dots represent the natural frequencies of the Timoshenko beam model.

The first natural frequencies show similarities regardless of the value of  $G$ . However, differences can be observed at higher natural frequencies between the Timoshenko beam model and the Euler-Bernoulli beam model. As the value of the shear modulus  $G$  increases, the natural frequencies of the Timoshenko beam model tend to converge towards those of the Euler-Bernoulli beam model. Nonetheless, a constant difference remains, with the greatest difference occurring at the highest bending mode.

## B.3. Conclusion

The results obtained by replicating the article by Taciroglu et al. [2] verify the code of the discrete Timoshenko beam model, as the results obtained with this model closely resemble the natural frequencies and mode shapes obtained by Taciroglu et al. [2].

The results obtained by comparing the Timoshenko beam model with the Euler-Bernoulli beam model verifies the behavior of the discrete Timoshenko beam model. The consistent difference in the natural frequencies are due to the consideration of rotational inertia. The Timoshenko beam model takes rotational inertia in consideration, while the Euler-Bernoulli beam model does not take that into account. Rotational inertia measures the resistance to change in rotational motion, which is expected to be greater at higher modes, as there is more rotation at higher modes. Therefore, it is expected that the difference between the natural frequencies increases at higher modes. Moreover, as the shear modulus  $G$  increases, the natural frequencies of the Timoshenko beam model tend to converge towards those of the Euler-Bernoulli beam model. This is also expected, as with the Euler-Bernoulli beam model, no shear deformation can occur. This means that the Euler-Bernoulli beam model can be seen as having a shear modulus of  $\infty$ .



# C

## Structural Properties

Structural properties refer to intrinsic properties of the system, such as material properties (density, elasticity), geometric properties (dimensions, shapes), or boundary conditions [9], and form the basis of the discrete Timoshenko beam models used in this research. However, the values of the structural properties are not predefined. Therefore, in this appendix, the values are determined using the floor plans and Finite Element (FE) models of the residential tower New Orleans and the office tower Delftse Poort.

Section C.1 explains how each structural property is determined using the FE models and the floor plans of the New Orleans and the Delftse Poort. Sections C.2 and C.3 present the results obtained per building. The appendix ends with a conclusion.

### C.1. Function description

With the floor plans of the buildings, the length  $L$  and the cross-sectional area  $A$  per segment are directly determined. The mass density of each segment  $\rho$  is calculated by considering the dead load  $G$ , variable load  $Q$ , and the volume of the segment:

$$\rho = \frac{G + 0.3 \times Q}{V} \quad (\text{C.1})$$

Two approaches are considered to obtain the bending stiffness. The first approach calculates the bending stiffness per segment  $EI_s$  (with  $s$  being  $x$  or  $y$ ) using the elastic modulus  $E_s$  applied in the FE model and the second moment of area  $I_s$ , calculated using the term  $\frac{1}{12}bh^3$  and the Steiner rule:

$$EI_s = E_s \left( \frac{1}{12}bh^3 + bd^2 \right) \quad (\text{C.2})$$

The FE segment approach considered the segments of the FE model. The FE model of the segment is clamped at the bottom, and a force  $F$  is applied at the top, resulting in a relative displacement  $\Delta$ . The bending stiffness  $EI_s$  is then determined using this displacement  $\Delta$  and Equation C.3:

$$EI_s = \frac{FL^3}{3\Delta} \quad (\text{C.3})$$

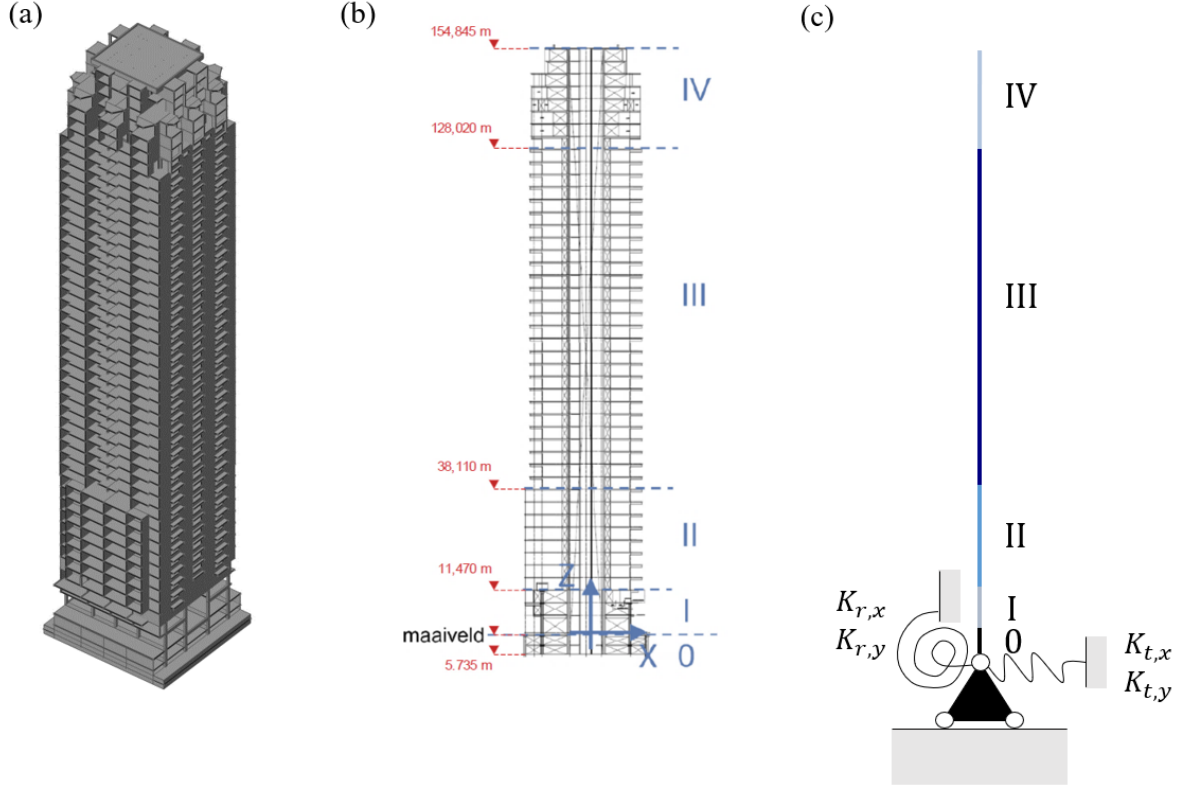
The value of the shear modulus  $G_s$  is determined using the relationship between the shear modulus  $G_s$  and the elastic modulus  $E_s$  of the segment:

$$G_s = \frac{E_s}{2(1 + \nu)} \quad (\text{C.4})$$

The Poisson ratio  $\nu$  is set to 0.2, which is typical for in-situ buildings. The shear coefficient  $k$  is taken as 0.85, in line with the study conducted by E. Taciroglu et al. [2]. The results are shown in Sections C.2 and C.3.

## C.2. The residential tower New Orleans

The discrete Timoshenko beam model applied to approximate the dynamic behavior of the New Orleans is created using the FE model shown in Figure C.1. This FE model is divided into five segments (0 till IV). The segments were defined based on the floor plans. The floors within a segment have the same floor plan in the FE model. Therefore, a five-segment Timoshenko beam model is used to represent the the FE model in Figure C.1. The floor plans of the segments are given in Figures C.2, C.3, C.4, C.5, and C.6.



**Figure C.1:** Pictures of (a) the FE model of the New Orleans, (b) the 5 segments defined for the New Orleans model, and (c) the five-segment Timoshenko beam model.

The calculation of the bending stiffness using the Steiner approach is shown in Sections C.2.1, C.2.2, C.2.3, C.2.4, and C.2.5. The calculation of the other structural properties is shown in Section C.2.7.

## C.2.1. Segment 0

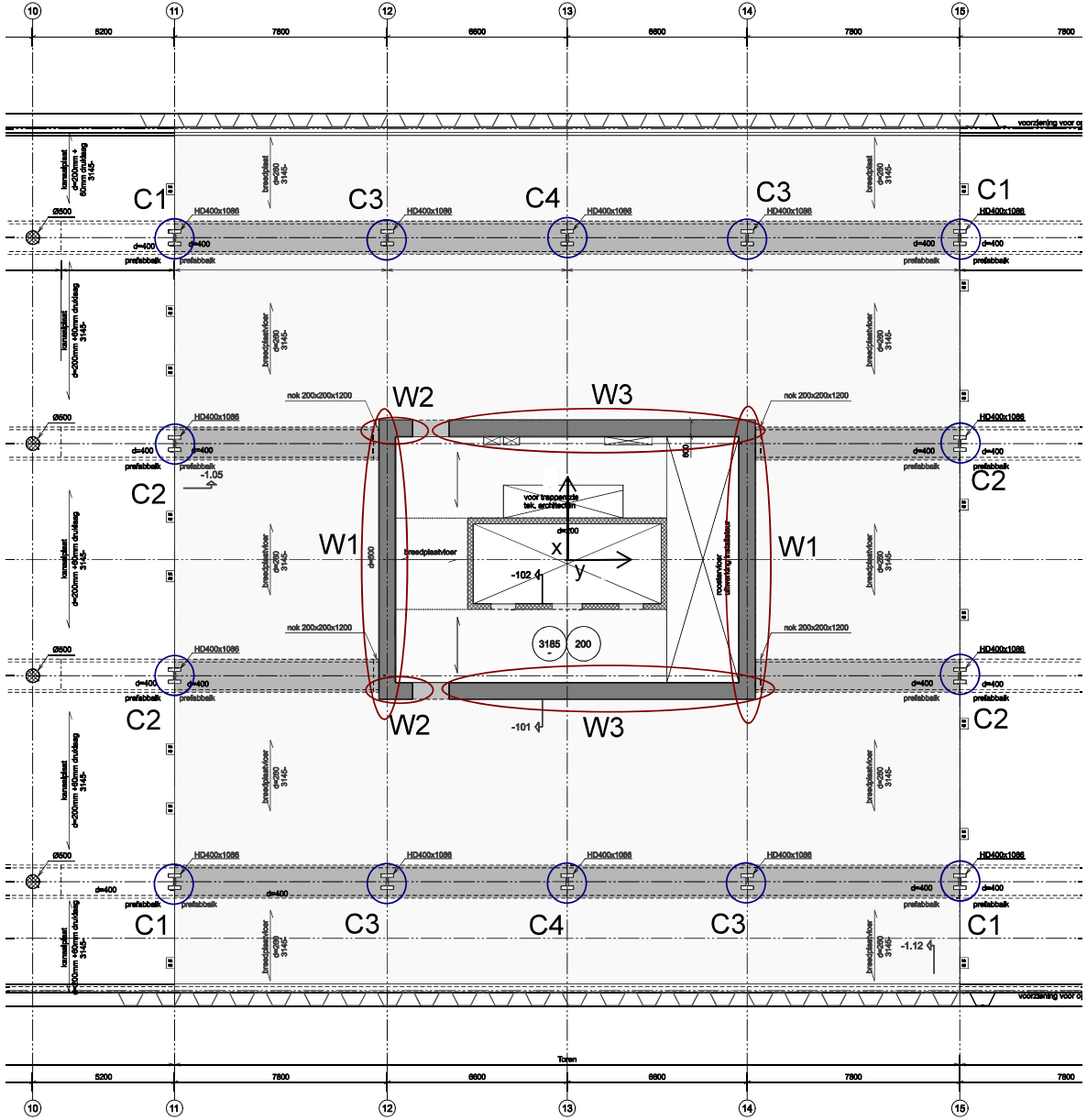


Figure C.2: The floor plan of segment 0.

For the Steiner approach, the second moment of area needs to be calculated. The second moment of area around the  $y$ -axis is denoted as  $I_y$  and is provided in Table C.1. The second moment of area around the  $x$ -axis is denoted as  $I_x$  and is provided in Table C.2. Both values are calculated using the floor plan shown in Figure C.2. The Steiner approach assumes rigid connection between the elements when calculating the second moment of area. Due to the lack of a rigid connection between the core and columns 1 and 2, indicated as C1 and C2 in Figure C.2, these columns are excluded from the calculation of  $I_x$ .

In the FEM model,  $E_{columns}$  is equal to  $210 \times 10^9 \text{ N/m}^2$  and  $E_{walls}$  is equal to  $38.2 \times 10^9 \text{ N/m}^2$ . This gives the bending stiffnesses  $EI_y = 6.71 \times 10^{13} \text{ Nm}^2$  and  $EI_x = 3.19 \times 10^{13} \text{ Nm}^2$ .

Using the FE segment approach, the bending stiffnesses  $EI_y = 5.87 \times 10^{13} \text{ Nm}^2$  and  $EI_x = 2.73 \times 10^{13} \text{ Nm}^2$  are obtained.

Element	N	b [m]	h [m]	d [m]	$\frac{1}{12}bh^3$ [m <sup>4</sup> ]	Steiner [m <sup>4</sup> ]	$I_y$ [m <sup>4</sup> ]
Column 1 flange	8	0.454	0.125	11.65	0.000 591	61.618	61.619
Column 1 web	4	0.078	0.838	11.65	0.0153	35.477	35.492
Column 2 flange	4	0.454	0.125	4.45	0.000 296	4.495	4.495
Column 2 web	2	0.078	0.838	4.45	0.007 64	2.588	2.596
Column 3 flange	8	0.454	0.125	11.65	0.000 591	61.618	61.619
Column 3 web	4	0.078	0.838	11.65	0.0153	35.477	35.492
Column 4 flange	8	0.454	0.125	11.65	0.000 296	61.618	30.809
Column 4 web	4	0.078	0.838	11.65	0.007 64	35.477	17.746
Wall 1	2	0.6	8.4	0	59.270	0	59.270
Wall 2	2	1.295	0.6	4.75	0.0466	35.062	35.492
Wall 3	2	10.595	0.6	4.75	0.381	286.859	287.241
						<b>Total</b>	631.489

**Table C.1:** The second moment of area  $I_y$  of segment 0.

Element	N	b [m]	h [m]	d [m]	$\frac{1}{12}bh^3$ [m <sup>4</sup> ]	Steiner [m <sup>4</sup> ]	$I_x$ [m <sup>4</sup> ]
Column 3 flange	8	0.125	0.454	6.6	0.0078	19.776	19.784
Column 3 web	4	0.838	0.078	6.6	0.000 13	11.386	11.386
Wall 1	2	8.9	0.6	6.6	0.320	439.221	465.541
Wall 2	2	0.6	1.295	5.803	0.217	52.331	52.548
Wall 3	2	0.6	10.595	1.453	118.933	26.842	145.775
						<b>Total</b>	695.035

**Table C.2:** The second moment of area  $I_x$  of segment 0.

### C.2.2. Segment I

The calculated second moment of area  $I_y$  is provided in Table C.3. The calculated second moment of area  $I_x$  is provided in Table C.4. Both are calculated using the floor plan shown in Figure C.3.

Element	N	b [m]	h [m]	d [m]	$\frac{1}{12}bh^3$ [m <sup>4</sup> ]	Steiner [m <sup>4</sup> ]	$I_y$ [m <sup>4</sup> ]
Column 1 flange	8	0.454	0.125	11.65	0.000 591	61.618	61.619
Column 1 web	4	0.078	0.838	11.65	0.0153	35.477	35.492
Column 2 flange	4	0.454	0.125	4.45	0.000 296	4.495	4.495
Column 2 web	2	0.078	0.838	4.45	0.007 64	2.588	2.596
Column 3 flange	8	0.454	0.125	11.65	0.000 591	61.618	61.619
Column 3 web	4	0.078	0.838	11.65	0.0153	35.477	35.492
Column 4 flange	8	0.454	0.125	11.65	0.000 296	61.618	30.809
Column 4 web	4	0.078	0.838	11.65	0.007 64	35.477	17.746
Wall 1	2	0.6	8.4	0	29.635	0	59.270
Wall 2	1	1.295	0.6	4.75	0.0233	17.531	17.554
Wall 3	1	10.595	0.6	4.75	0.191	143.4298	143.621
Wall 4	1	13.2	0.6	4.75	0.238	178.695	178.933
						<b>Total</b>	649.246

**Table C.3:** The second moment of area  $I_y$  of segment I.

In the FEM model,  $E_{columns}$  is equal to  $210 \times 10^9$  N/m<sup>2</sup> and  $E_{walls}$  is equal to  $38.2 \times 10^9$  N/m<sup>2</sup>. This gives the bending stiffnesses  $EI_y = 6.77 \times 10^{13}$  Nm<sup>2</sup> and  $EI_x = 3.31 \times 10^{13}$  Nm<sup>2</sup>. Using the FE segment approach the bending stiffnesses  $EI_y = 3.66 \times 10^{13}$  Nm<sup>2</sup> and  $EI_x = 3.26 \times 10^{13}$  Nm<sup>2</sup> are obtained.

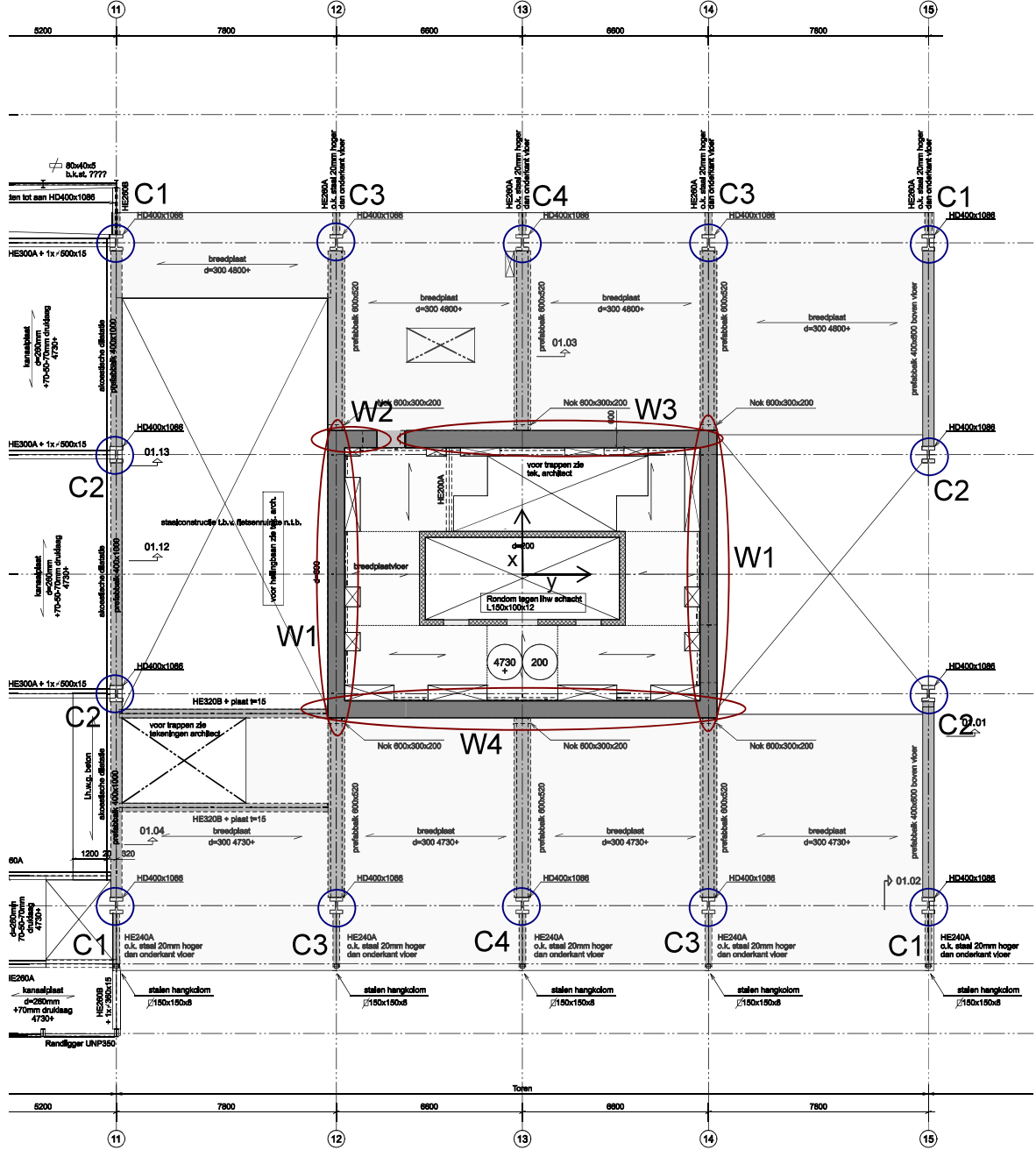


Figure C.3: The floor plan of segment I.

Element	N	b [m]	h [m]	d [m]	$\frac{1}{12}bh^3$ [m <sup>4</sup> ]	Steiner [m <sup>4</sup> ]	$I_x$ [m <sup>4</sup> ]
Column 3 flange	8	0.125	0.454	6.6	0.0078	19.776	19.784
Column 3 web	4	0.838	0.078	6.6	0.00013	11.386	11.386
Wall 1	2	8.9	0.6	6.6	0.320	439.221	465.541
Wall 2	1	0.6	1.295	5.803	0.108	26.165	26.274
Wall 3	1	0.6	10.595	1.453	59.467	13.421	72.888
Wall 4	1	0.6	13.2	0	114.9984	0	131.719
						<b>Total</b>	<b>727.592</b>

Table C.4: The second moment of area  $I_x$  of segment I.

### C.2.3. Segment II

The calculated second moment of area  $I_y$  is provided in Table C.5. The calculated second moment of area  $I_x$  is provided in Table C.6. Both are calculated using the floor plan shown in Figure C.4.

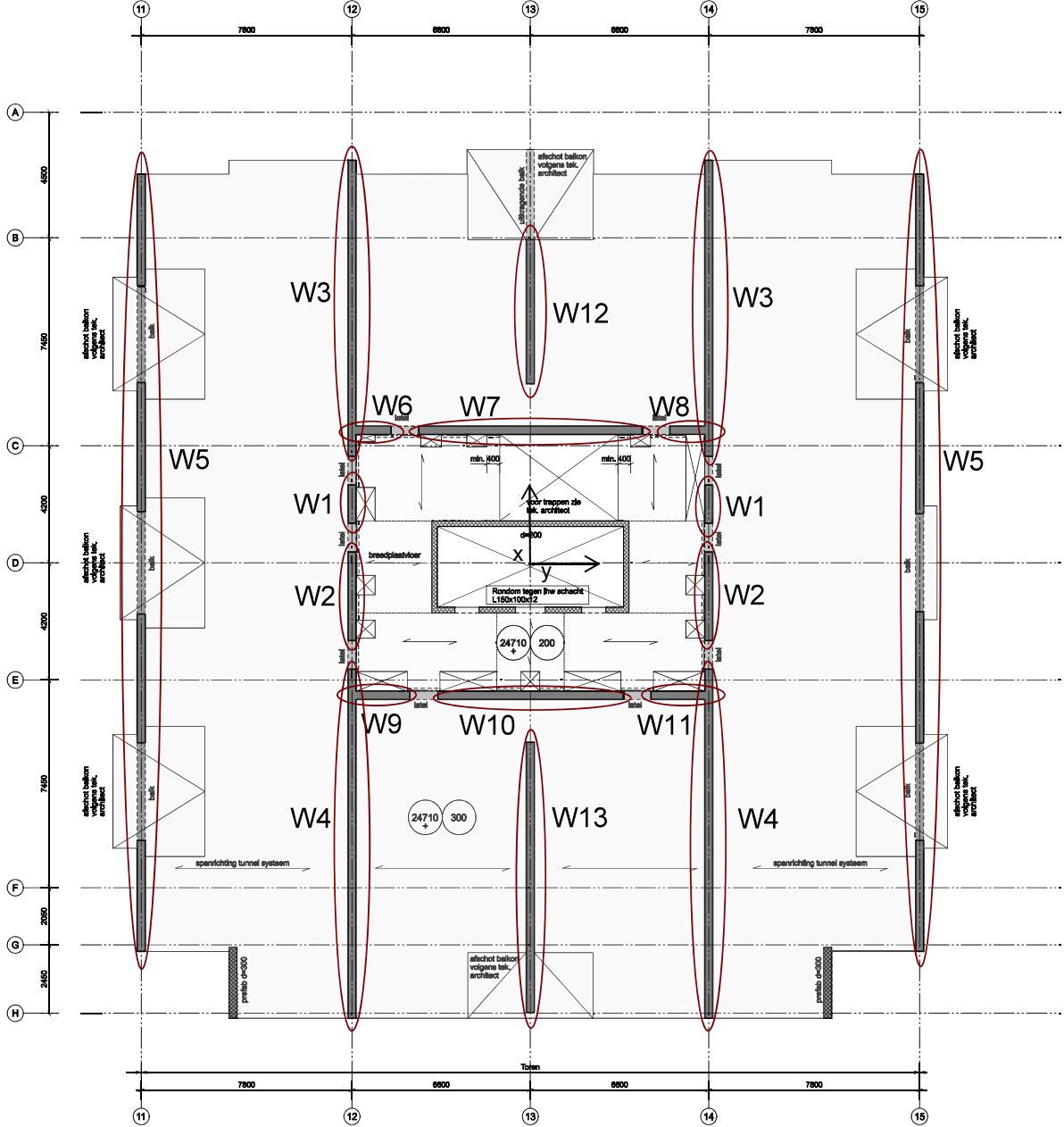


Figure C.4: The floor plan of segment II.

In the FEM model,  $E_{columns}$  is equal to  $210 \times 10^9 \text{ N/m}^2$  and  $E_{walls}$  is equal to  $38.2 \times 10^9 \text{ N/m}^2$ . This gives the bending stiffnesses  $EI_y = 1.22 \times 10^{14} \text{ Nm}^2$  and  $EI_x = 3.08 \times 10^{13} \text{ Nm}^2$ . Using the FE segment approach, Besix obtained the bending stiffnesses  $EI_y = 4.76 \times 10^{13} \text{ Nm}^2$  and  $EI_x = 1.04 \times 10^{14} \text{ Nm}^2$ .

Element	N	b [m]	h [m]	d [m]	$\frac{1}{12}bh^3$ [m <sup>4</sup> ]	Steiner [m <sup>4</sup> ]	$I_y$ [m <sup>4</sup> ]
Wall 1	2	0.3	1.36	2.041	0.126	3.402	3.528
Wall 2	2	0.3	3.19	1.124	1.629	2.421	4.050
Wall 3	2	0.3	10.59	9.08	59.366	523.815	583.181
Wall 4	2	0.3	12.54	10.121	98.620	770.704	869.324
Wall 5	2	0.3	27.86	0	1081.218	0	1081.218
Wall 6	1	1.295	0.3	4.75	0.003	8.766	8.768
Wall 7	1	7.968	0.3	4.75	0.018	54.055	54.073
Wall 8	1	1.295	0.3	4.75	0.003	8.766	8.768
Wall 9	1	1.995	0.3	4.75	0.004	13.504	13.508
Wall 10	1	6.625	0.3	4.75	0.015	44.843	44.858
Wall 11	1	1.995	0.3	4.75	0.004	13.504	13.508
Wall 12	1	0.3	5.16	9.003	3.425	125.350	128.775
Wall 13	1	0.3	9.66	11.253	22.501	366.784	389.285
						<b>Total</b>	<b>3202.844</b>

**Table C.5:** The second moment of area  $I_y$  of segment II.

Element	N	b [m]	h [m]	d [m]	$\frac{1}{12}bh^3$ [m <sup>4</sup> ]	Steiner [m <sup>4</sup> ]	$I_x$ [m <sup>4</sup> ]
Wall 1	2	1.36	0.3	6.6	0.0061	35.571	35.577
Wall 2	2	3.19	0.3	6.6	0.0144	83.478	83.493
Wall 3	2	10.59	0.3	6.6	0.0477	276.754	276.802
Wall 4	2	12.54	0.3	6.6	0.0564	327.772	327.828
Wall 6	1	0.3	1.295	5.803	0.0543	13.083	13.137
Wall 7	1	0.3	7.968	0	12.647	0	12.647
Wall 8	1	0.3	1.295	5.803	0.0543	13.083	13.137
Wall 9	1	0.3	1.995	5.453	0.1985	17.797	17.995
Wall 10	1	0.3	6.625	0	7.2694	0	7.2694
Wall 11	1	0.3	1.995	5.453	0.1985	17.797	17.995
Wall 12	1	5.16	0.3	0	0.0116	0	0.012
Wall 13	1	9.66	0.3	0	0.0217	0	0.022
						<b>Total</b>	<b>805.913</b>

**Table C.6:** The second moment of area  $I_x$  of segment II.

### C.2.4. Segment III

The calculated second moment of area  $I_y$  is provided in Table C.7. The calculated second moment of area  $I_x$  is provided in Table C.8. Both are calculated using the floor plan shown in Figure C.5.

Element	N	b [m]	h [m]	d [m]	$\frac{1}{12}bh^3$ [m <sup>4</sup> ]	Steiner [m <sup>4</sup> ]	$I_y$ [m <sup>4</sup> ]
Wall 1	2	0.3	3.495	3.153	2.135	20.847	22.982
Wall 2	2	0.3	3.195	1.198	1.631	2.751	4.382
Wall 3	4	0.3	9.530	9.515	86.552	1035.361	1121.913
Wall 4	2	0.3	5.155	9.003	6.849	250.7	257.550
Wall 5	2	0.3	27.860	0	1081.218	0	1081.218
Wall 6	2	0.3	1.095	4.353	0.066	12.449	12.515
Wall 7	1	1.295	0.3	4.75	0.003	8.766	8.768
Wall 8	1	10.595	0.3	4.75	0.024	71.715	71.739
Wall 9	1	1.995	0.3	4.75	0.004	13.504	13.508
Wall 10	1	9.895	0.3	4.75	0.022	66.977	66.999
						<b>Total</b>	<b>2661.574</b>

**Table C.7:** The second moment of area  $I_y$  of segment III.

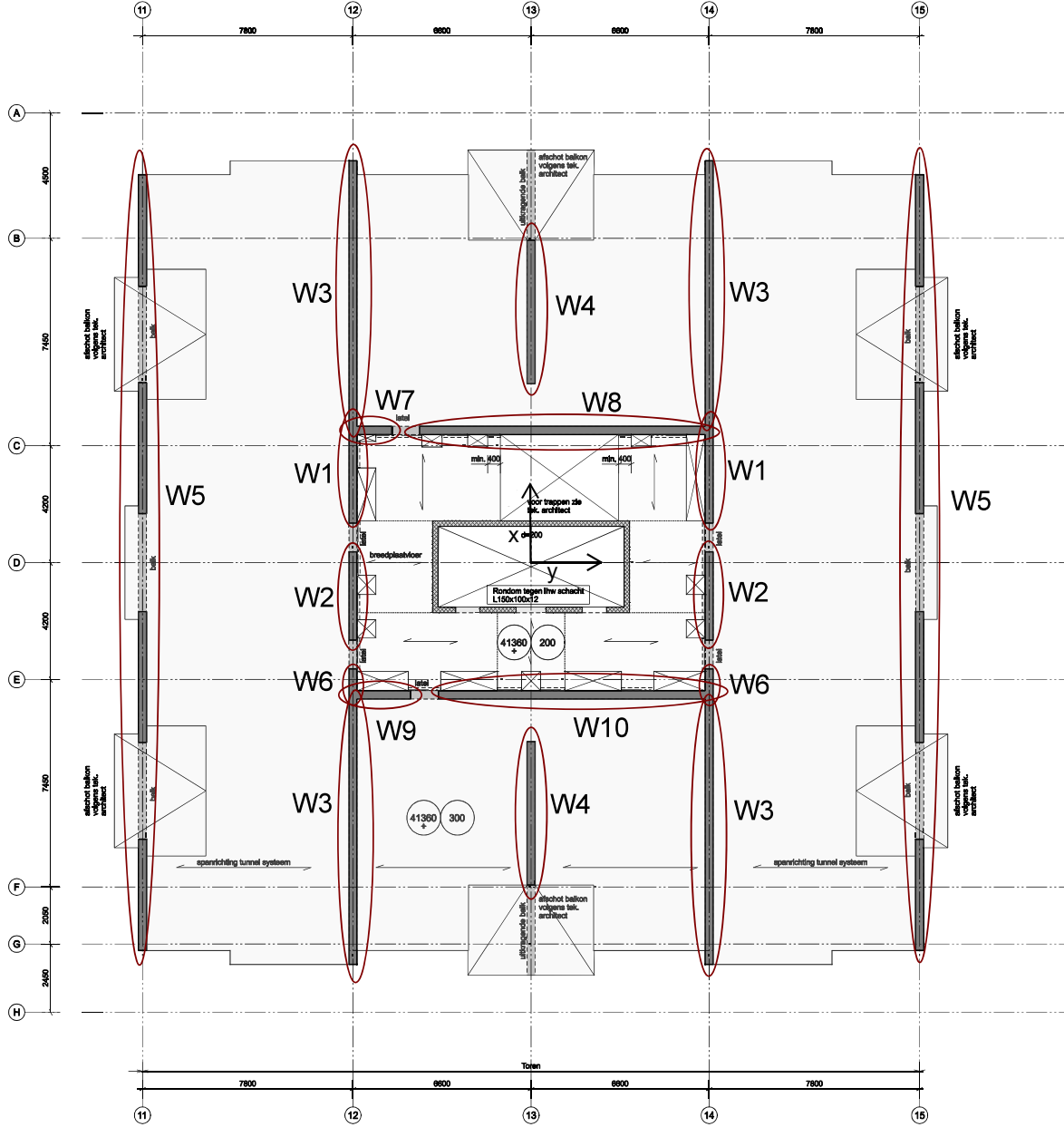


Figure C.5: The floor plan of segment III.

In the FEM model,  $E_{columns}$  is equal to  $210 \times 10^9 \text{ N/m}^2$  and  $E_{walls}$  is equal to  $38.2 \times 10^9 \text{ N/m}^2$ . This gives the bending stiffnesses  $EI_y = 1.02 \times 10^{14} \text{ Nm}^2$  and  $EI_x = 3.07 \times 10^{13} \text{ Nm}^2$ .

Using the FE segment approach, Besix obtained the bending stiffnesses  $EI_y = 4.59 \times 10^{13} \text{ Nm}^2$  and  $EI_x = 8.92 \times 10^{13} \text{ Nm}^2$ .



Element	N	b [m]	h [m]	d [m]	$\frac{1}{12}bh^3$ [m <sup>4</sup> ]	Steiner [m <sup>4</sup> ]	$I_x$ [m <sup>4</sup> ]
Wall 1	2	3.495	0.3	6.6	0.0157	91.345	91.361
Wall 2	2	3.195	0.3	6.6	0.0144	83.504	83.5189
Wall 3	4	9.530	0.3	9.515	0.0858	498.152	498.238
Wall 4	2	5.155	0.3	9.003	0.0232	0	0.0232
Wall 6	2	1.095	0.3	6.6	0.0049	28.619	28.624
Wall 7	1	0.3	1.295	5.803	0.0543	13.083	13.137
Wall 8	1	0.3	10.595	1.453	29.733	6.710	36.444
Wall 9	1	0.3	1.995	5.453	0.1985	17.797	17.995
Wall 10	1	0.3	9.895	1.803	24.221	9.650	33.870
						<b>Total</b>	803.211

**Table C.8:** The second moment of area  $I_x$  of segment III.**C.2.5. Segment IV**

The calculated second moment of area  $I_y$  is provided in Table C.7. The calculated second moment of area  $I_x$  is provided in Table C.8. Both are calculated using the floor plan shown in Figure C.6.

Element	N	b [m]	h [m]	d [m]	$\frac{1}{12}bh^3$ [m <sup>4</sup> ]	Steiner [m <sup>4</sup> ]	$I_y$ [m <sup>4</sup> ]
Wall 1	1	0.3	3.490	5.59	1.063	32.717	33.779
Wall 2	1	0.3	5.561	0	4.299	0	4.2993
Wall 3	2	0.3	6.744	7.216	15.336	210.699	226.035
Wall 4	1	0.3	6.921	4.880	8.288	49.456	57.744
Wall 5	1	0.3	3.194	6.6	0.0072	41.739	41
Wall 6	2	13.2	0.3	4.75	0.0594	178.695	178.754
Wall 7	4	0.3	1.775	11.476	0.559	280.494	281.053
Wall 8	2	0.3	1.48	9.849	0.162	86.072	86.233
						<b>Total</b>	870.054

**Table C.9:** The second moment of area  $I_y$  of segment IV.

Element	N	b [m]	h [m]	d [m]	$\frac{1}{12}bh^3$ [m <sup>4</sup> ]	Steiner [m <sup>4</sup> ]	$I_x$ [m <sup>4</sup> ]
Wall 1	1	3.490	0.3	6.6	0.0079	45.607	45.615
Wall 2	1	5.561	0.3	6.6	0.0125	72.671	72.684
Wall 3	2	6.744	0.3	9.515	0.0303	176.261	176.292
Wall 4	1	6.921	0.3	6.6	0.0156	90.444	90.459
Wall 5	1	3.194	0.3	6.6	0.0072	41.739	41
Wall 6	2	0.3	13.2	0	114.998	0	114.998
Wall 7	4	1.775	0.2	6.6	0.0047	61.855	61.860
Wall 8	2	1.479	0.3	6.6	0.0067	38.655	38.662
						<b>Total</b>	803.211

**Table C.10:** The second moment of area  $I_x$  of segment IV.

In the FEM model,  $E_{columns}$  is equal to  $210 \times 10^9$  N/m<sup>2</sup> and  $E_{walls}$  is equal to  $32.8 \times 10^9$  N/m<sup>2</sup>. This gives the bending stiffnesses  $EI_y = 3.32 \times 10^{13}$  Nm<sup>2</sup> and  $EI_x = 2.45 \times 10^{13}$  Nm<sup>2</sup>.

Using the FE segment approach, Besix obtained the bending stiffnesses  $EI_y = 2.73 \times 10^{13}$  Nm<sup>2</sup> and  $EI_x = 2.97 \times 10^{13}$  Nm<sup>2</sup>.

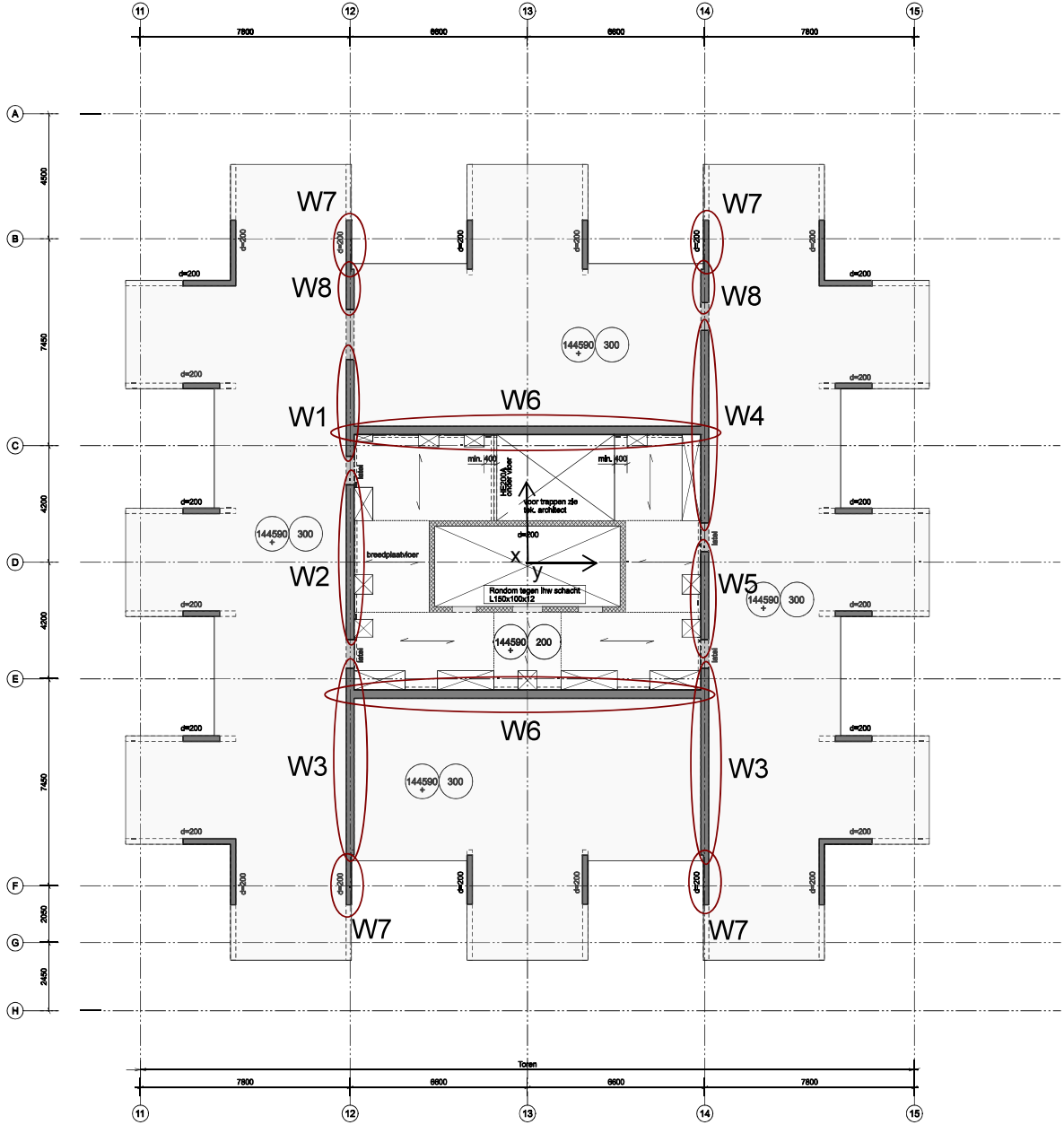


Figure C.6: The floor plan of segment IV.

### C.2.6. Bending stiffness

The obtained bending stiffnesses are given in Table C.11. Additionally, the displacements at the top, using either a five-segment beam model with these bending stiffnesses or the FE model, are presented, when a force of 10,000 kN is applied and the base of the model is fixed.

The displacement in the y-direction obtained using the bending stiffnesses of Steiner approach  $EI_x$  shows significant differences. This could be due to not including columns 1 and 2 and wall 5 in  $I_x$ . Even when there is no rigid connection between these structural elements and the core, it can still be expected that these elements contribute to the rotational resistance. The actual contribution of these elements lies somewhere between the extremes of full inclusion and exclusion, but this contribution cannot be calculated using Steiner.

The displacement of the FE segment approach in the y-direction shows a closer resemblance to the displacement obtained with the Steiner approach. However, the stiffnesses are not as expected. Since the floor plans do not change significantly, it is unexpected to have such differences between the bending stiffnesses of the different segments. This approach fails to account for the intermediate relationships between the different segments (0 through IV) of the building. Segments 0 and I have column structures, which exhibit weaker behavior when considered individually than when integrated into the overall system. This is due to the additional mass from other segments. Therefore, considering these segment individually will lead to an inaccurate assessment of their bending stiffnesses.

Therefore, the bending stiffnesses of the Steiner approach are applied, with a correction factor included to account for the displacement differences between the FE model and five-segment beam using these segment stiffnesses. Since the FE model is stiffer in the y-direction, the segment stiffnesses  $EI_x$  are multiplied by  $\frac{887}{516}$ . Since the FE model is weaker in the x-direction, the segment stiffnesses  $EI_y$  are multiplied by  $\frac{344}{394}$ .

	$EI_x$ [Nm <sup>2</sup> ]		$EI_y$ [Nm <sup>2</sup> ]		<b>FEM</b>	
	Steiner	FE segment	Steiner	FE segment		
0	3.19e13	2.37e13	6.71e13	5.87e13	516	394
I	3.31e13	3.62e13	6.77e13	3.66e13		
II	3.08e13	1.04e14	1.22e14	4.76e13		
III	3.07e13	8.92e13	1.04e14	4.59e13		
IV	2.45e13	2.97e13	2.85e13	2.73e13		
$\Delta u$ [mm]	887	472	344	613		

**Table C.11:** The bending stiffnesses and the total displacements determined with the different approaches.

### C.2.7. Structural properties

The dead load and variable load provided by Besix are shown in Table C.12. The obtained structural properties are shown in Table C.13. The  $E$  modulus of the concrete is used as the  $E$  of the segments, as the building is almost entirely made of concrete. The second moment of area  $I$  of each segment is determined by dividing the bending stiffness by this  $E$  modulus.

Segment	Dead load [kN]	Variable load *0.3 [kN]
0	83621	1626
I	42088	1584
II	95562	3937
III	292545	11787
IV	76016	3058

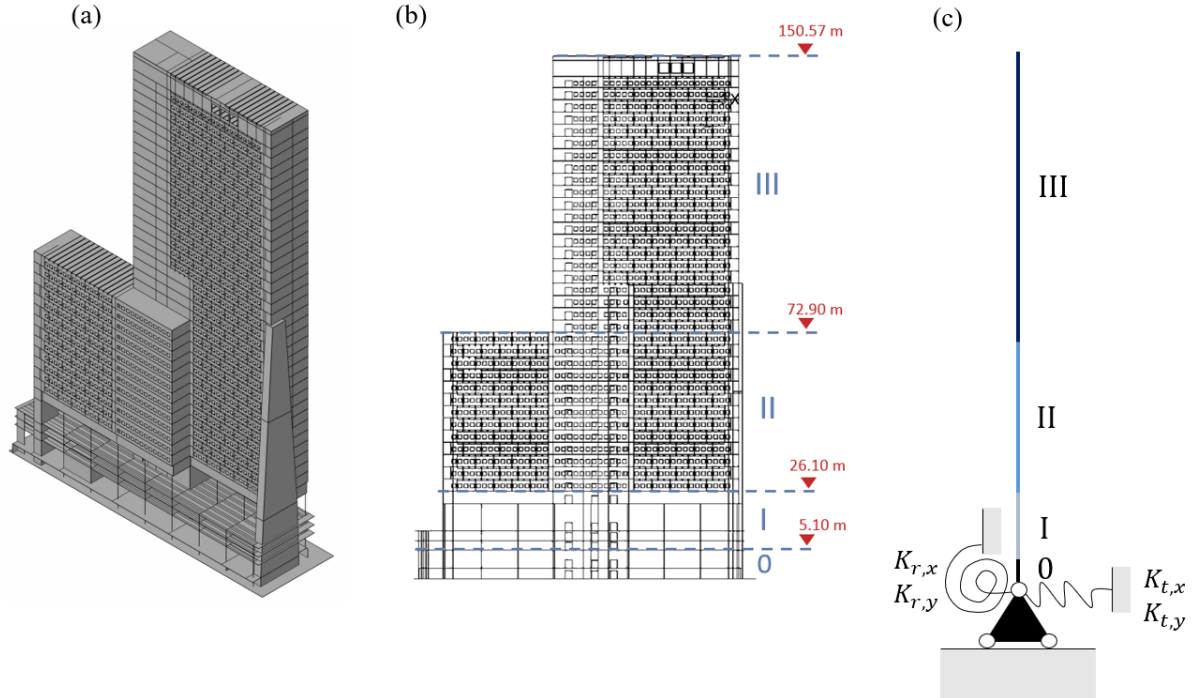
**Table C.12:** The dead and variable load of the residential tower New Orleans.

Segment	0	I	II	III	IV
$L$ [m]	5.735	11.47	26.640	89.91	26.825
$A$ [m <sup>2</sup> ]	784	784	784	784	784
$I_x$ [m <sup>4</sup> ]	1444	1500	1393	1389	1293
$I_y$ [m <sup>4</sup> ]	1532	1548	2796	2324	760
$\rho$ [kg/m <sup>3</sup> ]	1933	495	486	440	383
$E$ [N/m <sup>2</sup> ]	38.2e9	38.2e9	38.2e9	38.2e9	32.8e9
$\nu$ [-]	0.2	0.2	0.2	0.2	0.2
$G$ [N/m <sup>2</sup> ]	15.9e9	15.9e9	15.9e9	15.9e9	13.7e9
$k$ [-]	0.85	0.85	0.85	0.85	0.85
<b>Boundary</b>					
$K_{r,x}$ [GNm/rad]	2380	$K_{t,x}$ [GN/m]	19		
$K_{r,y}$ [GNm/rad]	2625	$K_{t,y}$ [GN/m]	4		

**Table C.13:** The structural property values obtained with the floor plans and FE model of the New Orleans.

### C.3. The office tower the Delfste Poort

The discrete Timoshenko beam model chosen to approximate the dynamic behavior of the Delftse Poort is created using the Finite Element (FE) model of the building. This FE model, shown in Figure 6.1, is divided into four segments. The segments were defined based on the floorplans. Therefore, a four-segment Timoshenko beam model is used to represent the FE model in Figure 6.1. The floor plans representing the segments are shown in Figures C.8, C.9, and C.10. Since segments 0 and I have many similarities, the bending stiffness of both segments, determined using the Steiner approach, is calculated using the same floor plan. This is shown in Sections C.3.1, C.3.2, and C.3.3. The bending stiffness determined using the FE segment approach is also provided in those sections. The calculation of the other structural properties is shown in Section C.3.5.



**Figure C.7:** Pictures of (a) the FE model of the Delftse Poort, (b) the 4 segments defined for the Delftse Poort model, and (c) the four-segment Timoshenko beam model.

### C.3.1. Segment 0 and I

The calculated second moment of area  $I_x$  is provided in Table C.14. The calculated second moment of area  $I_y$  is provided in Table C.15. The percentage indicates the degree of closeness of the wall. The calculation takes this factor into account. Both are calculated using the floor plan shown in Figure C.8.

Element	N	b [m]	h [m]	d [m]	$\frac{1}{12}bh^3$ [m <sup>4</sup> ]	Steiner [m <sup>4</sup> ]	$I_x$ [m <sup>4</sup> ]
Wall 1	1	3.2	0.3	11.54	0.072	127.887	127.894
Wall 2	1	3.2	0.3	5.44	0.072	28.430	28.437
Wall 3	1	0.3	5.4	8.69	3.937	122.389	126.326
Wall 4	1	0.3	4.4	9.19	2.130	111.528	113.658
Wall 5	1	0.5	14.4	3.01	124.416	65.151	189.567
Wall 6	1	0.3	14.4	3.01	74.645	39.090	113.740
Wall 7	1	0.3	5.4	0.11	3.937	0.019	3.956
Wall 8	1	0.5	14.4	3.01	124.416	65.151	189.567
Wall 9	1	2.62	0.3	0.158	0.006	0.020	0.026
Wall 10	1	0.5	14.4	3.01	124.416	65.151	189.567
Wall 11 (90%)	1	23.0	0.3	4.34	0.047	117.071	117.112
Wall 12 (75%)	1	23.0	0.3	4.36	0.039	98.29	98.33
Wall 13	1	0.3	5.9	1.54	5.134	4.208	9.343
Wall 14	1	0.3	2.6	4.34	0.006	14.705	14.710
Wall 15	1	1.6	0.3	1.208	0.0036	0.701	0.704
Wall 16	1	0.4	4.85	6.97	3.803	94.163	97.966
Wall 17	1	0.4	2.6	2.76	0.586	7.911	8.497
Wall 18	1	0.4	8.5	0.19	20.470	0.125	20.596
Wall 19	1	0.4	13.35	2.67	79.309	37.980	117.289
Wall 20	1	3.1	0.4	2.67	0.017	23.014	23.031
Wall 21	1	3.1	0.4	9.64	0.017	115.278	115.295
Column 1	1	1.24	1.24	2.608	0.197	10.491	10.656
Column 2	1	1.24	1.24	10.99	0.197	185.776	185.973
Column 3	1	1.24	1.24	2.608	0.197	10.491	10.656
Column 4	1	1.24	1.24	10.99	0.197	185.776	185.973
Column 5	1	1.24	1.24	2.608	0.197	10.491	10.656
Column 6	1	1.24	1.24	10.99	0.197	185.776	185.973
Column 7	1	1.24	1.24	9.81	0.197	147.916	148.113
Column 8	1	1.24	1.24	10.99	0.197	185.776	185.973
Column 9	1	1.24	1.24	9.81	0.197	147.916	148.113
Column 10	1	1.24	1.24	10.99	0.197	185.776	185.973
Column 11	1	1.24	1.24	9.81	0.197	147.916	148.113
Column 12	1	1.24	1.24	3.79	0.197	22.108	22.305
Column 13	1	1.24	1.24	9.81	0.197	147.916	148.113
Column 12	1	1.24	1.24	3.79	0.197	22.108	22.305
Column 15	1	1.24	1.24	9.81	0.197	147.916	148.113
Column 16	1	1.24	1.24	9.81	0.197	147.916	148.113
						<b>Total</b>	3600.730

**Table C.14:** The second moment of area  $I_x$  of segment 0 and I.

Element	N	b [m]	h [m]	d [m]	$\frac{1}{12}bh^3$ [m <sup>4</sup> ]	Steiner [m <sup>4</sup> ]	$I_y$ [m <sup>4</sup> ]
Wall 1	1	0.3	3.2	48.66	0.819	2272.864	2273.684
Wall 2	1	0.3	3.2	48.66	0.819	2272.864	2273.684
Wall 3	1	5.4	0.3	50.12	0.012	4067.458	4067.47
Wall 4	1	4.4	0.3	47.21	0.010	2941.702	2941.712
Wall 5	1	14.4	0.5	17.61	0.150	2232.211	2232.361
Wall 6	1	14.4	0.3	8.71	0.032	327.556	327.589
Wall 7	1	5.4	0.3	5.81	0.012	54.641	54.652
Wall 8	1	14.4	0.5	4.21	0.15	127.471	127.621
Wall 9	1	0.3	2.62	2.65	0.450	5.510	5.960
Wall 10	1	14.4	0.5	4.89	0.15	172.333	172.483
Wall 11 (90%)	1	0.3	23.0	6.357	273.758	251.007	524.764
Wall 12 (75%)	1	0.3	23.0	6.357	228.131	209.172	437.303
Wall 13	1	5.9	0.3	34.49	0.013	2105.808	2105.821
Wall 14	1	0.3	2.6	35.64	0.439	990.894	991.334
Wall 15	1	0.3	1.6	36.34	0.102	633.968	634.070
Wall 16	1	4.85	0.4	35.74	0.026	2478.380	2478.406
Wall 17	1	2.6	0.4	35.74	0.014	1328.616	1328.630
Wall 18	1	8.5	0.4	37.34	0.045	4741.134	4741.179
Wall 19	1	13.35	0.4	38.39	0.071	7871.013	7871.084
Wall 20	1	0.4	3.1	37.04	0.993	1701.448	1702.441
Wall 21	1	0.4	3.1	37.04	0.993	1701.448	1702.441
Column 1	1	1.24	1.24	49.56	0.197	3776.285	3776.482
Column 2	1	1.24	1.24	38.76	0.197	2309.714	2309.911
Column 3	1	1.24	1.24	38.76	0.197	2309.714	2309.911
Column 4	1	1.24	1.24	27.96	0.197	1201.835	1202.032
Column 5	1	1.24	1.24	27.96	0.197	1201.835	1202.032
Column 6	1	1.24	1.24	17.16	0.197	452.646	452.646
Column 7	1	1.24	1.24	17.16	0.197	452.646	452.646
Column 8	1	1.24	1.24	6.36	0.197	62.149	62.346
Column 9	1	1.24	1.24	6.36	0.197	62.149	62.346
Column 10	1	1.24	1.24	4.44	0.197	30.343	30.541
Column 11	1	1.24	1.24	4.44	0.197	30.343	30.541
Column 12	1	1.24	1.24	15.24	0.197	357.229	357.426
Column 13	1	1.24	1.24	15.24	0.197	357.229	357.426
Column 14	1	1.24	1.24	26.04	0.197	1042.806	1042.806
Column 15	1	1.24	1.24	26.04	0.197	1042.806	1042.806
Column 16	1	1.24	1.24	36.84	0.197	2087.075	2087.272
						<b>Total</b>	55 774.650

**Table C.15:** The second moment of area  $I_y$  of segment 0 and I.

In the FEM model,  $E_{walls}$  is equal to  $11 \times 10^9$  N/m<sup>2</sup>. This gives the bending stiffnesses  $EI_x = 40 \times 10^{12}$  Nm<sup>2</sup> and  $EI_y = 614 \times 10^{12}$  Nm<sup>2</sup>. Using the FE segment approach, the bending stiffnesses  $EI_x = 0.26 \times 10^{12}$  Nm<sup>2</sup> and  $EI_y = 1.04 \times 10^{12}$  Nm<sup>2</sup> are obtained.

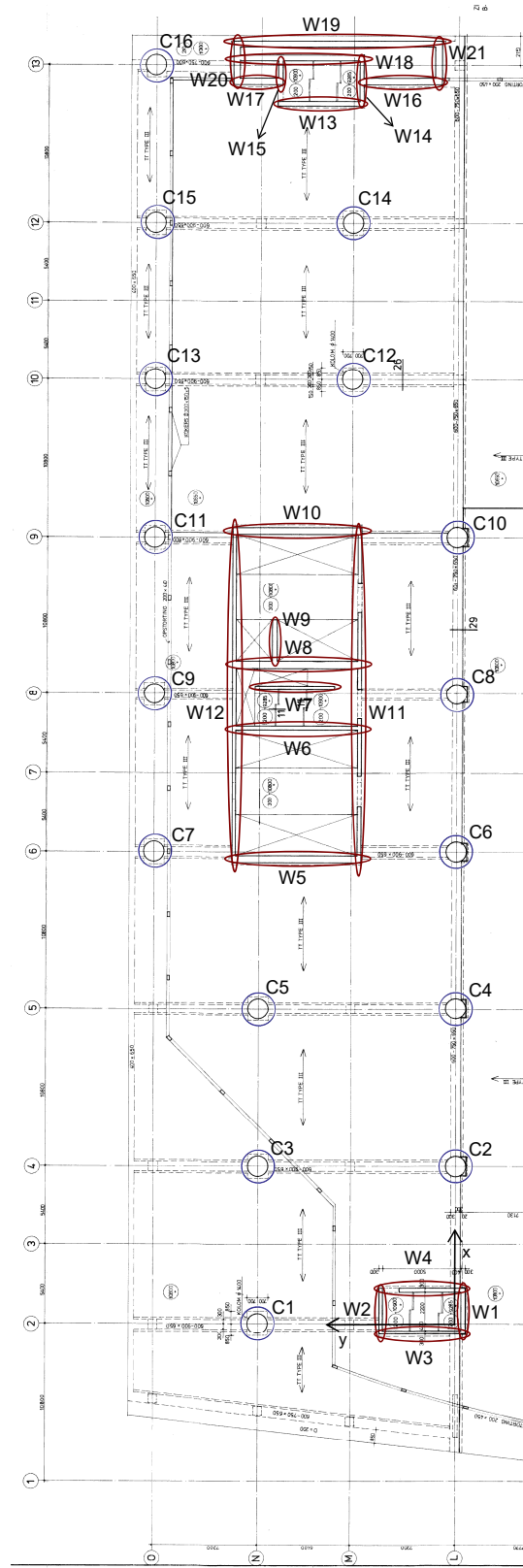


Figure C.8: The floorplan of segment 0 and I.

### C.3.2. Segment II

The calculated second moment of area  $I_x$  is provided in Table C.16. The calculated second moment of area  $I_y$  is provided in Table C.17. The percentage indicates the degree of closeness of the wall. The calculation takes this factor into account. Both are calculated using the floor plan shown in Figure C.9.

Element	N	b [m]	h [m]	d [m]	$\frac{1}{12}bh^3$ [m <sup>4</sup> ]	Steiner [m <sup>4</sup> ]	$I_x$ [m <sup>4</sup> ]
Wall 1 (45%)	1	32.3	0.3	3.50	0.033	53.367	53.399
Wall 2 (45%)	1	55.3	0.3	11.20	0.056	936.746	936.801
Wall 3	1	0.3	14.4	3.85	74.650	64.088	138.737
Wall 4	1	0.3	5.4	8.35	3.937	112.995	116.931
Wall 5 (90%)	1	0.5	21.9	0.10	393.878	0.102	393.982
Wall 6 (60%)	1	22.0	0.3	10.70	0.030	453.242	453.272
Wall 7 (45%)	1	32.9	0.4	10.65	0.079	671.482	671.561
Wall 8 (75%)	1	22.0	0.3	4.00	0.037	79.265	79.302
Wall 9 (45%)	1	32.0	0.3	4.00	0.032	69.177	69.209
Wall 10 (75%)	1	22.0	0.3	4.70	0.037	109.270	109.307
Wall 11	1	2.62	0.3	1.85	0.006	2.685	2.691
Wall 12 (85%)	1	0.5	14.6	3.35	105.754	68.615	174.368
Wall 13 (90%)	1	0.5	21.6	0.25	377.914	0.615	378.529
Wall 14	1	0.3	5.5	1.10	4.159	2.002	6.162
Wall 15	1	0.4	14.3	3.30	97.474	62.229	159.703
Wall 16	1	0.4	4.3	6.45	2.650	71.593	74.243
Wall 17	1	0.4	13.0	2.10	73.233	22.968	96.201
Wall 18	1	0.7	0.4	4.15	0.003	4.819	4.822
Wall 19	1	2.3	0.4	8.40	0.012	64.940	64.953
						<b>Total</b>	3984.172

**Table C.16:** The second moment of area  $I_x$  of segment II.

Element	N	b [m]	h [m]	d [m]	$\frac{1}{12}bh^3$ [m <sup>4</sup> ]	Steiner [m <sup>4</sup> ]	$I_y$ [m <sup>4</sup> ]
Wall 1 (45%)	1	0.3	32.3	33.39	379.106	4862.781	5241.886
Wall 2 (45%)	1	0.3	55.3	21.89	1902.514	3578.712	5481.226
Wall 3	1	14.4	0.3	49.39	0.032	10 539.980	10 540.020
Wall 4	1	5.4	0.3	46.49	0.012	3502.008	3502.020
Wall 5 (90%)	1	21.9	0.5	16.89	0.205	2812.836	2813.041
Wall 6 (60%)	1	0.2	22.0	5.64	159.72	126.165	285.885
Wall 7 (45%)	1	0.4	32.9	21.81	534.169	2815.806	3349.976
Wall 8 (75%)	1	0.3	22.0	5.64	199.650	157.706	357.356
Wall 9 (45%)	1	0.3	32.0	21.86	368.64	2063.514	2432.154
Wall 10 (75%)	1	0.3	22.0	5.64	199.650	157.706	357.356
Wall 11	1	0.3	2.62	1.93	0.450	2.941	3.391
Wall 12 (85%)	1	14.4	0.5	3.49	0.128	74.732	74.860
Wall 13 (90%)	1	21.6	0.5	5.61	0.203	305.425	305.627
Wall 14	1	5.5	0.3	35.11	0.012	2033.460	2033.473
Wall 15	1	14.3	0.4	38.056	0.076	8283.849	8283.926
Wall 16	1	4.3	0.4	36.406	0.023	2279.627	2279.650
Wall 17	1	13.0	0.4	39.16	0.069	7972.420	7972.489
Wall 18	1	0.4	0.7	38.61	0.011	417.309	417.320
Wall 19	1	0.4	2.3	37.76	0.406	1311.444	1311.849
						<b>Total</b>	57 043.499

**Table C.17:** The second moment of area  $I_y$  of segment II.



In the FEM model,  $E_{walls}$  is equal to  $22 \times 10^9$  N/m<sup>2</sup>. This gives the bending stiffnesses  $EI_x = 88 \times 10^{12}$  Nm<sup>2</sup> and  $EI_y = 1255 \times 10^{12}$  Nm<sup>2</sup>. Using the FE segment approach, the bending stiffnesses  $EI_x = 48.81 \times 10^{12}$  Nm<sup>2</sup> and  $EI_y = 170.84 \times 10^{12}$  Nm<sup>2</sup> are obtained.

### C.3.3. Segment III

The calculated second moment of area  $I_x$  is provided in Table C.18. The calculated second moment of area  $I_y$  is provided in Table C.19. The percentage indicates the degree of closeness of the wall. The calculation takes this factor into account. Both are calculated using the floor plan shown in Figure C.10.

Element	N	b [m]	h [m]	d [m]	$\frac{1}{12}bh^3$ [m <sup>4</sup> ]	Steiner [m <sup>4</sup> ]	$I_x$ [m <sup>4</sup> ]
Wall 1 (45%)	1	54.9	0.3	7.60	0.056	427.885	427.941
Wall 2 (50%)	1	54.9	0.3	6.30	0.062	327.034	327.096
Wall 3	1	0.5	15.0	0.25	140.625	0.462	141.087
Wall 4 (75%)	1	12.9	0.3	1.60	0.022	7.414	7.435
Wall 5	1	0.3	8.4	2.75	14.818	19.083	33.900
Wall 6 (85%)	1	0.5	14.4	0.25	105.754	0.377	106.131
Wall 7	1	0.3	5.5	4.202	4.159	29.131	33.290
Wall 8	1	0.4	14.4	0.45	99.533	0.355	99.888
						<b>Total</b>	1176.768

**Table C.18:** The second moment of area  $I_x$  of segment III.

Element	N	b [m]	h [m]	d [m]	$\frac{1}{12}bh^3$ [m <sup>4</sup> ]	Steiner [m <sup>4</sup> ]	$I_y$ [m <sup>4</sup> ]
Wall 1 (45%)	1	0.3	54.9	4.90	1861.528	177.954	2039.482
Wall 2 (50%)	1	0.3	54.9	4.90	2068.364	197.727	2266.091
Wall 3	1	15.0	0.5	22.80	0.156	3898.781	3898.937
Wall 4 (75%)	1	0.3	12.9	16.10	40.250	752.352	792.602
Wall 5	1	8.4	0.3	13.80	0.019	479.905	479.924
Wall 6 (85%)	1	14.4	0.5	9.40	0.128	540.757	540.884
Wall 7	1	5.5	0.3	29.20	0.012	1406.861	1406.874
Wall 8	1	14.4	0.4	32.15	0.077	5953.687	5953.764
						<b>Total</b>	17 378.560

**Table C.19:** The second moment of area  $I_y$  of segment III.

In the FEM model,  $E_{walls}$  is equal to  $33 \times 10^9$  N/m<sup>2</sup>. This gives the bending stiffnesses  $EI_x = 39 \times 10^{12}$  Nm<sup>2</sup> and  $EI_y = 573 \times 10^{12}$  Nm<sup>2</sup>. Using the FE segment approach, the bending stiffnesses  $EI_x = 41.60 \times 10^{12}$  Nm<sup>2</sup> and  $EI_y = 255.56 \times 10^{12}$  Nm<sup>2</sup> are obtained.

### C.3.4. Bending stiffness

The obtained bending stiffnesses are given in Table C.20. Additionally, the displacements at the top, using either a four-segment beam model with these bending stiffnesses or the FE model, are presented, when a force of 10,000 kN is applied and the base of the model is fixed.

With the FE segment approach, the bending stiffnesses vary significantly. The top segment, which represent the smallest part of the building, shows larger bending stiffness values then the bottom segments. These differences are not observed with the bending stiffnesses obtained using the Steiner approach. The displacements obtained with the Steiner approach show similarities with the displacements obtained using the FEM model.

The FE segment approach fails to account for the intermediate relationships between the different segments. For example, segments 0 and I have column structures, which exhibit weaker behavior when considered individually than when integrated into the overall system. This is due the additional mass from other segments. The E-moduli values of the different segments in the FE model may also contribute to the differences, as the top segment has a E-modulus value of  $33 \times 10^9$  N/m<sup>2</sup>, while the bottom segment has a E-modulus value of  $11 \times 10^9$  N/m<sup>2</sup>. However, it is not expected that this caused the factor of 100 difference between the segment stiffnesses.

Due to the differences in the segment stiffnesses obtained using the FE segment approach, the segment stiffnesses obtained using the Steiner approach are applied, with a correction factor included to account for the displacement differences between the FE model and four-segment beam using these segment stiffnesses. Since the FE model is weaker in the x-direction, the segment stiffnesses  $EI_y$  are multiplied by  $\frac{16}{26}$ . Since the FE model is stiffer in the y-direction, the segment stiffnesses  $EI_x$  are multiplied by  $\frac{236}{199}$ .

	$EI_y$ [Nm <sup>2</sup> ]		$EI_x$ [Nm <sup>2</sup> ]		<b>FEM</b>	
	<b>Steiner</b>	<b>FE segment</b>	<b>Steiner</b>	<b>FE segment</b>		
0	614e12	1.04e12	40e12	0.26e12	<b>X</b>	<b>Y</b>
I	614e12	6.46e12	40e12	5.53e12		
II	1255e12	170.84e12	88e12	48.81e12		
III	573e12	255.56e12	39e12	41.60e12		
$\Delta u$ [mm]	16	-	236	-	26	199

**Table C.20:** The bending stiffnesses and displacements determined with the different approaches.

### C.3.5. Structural properties

The dead load and variable load provided by Aronsohn are shown in Table C.21. The obtained structural properties are shown in Table C.22.

<b>Segment</b>	<b>Dead load [kN]</b>	<b>Variable load *0.3 [kN]</b>
0	202190	4641
I	154104	7811
II	241647	16538
III	228309	15722

**Table C.21:** The dead and variable load of the office tower the Delfste Poort.

Segment	0	I	II	III
$L$ [m]	3.15	22.65	46.8	81.27
$A$ [m <sup>2</sup> ]	1320	1320	1320	825
$I_x$ [m <sup>4</sup> ]	4312	4312	4744	1402
$I_y$ [m <sup>4</sup> ]	34350	34350	35105	10685
$\rho$ [kg/m <sup>3</sup> ]	5071	552	426	371
$E$ [N/m <sup>2</sup> ]	11e9	11e9	22e9	33e9
$\nu$ [-]	0.2	0.2	0.2	0.2
$G$ [N/m <sup>2</sup> ]	4.6e9	4.6e9	9.2e9	1.38e9
$k$ [-]	0.85	0.85	0.85	0.85
<b>Boundary</b>				
$K_{r,x}$ [GNm/rad]	6486	$K_{t,x}$ [GN/m]	9.22	
$K_{r,y}$ [GNm/rad]	50371	$K_{t,y}$ [GN/m]	9.22	

**Table C.22:** The structural property values obtained with the floor plans and FE model of the Delftse Poort.

## C.4. Conclusion

With this appendix, the analysis of obtaining the structural property values of the residential tower New Orleans and the office tower the Delftse Poort is provided. The analysis offers insights into how the translation from the FE (Finite Element) model to the discrete Timoshenko beam model is performed and how the different approaches affect the results obtained.

The analysis indicates difficulties obtaining values for the bending stiffness per segment. The Steiner approach, which calculated the bending stiffness per segment using the elastic modulus  $E$  and the second moment of area  $I$  (calculated with  $\frac{1}{12}bh^3$  and the Steiner rule), assumed fully rigid connections between elements when determining  $I$ , making it impossible to accurately account for the contributions of elements that were not rigidly connected.

The FE approach, which used the relative displacement  $\Delta$  of the FE segment to calculate the bending stiffness, did not account for the intermediate relationships between segments, leading to inaccurate estimation of segments with column structures. These segments exhibit weaker behavior when considered individually compared to when they were part of the overall system. This is due to the additional mass of the other segments.

Finally, it should be noted that the FE model also simplifies reality, and differences in behavior should be expected. The discrete Timoshenko beam model is more simplified than the FE model.

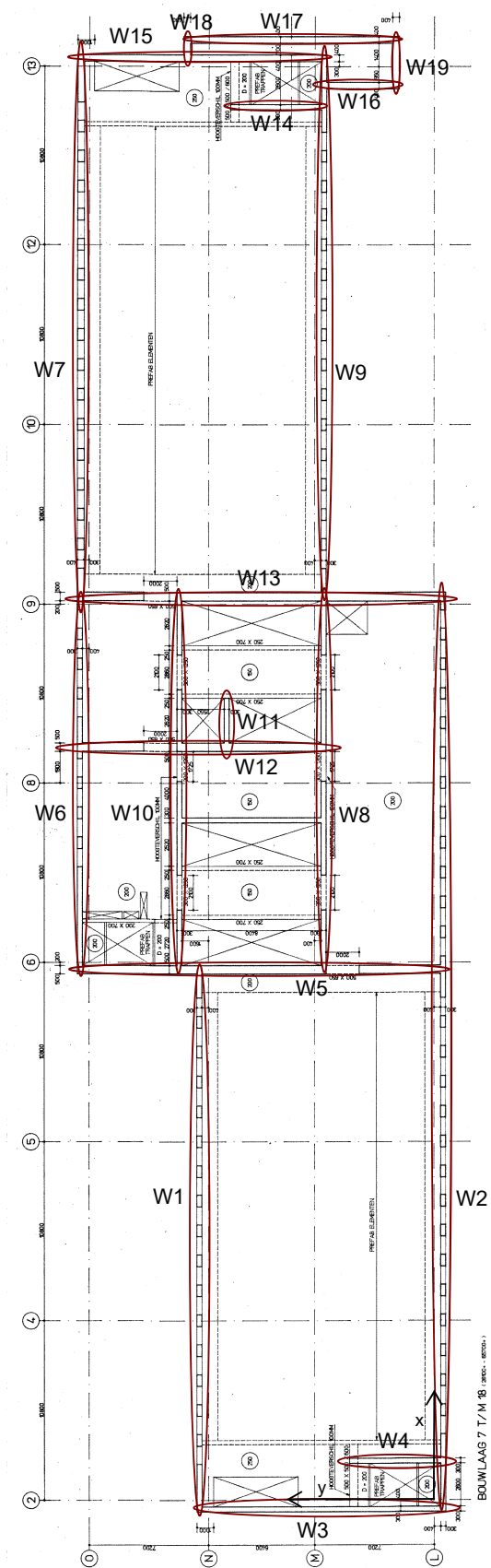


Figure C.9: The floor plan of segment II.

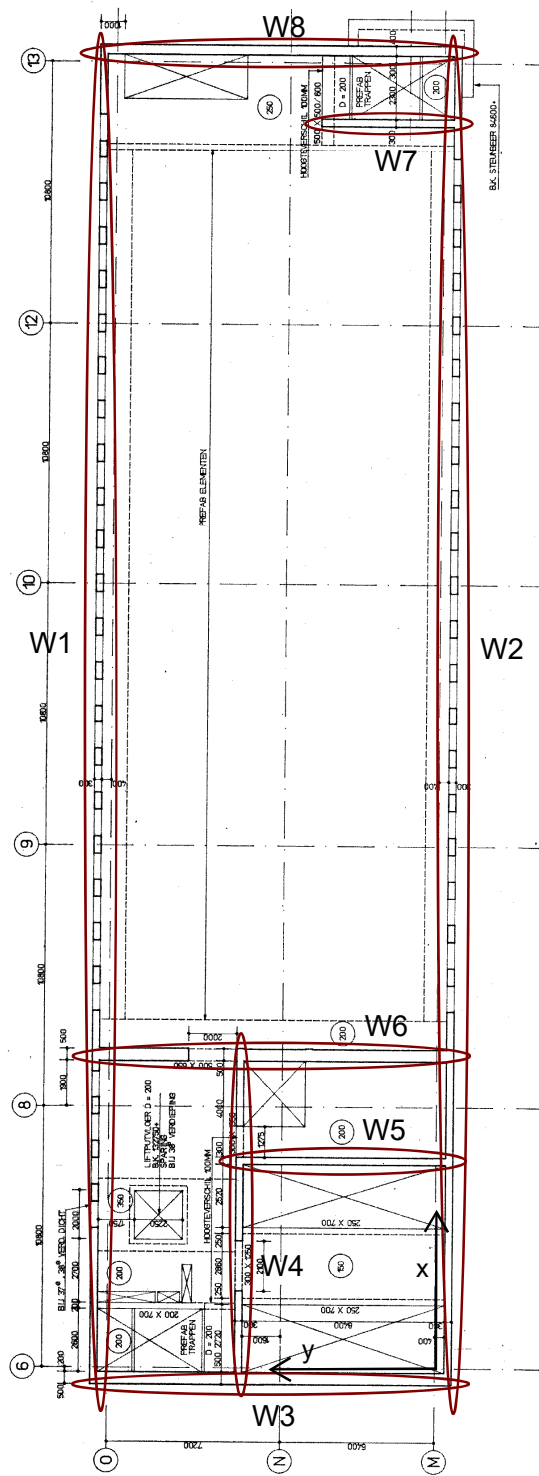


Figure C.10: The floor plan of segment III.

# D

## Measured modal properties

Modal properties refer to the characteristics of a physical system that describe its natural modes of vibration. These properties are important for understanding how a structure responds to dynamic loads and form the basis of the model update method applied in this research. However, these properties need to be determined first from vibration measurements.

Therefore, this appendix explains how the modal properties are determined from vibration measurements. It describes the measurement setup for the New Orleans and the Delftse Poort to obtain the vibration measurements and explains the data analysis used to determine the measured modal properties. The results are presented at the end.

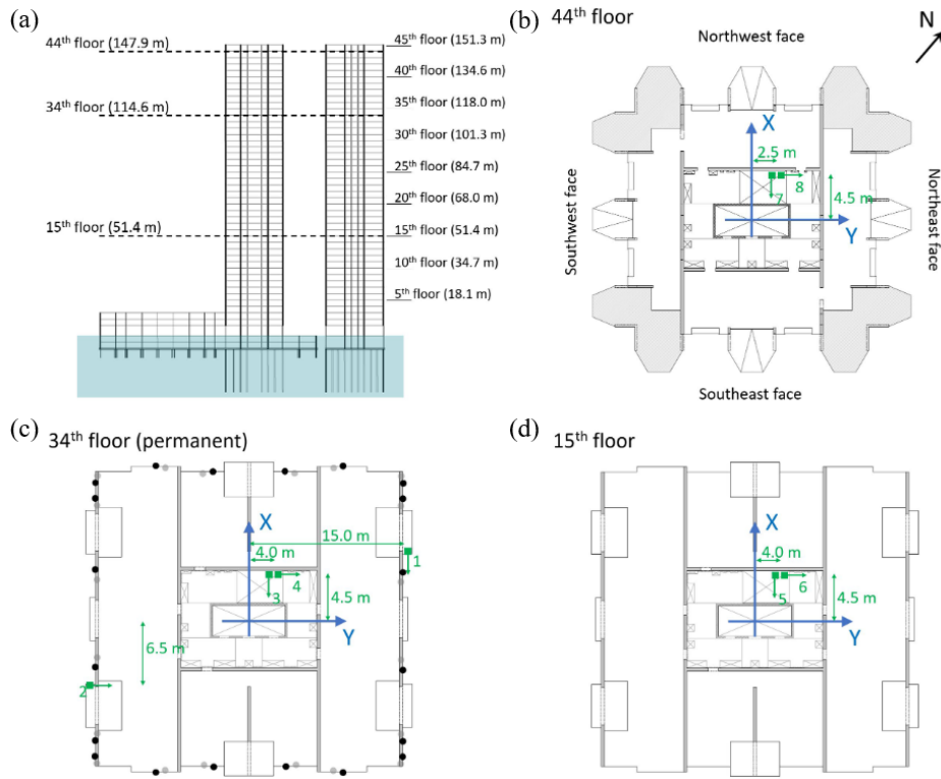
### D.1. Measurement setup the New Orleans

Since 2011, vibration measurements have been conducted on the residential tower New Orleans in Rotterdam, as shown in Figure D.1. The vibrations were monitored on the 34th floor using a permanent monitoring system. The system consisted of four acceleration sensors and was part of a research project focusing on local wind loads on facade elements and the influence of pressure equalization on these loads [17, 12].



**Figure D.1:** An picture of the residential tower New Orleans.

To determine the mode shapes of the building, for several months, additional acceleration sensors were installed on the 15th and 44th floors. The positions and orientations of the sensors are illustrated in Figure D.2 [17, 12].



**Figure D.2:** Pictures of (a) the New Orleans and the positions of the acceleration sensors (green squares) on (b) the 44th floor, (c) the 34th floor, and (d) the 15th floor. The green arrows indicate the directions of the acceleration sensors.

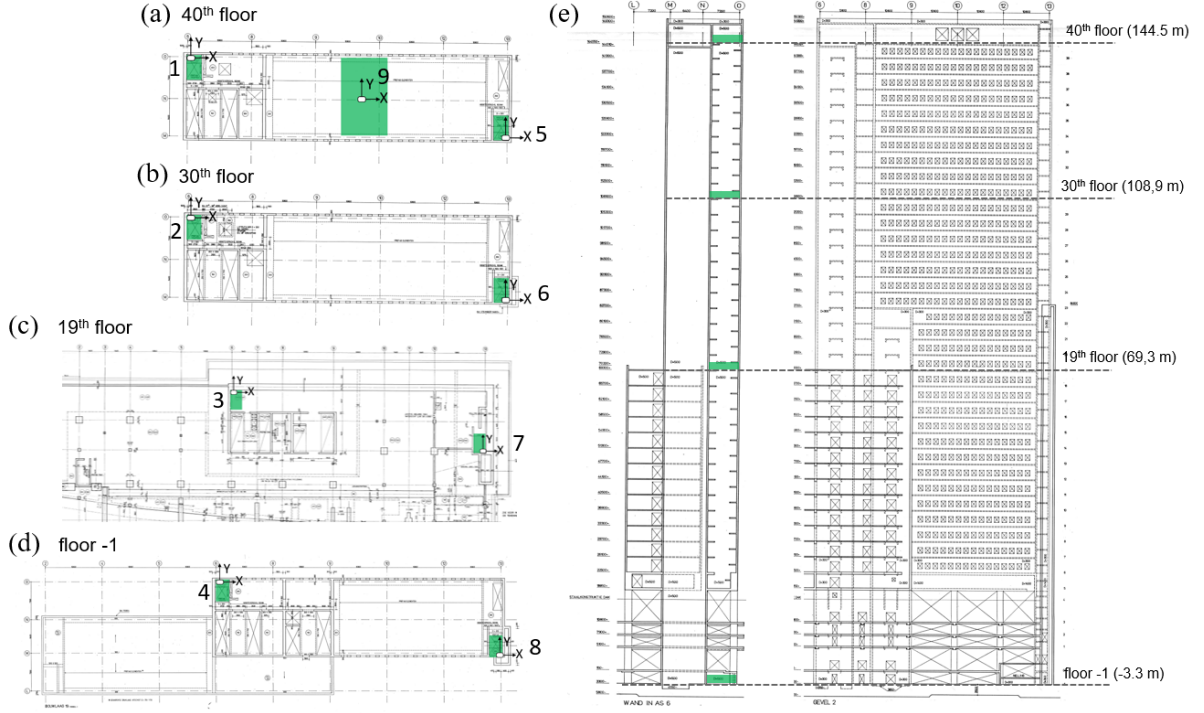
## D.2. Measurement setup the Delfste Poort

Vibration measurements are also conducted on the office tower the Delfste Poort, as shown in Figure D.3.



**Figure D.3:** An picture of the office tower the Delfste Poort.

The system consisted of two or three acceleration sensors on the 19th, 30th and 40th floor and floor 1. The positions and orientations of the sensors are illustrated in Figure D.4.



**Figure D.4:** Pictures of (e) the Delfste Poort and the positions of the acceleration sensors (green squares) on (d) floor 1, on (c) the 19th floor, (b) the 30th floor and (a) the 40th floor. The arrows indicate the directions of the acceleration sensors.

### D.3. Data analysis

For the data analysis, the technique Frequency Domain Decomposition (FDD) is applied. The technique uses the relationship between the inputs  $x(t)$  and the measured responses  $y(t)$ , expressed as [23]:

$$G_{yy}(j\omega) = \bar{H}(j\omega)G_{xx}(j\omega)H(j\omega)^T$$

where the  $G_{xx}(j\omega)$  is the  $(r \times r)$  Power Spectral Density (PSD) matrix of the input,  $r$  is the number of inputs,  $G_{yy}(j\omega)$  is the  $(m \times r)$  PSD of the responses,  $m$  is the number of responses and  $H(j\omega)$  is the frequency response function. The overbar and superscript  $T$  respectively denote the complex conjugate and transpose of the frequency response function [23].

The inputs  $x(t)$  are unknown. Therefore, to estimate the PSD matrix of the measured responses  $y(t)$ , white noise is assumed as input, which has a constant PSD matrix  $G_{xx}(j\omega) = C$  [23]:

$$\hat{G}_{yy}(j\omega) = \bar{H}(j\omega)C(j\omega)H(j\omega)^T$$

The obtained estimate of the PSD  $\hat{G}_{yy}$  is then decomposed at the each known frequency  $\omega = \omega_i$  using Singular Value Decomposition:

$$\hat{G}_{yy}(j\omega_i) = U_i S_i U_i^H$$

Where the matrix  $U_i$  is the unitary matrix containing the singular vectors  $u_{ij}$  and  $S_i$  is the diagonal matrix containing the scalars singular values  $s_{ij}$  at each frequency instance  $\omega$  [23]. The singular vectors  $u_{ij}$  can be associated with the mode shapes of the tested structure. The obtained mode shapes  $\phi_i$  from the matrix  $U$  at the identified natural frequencies are scaled using Degree Of Freedom (DOF) scaling, defined as [12]:

$$\phi_{i,D} = \frac{\phi_i}{\phi_{i,Dn}}$$

where  $\phi_{i,Dn}$  is the DOF with the largest component [24].



To see if the correlation between the obtained mode shapes is low (i.e if the obtained mode shapes are orthogonal), the Modal Assurance Criteria is applied, defined as:

$$MAC(\phi_i, \hat{\phi}_i) = \frac{|\phi_i \cdot \hat{\phi}_i|^2}{(\phi_i \hat{\phi}_i)(\phi_i \cdot \hat{\phi}_i)}$$

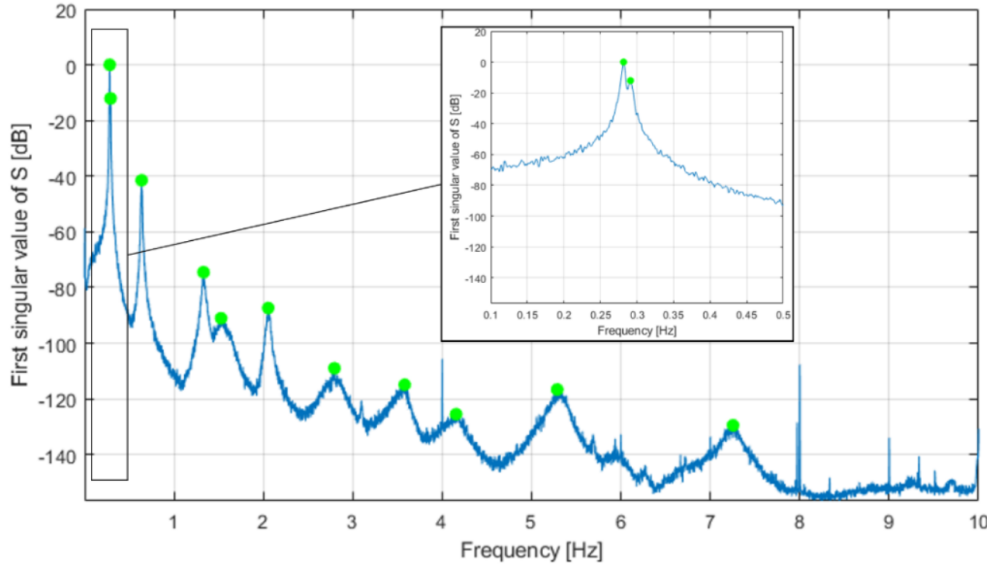
where  $\phi_i$  and  $\hat{\phi}_i$  are the compared mode shapes. If the MAC is high (larger than 35%), it can mean two things [24]:

- The modes are at close frequencies, indicating that the two modes are actually the same mode.
- The modes are at distance frequencies, indicating that the number of sensor positions is not sufficient to differentiate between the two modes.

The mode shapes that have a high MAC value are eliminated. The results of this analysis are given in Section D.4.

## D.4. Results data analysis

The data applied of the New Orleans was recorded on 29/03/2020. A total of 129 10-minute records were used, which were sampled with 20 Hz [24]. The dimensionless spectrum of the first singular value is given in Figure D.5. The obtained modes are listed in Tables D.1 and D.2. The MAC values are given in Table D.3.



**Figure D.5:** The dimensionless spectrum of the first singular value. The green dots indicate the natural frequencies of the residential tower New Orleans [24].

According to Tables D.1 and D.2, for modes 1, 2, 3, 4, 5, 6, 7, and 9, a clear dominant direction is found. For the bending modes, relatively small modal displacements are found in the non-dominant direction. For modes 8, 10, and 11, however, no clear dominant direction can be determined from the modal displacements [24].

Table D.3 shows that modes 1, 2, 4, 5, 7, and 8 have MAC values lower than 35%. This indicates that the applied measurement setup is sufficient to observe these modes. Additionally, the table indicates the orthogonality of these modes [24].

Modes 3, 6, 9, 10, and 11 have more than 35% correlation with other modes. This indicates that the sensor setup is not sufficient to properly distinguish the mode shapes of these modes from the mode shapes of lower modes. However, it should be noted that when mode 3 is not considered, mode 6 does meet the 35% correlation criterion [24].

Mode	1	2	3	4	5	6
Dominant direction	X	Y	$\Theta$	Y	X	$\Theta$
Natural frequency [Hz]	0.282	0.291	0.638	1.332	1.527	2.054
<b>Modal displacement</b>						
Sensor 5 (51.4 m)	-0.30	-0.05	-0.13	-0.00	0.89	0.29
Sensor 6 (51.4 m)	-0.02	-0.29	-0.15	-0.90	0.05	0.50
Sensor 1 (114.6 m)	-0.71	-0.10	-1.00	-0.02	-0.23	-1.00
Sensor 2 (114.6 m)	-0.03	-0.78	-0.43	-0.19	-0.02	-0.45
Sensor 3 (114.6 m)	-0.67	-0.11	-0.28	-0.00	-0.06	-0.24
Sensor 4 (114.6 m)	-0.03	-0.76	-0.30	-0.06	-0.02	-0.21
Sensor 7 (147.9 m)	-1.00	-0.12	-0.23	-0.04	-1.00	-0.25
Sensor 8 (147.9 m)	-0.09	-1.00	-0.38	1.00	-0.11	-0.61

**Table D.1:** The identified measured modes (1-6) of the residential tower New Orleans (normalized).

Mode	7	8	9	10	11
Dominant direction	Y		Y		
Natural frequency [Hz]	2.771	3.560	4.155	5.300	7.250
<b>Modal displacement</b>					
Sensor 5 (51.4 m)	0.02	0.33	0.03	0.08	-0.16
Sensor 6 (51.4 m)	-0.73	0.22	0.18	-0.03	-0.06
Sensor 1 (114.6 m)	-0.11	-1.00	-0.12	-1.00	-1.00
Sensor 2 (114.6 m)	0.83	0.11	1.00	0.15	0.20
Sensor 3 (114.6 m)	-0.06	-0.80	-0.08	-0.58	-0.51
Sensor 4 (114.6 m)	0.99	-0.38	0.95	-0.29	-0.13
Sensor 7 (147.9 m)	0.01	0.41	-0.06	0.16	0.16
Sensor 8 (147.9 m)	-1.00	0.39	0.35	0.14	0.11

**Table D.2:** The identified measured modes (7-11) of the residential tower New Orleans (normalized).

Mode	1	2	3	4	5	6	7	8	9	10	11
1	1.00										
2	0.04	1.00									
3	0.44	0.06	1.00								
4	0.00	0.20	0.01	1.00							
5	0.24	0.01	0.06	0.00	1.00						
6	0.27	0.03	0.69	0.24	0.19	1.00					
7	0.00	0.00	0.09	0.00	0.00	0.04	1.00				
8	0.11	0.00	0.32	0.01	0.00	0.30	0.06	1.00			
9	0.00	0.73	0.01	0.05	0.00	0.02	0.26	0.00	1.00		
10	0.27	0.00	0.63	0.01	0.01	0.43	0.00	0.88	0.00	1.00	
11	0.30	0.00	0.68	0.02	0.00	0.39	0.00	0.74	0.02	0.94	1.00

**Table D.3:** The MAC of the bending modes of Tables D.1 and D.2. The MAC values higher then 0.35 are indicated in lightsteelblue.

The obtained modes for the office tower the Delfste Poort are listed in Tables D.4 and D.5. The MAC values are given in Table D.6. According to Tables D.4 and D.5, for modes 1, 2, 3, 4, 5, 6, 7, 8, 9 and 10, a clear dominant direction is found.

For the bending modes, relatively small modal displacements are found in the non-dominant direction. For mode 11, however, no clear dominant can be determined from the modal displacements.

Table D.6 shows that modes 1, 2, 3, and 4 have MAC values below 35%. The applied measurements setup is sufficient to observe these modes. The orthogonality of these modes is also proven with this table. Modes 5, 6, 7, 8, 9, 10 and 11 have more than 35% correlations with other modes, indicating that the sensor setup is not sufficient to properly distinguish the mode shapes of these modes from the modes shapes of lower modes.

Mode	1	2	3	4	5	6
Dominant direction	Y	X	$\Theta$	Y	X	$\Theta$
Natural frequency [Hz]	0.403	0.861	1.165	1.624	2.027	2.454
<b>Modal displacement</b>						
Sensor 4 (−3.30 m)	0.00	−0.06	−0.05	0.02	0.28	−0.01
Sensor 4 (−3.30 m)	0.01	0.00	−0.03	0.01	0.02	−0.02
Sensor 3 (69.30 m)	−0.02	−0.48	0.00	0.03	0.48	0.01
Sensor 3 (69.30 m)	0.33	0.01	−0.51	0.37	−0.12	0.05
Sensor 7 (69.30 m)	−0.02	−0.49	0.13	0.04	0.44	0.06
Sensor 7 (69.30 m)	0.34	0.00	0.36	0.61	−0.19	0.39
Sensor 2 (108.90 m)	−0.02	−0.77	0.03	−0.05	−0.26	0.01
Sensor 2 (108.90 m)	0.65	0.00	−0.65	−0.19	−0.19	0.37
Sensor 6 (108.90 m)	−0.02	−0.75	0.23	0.03	−0.18	−0.09
Sensor 6 (108.90 m)	0.66	−0.03	0.71	0.42	0.12	−0.16
Sensor 1 (144.51 m)	−0.02	−1.00	0.06	−0.11	−1.00	−0.04
Sensor 1 (144.51 m)	0.96	−0.01	−0.68	−1.00	−0.01	0.55
Sensor 5 (144.51 m)	−0.06	−1.00	0.12	−0.05	−0.99	−0.09
Sensor 5 (144.51 m)	1.00	−0.10	1.00	−0.21	0.41	−1.00
Sensor 9 (144.51 m)	−0.03	−1.00	0.11	−0.08	−0.97	−0.09
Sensor 9 (144.51 m)	0.99	−0.06	0.58	−0.46	0.33	−0.59

**Table D.4:** The identified measured modes (1-5) of the office tower the Delfste Poort (normalized).

Mode	7	8	9	10	11
Dominant direction	X	Y	$\Theta$	X	
Natural frequency [Hz]	2.962	3.749	4.493	4.939	5.844
<b>Modal displacement</b>					
Sensor 4 (−3.30 m)	0.02	−0.03	0.01	0.07	−0.14
Sensor 4 (−3.30 m)	0.01	−0.01	0.03	0.00	−0.21
Sensor 3 (69.30 m)	0.29	0.07	−0.03	0.63	0.13
Sensor 3 (69.30 m)	−0.05	0.18	0.58	−0.15	0.14
Sensor 7 (69.30 m)	0.29	0.06	−0.25	0.71	0.27
Sensor 7 (69.30 m)	0.06	−0.36	−0.70	0.44	0.35
Sensor 2 (108.90 m)	−0.52	0.03	0.05	0.74	0.78
Sensor 2 (108.90 m)	0.04	0.67	0.21	−0.03	−0.02
Sensor 6 (108.90 m)	−0.48	−0.10	−0.05	0.92	1.00
Sensor 6 (108.90 m)	−0.17	0.92	−0.62	0.08	−0.07
Sensor 1 (144.51 m)	−0.85	−0.06	−0.05	−0.95	−0.81
Sensor 1 (144.51 m)	0.27	−0.90	−0.60	0.37	0.14
Sensor 5 (144.51 m)	−1.00	−0.02	−0.03	−1.00	−0.89
Sensor 5 (144.51 m)	−0.19	−0.67	1.00	−0.20	0.19
Sensor 9 (144.51 m)	−0.98	−0.06	−0.01	−0.99	−0.85
Sensor 9 (144.51 m)	−0.08	−1.00	0.60	−0.01	0.28

**Table D.5:** The identified measured modes (7-11) of the office tower the Delfste Poort (normalized).

Mode	1	2	3	4	5	6	7	8	9	10	11
1	1.00										
2	0.00	1.00									
3	0.06	0.03	1.00								
4	0.16	0.01	0.06	1.00							
5	0.03	0.42	0.02	0.00	1.00						
6	0.07	0.02	0.62	0.00	0.03	1.00					
7	0.00	0.68	0.07	0.00	0.72	0.07	1.00				
8	0.16	0.00	0.04	0.35	0.01	0.08	0.00	1.00			
9	0.04	0.00	0.03	0.03	0.02	0.51	0.01	0.06	1.00		
10	0.00	0.04	0.00	0.00	0.42	0.05	0.32	0.00	0.06	1.00	
11	0.03	0.05	0.01	0.00	0.34	0.00	0.18	0.03	0.00	0.82	1.00

**Table D.6:** The MAC of the modes of Tables D.4 and D.5. The MAC values higher then 0.35 are indicated in lightsteelblue.

## D.5. Conclusion

With this appendix, the analysis of obtaining the measured modal properties for the New Orleans and the Delftse Poort is provided, including a verification of the obtained mode shapes. For both buildings, a measurement setup consisting of different acceleration sensors at various heights was used, which successfully identified different modes of the buildings.

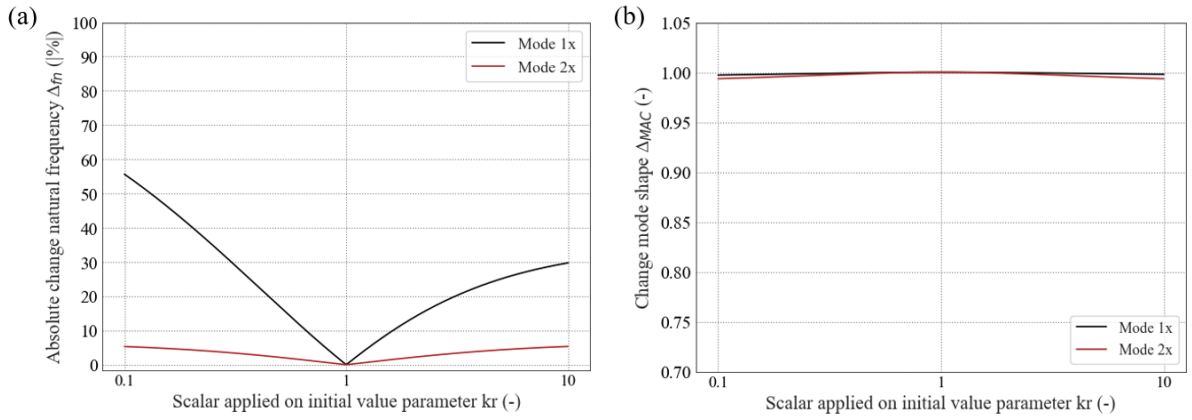
The MAC (Modal Assurance Criterion) was used to determine if the obtained mode shapes are orthogonal. For the identified modes 1, 2, 4, 5, 7, and 8 of the residential tower New Orleans, and modes 1, 2, 3, and 4 of the office tower the Delftse Poort, a MAC value lower then 35% was found, indicating that the sensor setup was sufficient to properly distinguish the mode shapes of these modes.

For the identified modes 3, 6, 9, and 10 of the residential tower New Orleans, and modes 5, 6, 7, 8, 9, 10, and 11 of the office tower the Delftse Poort, a MAC value equal or higher then 35% was found, indicating that the sensor setup was not sufficient to properly distinguish the mode shapes of these modes. This suggests that more sensors are needed at different heights, particularly for the office tower the Delfste Poort.

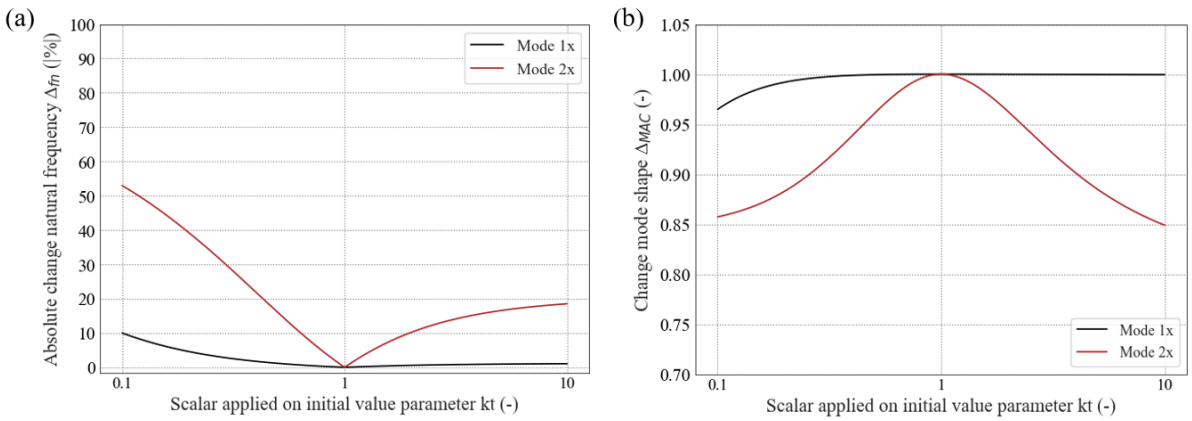
# E

## Influence of parameters

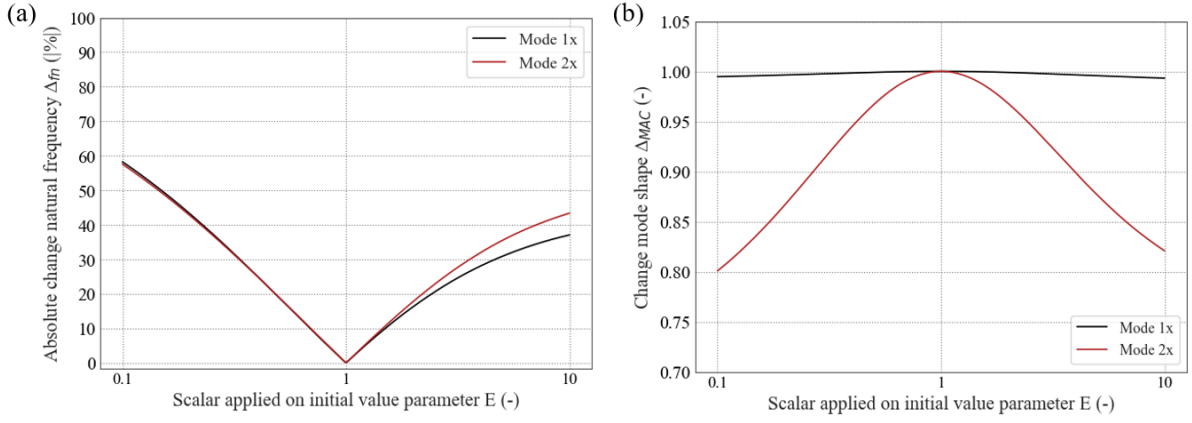
In this chapter, the results of the residential tower New Orleans of the study "Influence Parameters" in x-direction are given.



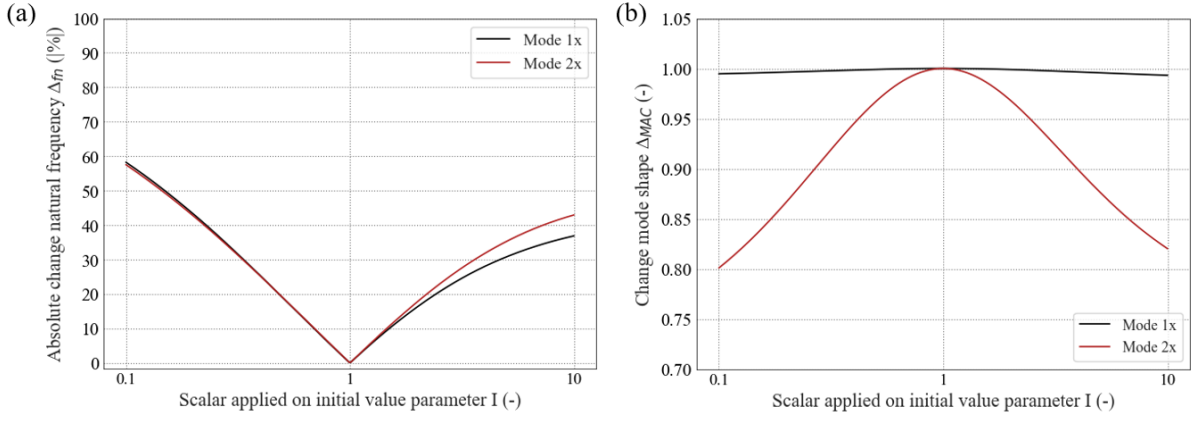
**Figure E.1:** The influence of parameter  $K_r$  on the model output (a,b) in x-direction.



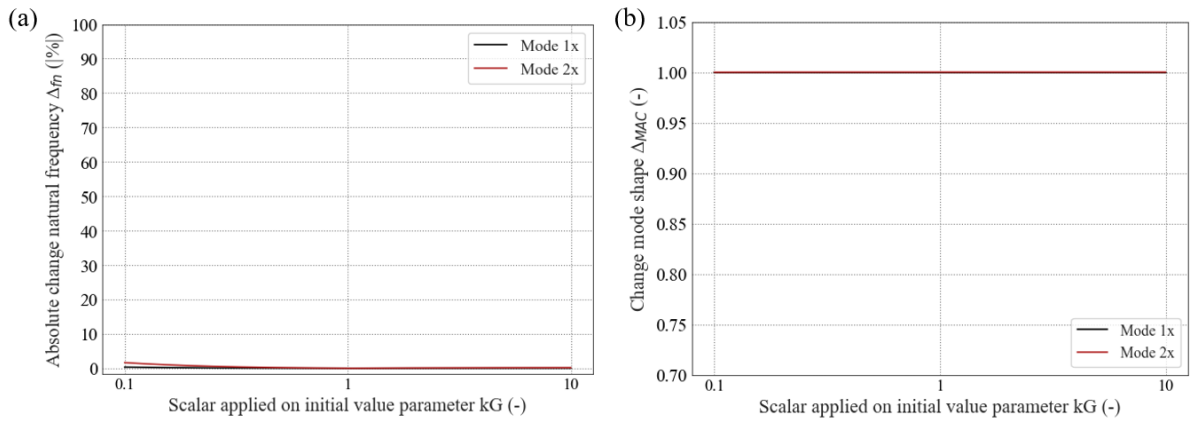
**Figure E.2:** The influence of parameter  $K_t$  on the model output (a,b) in x-direction.



**Figure E.3:** The influence of parameter  $E$  on the model output (a,b) in x-direction.



**Figure E.4:** The influence of parameter  $I$  on the model output (a,b) in x-direction.



**Figure E.5:** The influence of parameter  $kG$  on the model output (a,b) in x-direction.

# F

## Python

In this appendix, the Python code used is presented, including the models, model updating procedures, and various studies conducted. Since the studies involve two case studies using the same code, only the code for one case study is shown.

The discrete Timoshenko beam model (model and model updating)

```
1 from abc import ABC, abstractmethod
2 from copy import deepcopy
3 from dataclasses import dataclass
4 from typing import List, Union, Tuple, Dict, Optional, Any
5
6 import matplotlib.pyplot as plt
7 import pandas as pd
8 import numpy as np
9 import cmath as cm
10 from scipy.optimize import brentq
11 from utils.beam_utils import nullspace
12 from utils.modal_analysis import mac
13 from functools import partial
14 from scipy.stats import uniform
15 from tqdm import tqdm
16 from skopt import gp_minimize, forest_minimize
17 from scipy.optimize import Bounds, minimize
18 from enum import Enum
19
20 class UpdatingMethod(Enum):
21     CMC = "Crude_MonteCarlo"
22     MCMC = "Markov-chain_MonteCarlo"
23     GA = "Genetic_algorithm"
24     DE = "Differential_evolution"
25     PSO = "Partical_swarm_optimsation"
26     GP = "Gaussian_process"
27
28
29 ParamType = Union[dict, list, np.ndarray]
30 import matplotlib
31
32 matplotlib.use("TkAgg")
33 ----- Model Code -----
34 @dataclass
35 class Beam1D(ABC):
36     params: Dict[str, float]
37     height: Union[float, int]
38     bc: Optional[Tuple[str, str]] = ("fixed", "free")
39     bounds: dict = None
40     scales: dict = None
41     description: str = ""
42
43     @abstractmethod
44     def boundaries(
```

```

45         self,
46         omega: float,
47         factor_m: float,
48         factor_v: float,
49         kr: float,
50         p: float,
51         q: float,
52         z: Union[np.ndarray, float],
53         flag: str
54     ):
55         pass
56
57     @abstractmethod
58     def roots(self,
59               omega: float) -> float:
60         pass
61
62 @dataclass
63 class Timo(Beam1D):
64
65     def __init__(self, params, height):
66         self.params = deepcopy(params)
67         self.orig_params = deepcopy(params)
68         self.fn: Union[Dict[str, Any], np.ndarray] = dict()
69         self.phi: Union[Dict[str, Any], np.ndarray] = dict()
70         self.height = height
71
72     def mode_shapes(self):
73         pass
74
75     def roots(self, omega: float):
76         s2 = (self.params["E"] * self.params["I"]) / (self.params["kG"] *
77 self.params["A"] * (self.params["L"] ** 2))
78         b2 = (self.params["rho"] * self.params["A"] * (omega ** 2) *
79 (self.params["L"] ** 4)) / (self.params["E"] * self.params["I"])
80         r2 = (self.params["I"] / self.params["A"]) / self.params["L"] ** 2
81
82         b = b2 * s2 + b2 * r2
83         c = (b2 ** 2) * s2 * r2 - b2
84         d = (b ** 2) - 4 * c
85
86         m1 = cm.sqrt((-b - np.sqrt(d)) / 2)
87         p = abs(m1.imag)
88
89         if omega < np.sqrt((self.params["kG"] * self.params["A"]) /
90 (self.params["rho"] * self.params["I"])):
91             m3 = np.sqrt((-b + np.sqrt(d)) / 2)
92             q = m3
93
94         elif omega == np.sqrt((self.params["kG"] * self.params["A"]) /
95 (self.params["rho"] * self.params["I"])):
96             q = 0
97
98         else:
99             m3 = cm.sqrt((-b + np.sqrt(d)) / 2)
100             q = abs(m3.imag)
101
102         return p, q
103
104     def w(self, p, q, omega: float, z: Union[float, np.ndarray], dz: int = 0):
105         if omega < np.sqrt((self.params["kG"] * self.params["A"]) /
106 (self.params["rho"] * self.params["I"])):
107             if dz == 0:
108                 return np.array([np.sin(p * z), np.cos(p * z),
109 np.sinh(q * z), np.cosh(q * z)])
110             elif dz == 1:
111                 return np.array([p * np.cos(p * z), -p * np.sin(p * z),
112 q * np.cosh(q * z), q * np.sinh(q * z)])
113
114         elif omega == np.sqrt((self.params["kG"] * self.params["A"]) /
115 (self.params["rho"] * self.params["I"])):

```



```

116         if dz == 0:
117             return np.array([np.sin(p * z), np.cos(p * z), 0, 1])
118         elif dz == 1:
119             return np.array([p * np.cos(p * z), -p * np.sin(p * z), 0, 0])
120
121     else:
122         if dz == 0:
123             return np.array([np.sin(p * z), np.cos(p * z),
124                             np.sin(q * z), np.cos(q * z)])
125         elif dz == 1:
126             return np.array([p * np.cos(p * z), -p * np.sin(p * z),
127                             q * np.cos(q * z), -q * np.sin(q * z)])
128
129     def phi(self, p, q, omega: float, z: Union[float, np.ndarray], dz: int = 0):
130         s2 = (self.params["E"] * self.params["I"]) /
131         (self.params["kG"] * self.params["A"] * (self.params["L"] ** 2))
132         b2 = (self.params["rho"] * self.params["A"] * (omega ** 2) *
133              self.params["L"] ** 4) / (self.params["E"] * self.params["I"])
134
135         if omega < np.sqrt((self.params["kG"] * self.params["A"]) /
136                           (self.params["rho"] * self.params["I"])):
137             l1 = b2 * s2 / p - p
138             l2 = -b2 * s2 / p + p
139             l3 = b2 * s2 / q + q
140             l4 = l3
141
142         if dz == 0:
143             return np.array([l2 * np.cos(p * z), l1 * np.sin(p * z),
144                             l4 * np.cosh(q * z), l3 * np.sinh(q * z)])
145         elif dz == 1:
146             return np.array([-l2 * p * np.sin(p * z), l1 * p * np.cos(p * z),
147                             l4 * q * np.sinh(q * z), l3 * q * np.cosh(q * z)])
148
149         elif omega == np.sqrt((self.params["kG"] * self.params["A"]) /
150                               (self.params["rho"] * self.params["I"])):
151             l1 = b2 * s2 / p - p
152             l2 = -b2 * s2 / p + p
153
154         if dz == 0:
155             return np.array([l2 * np.cos(p * z), l1 * np.sin(p * z),
156                             1, b2 * s2 * z])
157         elif dz == 1:
158             return np.array([-l2 * p * np.sin(p * z), l1 * p * np.cos(p * z),
159                             0, b2 * s2])
160
161     else:
162         l1 = (b2 * s2 / p) - p
163         l2 = -(b2 * s2 / p) + p
164         l3 = (b2 * s2 / q) - q
165         l4 = -(b2 * s2 / q) + q
166
167         if dz == 0:
168             return np.array([l2 * np.cos(p * z), l1 * np.sin(p * z),
169                             l4 * np.cos(q * z), l3 * np.sin(q * z)])
170         elif dz == 1:
171             return np.array([-l2 * p * np.sin(p * z), l1 * p * np.cos(p * z),
172                             -l4 * q * np.sin(q * z), l3 * q * np.cos(q * z)])
173
174     def factor_continuity(self):
175         factor_m = 1 / (self.params["E"] * self.params["I"])
176         factor_v = 1 / (self.params["kG"] * self.params["A"])
177         return factor_m, factor_v
178
179     # Remove this if frequency dependency is not taken into account of this parameter! This
180     # is for the article.
181     def kr(self, omega):
182         Vs = np.sqrt(self.params["Gs"] / self.params["rhos"])
183         a0 = (omega * self.params["D"]) / (2 * Vs)
184
185         b = 2.4 - 0.4 / (self.params["B"] / self.params["D"])**3
186         d = 0.55 + 0.01 * np.sqrt((self.params["B"] / self.params["D"]) - 1)

```

```

186         krd = 1 - (d * a0**2)/(b + a0**2)
187         kr = self.params["kr"] * krd
188         return kr
189
190     # Remove kr: float and change kr to self.params["kr"] if the frequency dependency of this
191     # parameter is not taken into account
192     def boundaries(
193         self,
194         omega: float,
195         factor_m: float,
196         factor_v: float,
197         kr: float,
198         p: float,
199         q: float,
200         z: Union[np.ndarray, float],
201         flag: str
202     ):
203         if flag.lower() == "continuity+" or flag.lower() == "continuity-":
204             mm = np.zeros([4, 4])
205         else:
206             mm = np.zeros([2, 4])
207
208         if flag.lower() == "fixed":
209             mm[0, :] = Timo.w(self, omega=omega, p=p, q=q, z=z)
210             mm[1, :] = Timo.phi(self, omega=omega, p=p, q=q, z=z)
211
212         elif flag.lower() == "hinged":
213             mm[0, :] = Timo.phi(self, omega=omega, p=p, q=q, z=z, dz=1)
214             mm[1, :] = Timo.w(self, omega=omega, p=p, q=q, z=z)
215
216         elif flag.lower() == "rot+transl":
217             mm[0, :] = Timo.phi(self, omega=omega, p=p, q=q, z=z, dz=1) -
218             (kr / (self.params["E"] * self.params["I"]
219             / self.params["L"])) *
220             Timo.phi(self, omega=omega, p=p, q=q, z=z)
221             mm[1, :] = Timo.phi(self, omega=omega, p=p, q=q, z=z) -
222             Timo.w(self, omega=omega, p=p, q=q, z=z, dz=1) +
223             (self.params["kt"] / ((self.params["A"] * self.params["kG"]
224             / self.params["L"])) * Timo.w(self, omega=omega, p=p, q=q, z=z)
225
226         elif flag.lower() == "rot":
227             mm[0, :] = Timo.phi(self, omega=omega, p=p, q=q, z=z, dz=1) -
228             (kr / (self.params["E"] *
229             self.params["I"] / self.params["L"])) *
230             Timo.phi(self, omega=omega, p=p, q=q, z=z)
231             mm[1, :] = Timo.w(self, omega=omega, p=p, q=q, z=z)
232
233         elif flag.lower() == "transl":
234             mm[0, :] = Timo.phi(self, omega=omega, p=p, q=q, z=z)
235             mm[1, :] = Timo.phi(self, omega=omega, p=p, q=q, z=z) -
236             Timo.w(self, omega=omega, p=p, q=q, z=z, dz=1) +
237             (self.params["kt"] / ((self.params["A"] * self.params["kG"]
238             / self.params["L"])) * Timo.w(self, omega=omega, p=p, q=q, z=z)
239
240         elif flag.lower() == "free":
241             mm[0, :] = Timo.phi(self, omega=omega, p=p, q=q, z=z, dz=1)
242             mm[1, :] = Timo.phi(self, omega=omega, p=p, q=q, z=z) -
243             Timo.w(self, omega=omega, p=p, q=q, z=z, dz=1)
244
245         elif flag.lower() == "continuity+":
246             fm = factor_m * self.params["E"] * self.params["I"]
247             fv = factor_v * self.params["kG"] * self.params["A"]
248             mm[0, :] = Timo.w(self, omega=omega, p=p, q=q, z=z)
249             mm[1, :] = Timo.w(self, omega=omega, p=p, q=q, z=z, dz=1)
250             mm[2, :] = Timo.phi(self, omega=omega, p=p, q=q, z=z, dz=1) * fm
251             mm[3, :] = (Timo.phi(self, omega=omega, p=p, q=q, z=z) -
252             Timo.w(self, omega=omega, p=p, q=q, z=z, dz=1)) * fv
253
254         elif flag.lower() == "continuity-":
255             mm[0, :] = -Timo.w(self, omega=omega, p=p, q=q, z=z)
256             mm[1, :] = -Timo.w(self, omega=omega, p=p, q=q, z=z, dz=1)

```

```

256         mm[2, :] = -Timo.phi(self, omega=omega, p=p, q=q, z=z, dz=1)
257         mm[3, :] = -(Timo.phi(self, omega=omega, p=p, q=q, z=z) -
258             Timo.w(self, omega=omega, p=p, q=q, z=z, dz=1))
259         return mm
260
261
262 @dataclass
263 class TimoPieceWiseBeam1D:
264     beams: List[Beam1D]
265     bc: tuple
266     beam = Timo
267
268     def __init__(self, beams, bc):
269         self.beams = beams
270         self.params = {k: beam.params[k] for beam in beams for k in beam.params}
271         self.bc = bc
272         self.orig_params = deepcopy(self.params)
273
274     def update_dict(self, attr: str, new_dict: dict):
275         for ix, beam in enumerate(self.beams):
276             getattr(self, attr).update(new_dict)
277
278     def scale_parms(self, scalars: dict):
279         for ix, beam in enumerate(self.beams):
280             for k, val in scalars.items():
281                 beam.params[k] = beam.orig_params[k] * val
282
283     def update_params(self, updated_params: dict):
284         for ix, beam in enumerate(self.beams):
285             for k, val in updated_params.items():
286                 beam.params[k] = val
287
288     def update_originals(self, updated_params):
289         for k, val in updated_params.items():
290             self.orig_params[k] = val
291
292     def set_dict(self, attr: str, new_dict: dict):
293         setattr(self, attr, new_dict)
294
295     def modeshapes(
296         self,
297         min_fn: float = 0.002,
298         max_fn: float = 100,
299         pts: Union[np.ndarray, int] = 500,
300         n_intervals: int = 100,
301         n_modes: int = 3):
302
303         n_b = len(self.beams)
304         heights = np.cumsum([0] + [beam.height for beam in self.beams])
305         min_omega, max_omega = min_fn * 2 * np.pi, max_fn * 2 * np.pi
306         omega_vec = np.logspace(np.log10(min_omega), np.log10(max_omega), n_intervals)
307         omega_critical = np.sqrt((self.params["kG"] * self.params["A"]) /
308             (self.params["rho"] * self.params["I"]))
309         print("The critical frequency of the Timoshenko beam model is:")
310         print(omega_critical / (2 * np.pi))
311         omega_part_1 = omega_vec[omega_vec < omega_critical]
312         omega_part_3 = omega_vec[omega_vec > omega_critical]
313
314         omega_part_1 = np.append(omega_part_1, omega_critical - 0.00000001)
315         omega_part_3 = np.insert(omega_part_3, 0, omega_critical + 0.00000001)
316
317         fun_v_1 = np.array([self.determinant(omega=om_ii, z=heights) for om_ii in
318             omega_part_1])
319         fun_v_3 = np.array([self.determinant(omega=om_iii, z=heights) for om_iii in
320             omega_part_3])
321
322         idx_change_1 = np.where(np.diff(np.sign(fun_v_1)) != 0)[0]
323         nsol = len(idx_change_1)
324
325         if nsol < n_modes:
326             fun_v_2 = self.determinant(omega=omega_critical, z=heights)

```

```

325         if fun_v_2 == 0:
326             nsol += 1
327
328         idx_change_3 = np.where(np.diff(np.sign(fun_v_3)) != 0)[0]
329         nsol += len(idx_change_3)
330
331         omega_sol = np.empty(0)
332
333         for ii in idx_change_1:
334             if len(omega_sol) == n_modes:
335                 break
336             sol = brentq(self.determinant, omega_part_1[ii], omega_part_1[ii + 1], args=(
337                 heights,))
338             omega_sol = np.append(omega_sol, sol)
339
340         if len(omega_sol) != n_modes and fun_v_2 == 0:
341             omega_sol = np.append(omega_sol, omega_critical)
342
343         for iii in idx_change_3:
344             if len(omega_sol) == n_modes:
345                 break
346             sol = brentq(self.determinant, omega_part_3[iii], omega_part_3[iii + 1], args=(
347                 heights,))
348             omega_sol = np.append(omega_sol, sol)
349
350         if len(omega_sol) < n_modes:
351             print("Solution are less than required number of modes. Try a larger interval or
352                 larger bounds.")
353
354         if len(omega_sol) == 0:
355             print("Failed in obtaining solutions!!")
356
357         omega = omega_sol
358         fn = omega / (2 * np.pi)
359
360         if isinstance(pts, int):
361             pts_pieces = np.linspace(heights[-1], heights[1:], pts, endpoint=False).T
362             pts = np.insert(pts_pieces.ravel(), len(pts_pieces.ravel()), heights[-1])
363
364         phi = np.zeros((len(pts), len(omega)))
365         pts = pts.flatten()
366
367         coeff = np.zeros((n_b * 4, 1))
368
369         for aa, om_aa in enumerate(omega):
370             mm = self.determinant(omega=om_aa, z=heights, build=True)
371             rtol = 0.1
372             not_converge = True
373
374             while not_converge:
375                 coeff = nullspace(mm, atol=0, rtol=rtol).reshape(-1, 1)
376
377                 if coeff.shape == (0, 1) or coeff.shape[0] > 4 * n_b:
378                     rtol /= 10
379                 else:
380                     not_converge = False
381
382             for ix, beam in enumerate(self.beams):
383                 idx_z = (pts >= heights[ix]) & (pts <= heights[ix + 1])
384
385                 p_, q_ = beam.roots(omega=om_aa)
386
387                 z_ix = pts[idx_z]
388
389                 if om_aa != omega_critical:
390                     phi[idx_z, aa] = np.sum(
391                         np.real(beam.w(p=p_, q=q_, omega=om_aa, z=z_ix) * coeff[4 * ix:(4 *
392

```

```

392         'Warning, there is a double eigenvalue, please check with plots if
393         the procedure goes as intended!')
394
395     phi /= phi[np.argmax(np.abs(phi), axis=0), np.arange(phi.shape[1])]
396     self.fn = fn
397     self.phi = phi
398     return fn, phi, pts
399
400     def determinant(
401         self,
402         omega: float,
403         z: np.ndarray,
404         build: bool = False,
405     ):
406         n_b = len(self.beams)
407         mm = np.zeros((4 * n_b, 4 * n_b))
408         bc_s, bc_e = self.bc
409         f__v = []
410         f__m = []
411         for ix, beam in enumerate(self.beams):
412             fv = beam.factor_continuity()[1]
413             fm = beam.factor_continuity()[0]
414             f__v.append(fv)
415             f__m.append(fm)
416             if ix == 0:
417                 f_m = f__m[0]
418                 f_v = f__v[0]
419             else:
420                 f_m = f__m[ix - 1]
421                 f_v = f__v[ix - 1]
422
423             bc_s_ix = (bc_s if ix == 0 else "continuity+")
424             bc_e_ix = (bc_e if ix == (n_b - 1) else "continuity-")
425
426             _p = beam.roots(omega=omega)[0]
427             _q = beam.roots(omega=omega)[1]
428
429             _kr = beam.kr(omega=omega)
430
431             _mm_s = beam.boundaries(omega=omega, factor_m=f_m, factor_v=f_v, kr=_kr, p=_p, q=
432                 _q, z=z[ix], flag=bc_s_ix)
433             _mm_e = beam.boundaries(omega=omega, factor_m=f_m, factor_v=f_v, kr=_kr, p=_p, q=
434                 _q, z=z[ix + 1], flag=bc_e_ix)
435
436             c_ix = np.arange(4 * ix, (4 * ix + 4))
437             r_st = (ix if ix == 0 else 4 * ix - 2)
438
439             _mm = np.vstack([_mm_s, _mm_e])
440             r_ix = np.arange(r_st, r_st + _mm.shape[0])
441             mm[np.ix_(r_ix, c_ix)] = _mm
442
443         if build:
444             return mm
445         else:
446             return np.linalg.det(mm)
447
448     @dataclass
449     class TimoPieceWiseBeam2D:
450         beams_x: List[Beam1D]
451         beams_y: List[Beam1D]
452         bc: tuple
453
454         def __init__(self, beams_x, beams_y, bc):
455             self.beams_x = beams_x
456             self.beams_y = beams_y
457             self.params_x = {k: beam.params[k] for beam in beams_x for k in beam.params}
458             self.params_y = {k: beam.params[k] for beam in beams_y for k in beam.params}
459             self.bc = bc
460             self.orig_params_x = deepcopy(self.params_x)
461             self.orig_params_y = deepcopy(self.params_y)

```

```

460
461 def update_dict(self, attr: str, new_dict_x: dict, new_dict_y: dict):
462     for beam in self.beams_x:
463         getattr(beam, attr).update(new_dict_x)
464     for beam in self.beams_y:
465         getattr(beam, attr).update(new_dict_y)
466
467 def scale_params(self, scalars: dict):
468     for k, val in scalars.items():
469         for beam in self.beams_x:
470             beam.params[k] = beam.orig_params[k] * val
471         for beam in self.beams_y:
472             beam.params[k] = beam.orig_params[k] * val
473
474 def update_params(self, updated_params_x: dict, updated_params_y: dict):
475     for k, val in updated_params_x.items():
476         for beam in self.beams_x:
477             beam.params[k] = val
478     for k, val in updated_params_y.items():
479         for beam in self.beams_y:
480             beam.params[k] = val
481
482 def update_originals(self, updated_params_x, updated_params_y):
483     for k, val in updated_params_x.items():
484         self.orig_params_x[k] = val
485     for k, val in updated_params_y.items():
486         self.orig_params_y[k] = val
487
488 def set_dict(self, attr: str, new_dict_x: dict, new_dict_y: dict):
489     for beam in self.beams_x:
490         setattr(beam, attr, new_dict_x)
491     for beam in self.beams_y:
492         setattr(beam, attr, new_dict_y)
493
494 def modeshapes(
495     self,
496     min_fn: float = 0.002,
497     max_fn: float = 40000,
498     pts: Union[np.ndarray, int] = 500,
499     n_intervals: int = 10000,
500     n_modes_x: int = 3,
501     n_modes_y: int = 3,
502 ):
503     fn_x, phi_x, pts_x = TimoPieceWiseBeam1D(self.beams_x, self.bc).modeshapes(min_fn,
504                                         max_fn, pts, n_intervals, n_modes_x)
505     fn_y, phi_y, pts_y = TimoPieceWiseBeam1D(self.beams_y, self.bc).modeshapes(min_fn,
506                                         max_fn, pts, n_intervals, n_modes_y)
507
508     self.fn_x = fn_x
509     self.phi_x = phi_x
510     self.fn_y = fn_y
511     self.phi_y = phi_y
512
513     return fn_x, fn_y, phi_x, phi_y, pts_x, pts_y
514
515 ----- Model Updating Code -----
516
517 def sample_start_points(bounds: ParamType, npts: int, seed: int = 42):
518     np.random.seed(seed=seed)
519     st_pts = np.zeros((npts, len(bounds)))
520     for ii, _bb in enumerate(bounds):
521         st_pts[:, ii] = uniform.rvs(_bb[0], _bb[1], size=npts)
522     return st_pts
523
524 def assemble_boundaries(
525     parameters_to_update: ParamType,
526     a: float = 0.1,
527     b: float = 10,
528     logscale: bool = False,
529 ):

```

```

529
530     if isinstance(parameters_to_update, (list, np.ndarray)):
531         bounds = [(a, b)] * len(parameters_to_update)
532
533     elif isinstance(parameters_to_update, dict):
534         bounds = []
535         for pp, bb in parameters_to_update.items():
536             if bb is None:
537                 bounds.append((a, b))
538             else:
539                 bounds.append((bb[0], bb[1]))
540     else:
541         raise TypeError
542
543     if logscale:
544         bounds = [(np.log(a), np.log(b)) for (a, b) in bounds]
545     return bounds
546
547
548 def adjust_modes(phi_hat, phi):
549     rmse_plus = np.sqrt(np.mean((phi_hat - phi) ** 2, axis=0))
550     rmse_minus = np.sqrt(np.mean((phi_hat + phi) ** 2, axis=0))
551
552     flip = np.ones(len(rmse_plus))
553     flip[rmse_plus > rmse_minus] = -1
554     phi_flip = phi_hat * flip
555     return phi_flip
556
557 def cost_func1D(
558     theta,
559     model,
560     y_hat,
561     parameters_to_update,
562     logscale,
563     regularisation: float = None,
564     symmetry: float=None
565 ):
566     fn = y_hat["fn"]
567     n_modes = len(fn)
568
569     if logscale:
570         theta = np.array([np.exp(aa) for aa in theta])
571
572     if isinstance(parameters_to_update, (np.ndarray, list)):
573         parms_names = parameters_to_update
574     elif isinstance(parameters_to_update, dict):
575         parms_names = list(parameters_to_update.keys())
576     else:
577         raise TypeError
578     new_parms = {k: th for k, th in zip(parms_names, theta)}
579     model.scale_parms(scalers=new_parms)
580     model.modeshapes(n_modes=n_modes, pts=y_hat["z"])
581     cost_fn = np.sum(np.abs(fn - model.fn) / model.fn)
582
583     if fn.shape != model.fn.shape:
584         print("Error in shape of modes(x)")
585
586     if regularisation is not None:
587         current_params = np.array([model.params[vv] for vv in parms_names])
588         orig_params = np.array([model.orig_params[vv] for vv in parms_names])
589         delta_props = ((current_params - orig_params) / orig_params) ** 2
590         cost_fn += regularisation * delta_props.sum()
591
592     if "modes" in list(y_hat.keys()):
593         phi = np.atleast_2d(y_hat["modes"])
594         phi_hat = adjust_modes(model.phi, phi)
595         mac_ph_p = np.diag(mac(phi_hat, phi))
596         cost_modes = np.sum((1 - mac_ph_p))
597         return cost_fn + cost_modes
598     else:
599         return cost_fn

```

```

600
601 def cost_func2D(
602     theta,
603     model: TimoPieceWiseBeam2D,
604     y_hat,
605     parameters_to_update,
606     logscale,
607     regularisation: float = None,
608     symmetry: float = None,
609 ):
610     fn_x = y_hat["fn_x"]
611     fn_y = y_hat["fn_y"]
612     n_modes_x = len(fn_x)
613     n_modes_y = len(fn_y)
614
615     if logscale:
616         theta = np.array([np.exp(aa) for aa in theta])
617
618     if isinstance(parameters_to_update, (np.ndarray, list)):
619         parms_names = parameters_to_update
620     elif isinstance(parameters_to_update, dict):
621         parms_names = list(parameters_to_update.keys())
622     else:
623         raise TypeError
624     new_parms = {k: th for k, th in zip(parms_names, theta)}
625     model.scale_params(scalers=new_parms)
626     model.modeshapes(n_modes_x= n_modes_x, n_modes_y=n_modes_y, pts=y_hat["z"])
627
628     cost_fn = np.sum(np.abs(fn_x - model.fn_x) / model.fn_x)
629     if fn_x.shape != model.fn_x.shape:
630         print("Error in shape of modes_x")
631     cost_fn += np.sum(np.abs(fn_y - model.fn_y) / model.fn_y)
632     if fn_y.shape != model.fn_y.shape:
633         print("Error in shape of modes_y")
634
635     if regularisation is not None:
636         current_params = np.array([model.params[vv] for vv in parms_names])
637         orig_params = np.array([model.orig_params[vv] for vv in parms_names])
638         delta_props = ((current_params - orig_params) / orig_params) ** 2
639         cost_fn += regularisation * delta_props.sum()
640
641     if symmetry is not None:
642         npr = len(theta) // 2
643         ratios = np.array([theta[ii]/theta[npr+ii] for ii in range(npr)])
644         cost_fn += symmetry * np.abs(np.log(ratios)).sum()
645
646     if "modes_x" in list(y_hat.keys()):
647         phi_x = np.atleast_2d(y_hat["modes_x"])
648         phi_hat_x = adjust_modes(model.phi_x, phi_x)
649         mac_ph_p_x = np.diag(mac(phi_hat_x, phi_x))
650         cost_modes_x = np.sum((1 - mac_ph_p_x))
651         cost_fn += cost_modes_x
652
653     if "modes_y" in list(y_hat.keys()):
654         phi_y = np.atleast_2d(y_hat["modes_y"])
655         phi_hat_y = adjust_modes(model.phi_y, phi_y)
656         mac_ph_p_y = np.diag(mac(phi_hat_y, phi_y))
657         cost_modes_y = np.sum((1 - mac_ph_p_y))
658         cost_fn += cost_modes_y
659
660     return cost_fn
661
662 def update_model(
663     model: TimoPieceWiseBeam1D,
664     parameters_to_update: ParamType,
665     cost_func,
666     y_hat: dict,
667     npts: int = 100,
668     sim_type: UpdatingMethod = UpdatingMethod.CMC,
669     optim_method: str = "SLSQP",
670     seed: int = 42,

```



```

671     verbose: bool = False,
672     logscale: bool = False,
673     regularisation: float = None,
674     symmetry: float = None,
675     kwargs_optimizer: dict = None,
676 ):
677     if kwargs_optimizer is None:
678         kwargs_optimizer = {}
679
680     bounds = assemble_boundaries(
681         parameters_to_update=parameters_to_update, logscale=logscale
682     )
683
684     xl = [bb[0] for bb in bounds]
685     xu = [bb[1] for bb in bounds]
686
687     if isinstance(parameters_to_update, (np.ndarray, list)):
688         parms_names = parameters_to_update
689     elif isinstance(parameters_to_update, dict):
690         parms_names = list(parameters_to_update.keys())
691
692     params_opt = np.array([prm + "_opt" for prm in parms_names])
693     params_st = np.array([prm + "_st" for prm in parms_names])
694
695     args_cf = (model, y_hat, parameters_to_update, logscale, regularisation)
696
697     solutions = []
698     convergence = []
699
700     objective = partial(
701         cost_func,
702         model=model,
703         y_hat=y_hat,
704         logscale=logscale,
705         parameters_to_update=parameters_to_update,
706         regularisation=regularisation,
707         symmetry=symmetry
708     )
709
710     if sim_type == UpdatingMethod.CMC:
711
712         st_pts = sample_start_points(bounds=bounds, npts=npts, seed=seed)
713         iterations_per_start_point = []
714         number_function_evaluations = []
715
716         for ii in tqdm(range(npts)):
717             min_kwargs = dict(
718                 x0=st_pts[ii, :], method=optim_method, bounds=bounds
719             )
720
721             sol = minimize(objective, **min_kwargs)
722             solutions.append(sol.x)
723             convergence.append(sol.fun)
724             iterations_per_start_point.append(sol.nit)
725             number_function_evaluations.append(sol.nfev)
726
727             #print(iterations_per_start_point)
728             #print(number_function_evaluations)
729
730     elif sim_type == UpdatingMethod.GP:
731
732         sol = gp_minimize(
733             objective,
734             bounds,
735             initial_point_generator="lhs",
736             n_initial_points=npts,
737             n_calls=30,
738             verbose=verbose,
739         )
740         solutions.append(sol.x)
741         convergence.append(sol.fun)

```

```

742
743 elif sim_type in [UpdatingMethod.DE, UpdatingMethod.GA, UpdatingMethod.PSO]:
744     from pymoo.core.problem import ElementwiseProblem
745     import pymoo.optimize as opt
746     import importlib
747
748     algorithm_class = getattr(
749         importlib.import_module(
750             f"pymoo.algorithms.soo.nonconvex.{sim_type.name.lower()}"
751             # Here is the difference made between updating methods
752         ),
753         sim_type.name,
754     )
755     n_var = len(parameters_to_update)
756
757     class MoProblem(ElementwiseProblem):
758         def __init__(self):
759             super().__init__(n_var=n_var, n_obj=1, xl=xl, xu=xu)
760
761         def _evaluate(self, x, out, *args, **kwargs):
762             out["F"] = objective(x)
763
764     problem = MoProblem()
765     algorithm = algorithm_class(pop_size=npts, **kwargs_optimizer)
766     res = opt.minimize(problem, algorithm, seed=seed, verbose=verbose)
767     solutions = [ind.x for ind in res.pop]
768     convergence = [ind.f for ind in res.pop]
769     st_pts = np.ones((npts, n_var))
770
771     sols = np.vstack(solutions)
772     convergence = np.array(convergence).reshape(-1, 1)
773
774     df_data = np.concatenate([convergence, st_pts, sols], axis=1)
775     cols = np.concatenate([np.array(["Convergence"]), params_st, params_opt])
776     df_opt = pd.DataFrame(data=df_data, columns=cols)
777
778     return df_opt

```

The discrete Euler-Bernoulli beam model (model only)

```

1 from abc import ABC, abstractmethod
2 from copy import deepcopy
3 from dataclasses import dataclass
4 from typing import List, Union, Tuple, Dict, Optional
5
6 import numpy as np
7 from scipy.optimize import brentq
8
9 from utils.beam_utils import nullspace
10
11
12 def w_z(beta, z: Union[float, np.ndarray], dz: int = 0):
13     if dz == 0:
14         return np.array(
15             [np.cosh(beta * z), np.sinh(beta * z), np.cos(beta * z), np.sin(beta * z)]
16         )
17     elif dz == 1:
18         return beta * np.array(
19             [np.sinh(beta * z), np.cosh(beta * z), -np.sin(beta * z), np.cos(beta * z)]
20         )
21     elif dz == 2:
22         return beta ** 2 * np.array(
23             [np.cosh(beta * z), np.sinh(beta * z), -np.cos(beta * z), -np.sin(beta * z)]
24         )
25     elif dz == 3:
26         return beta ** 3 * np.array(
27             [np.sinh(beta * z), np.cosh(beta * z), np.sin(beta * z), -np.cos(beta * z)]
28         )
29
30

```

```

31 @dataclass
32 class Beam1D(ABC):
33     params: Dict[str, float]
34     height: Union[float, int]
35     bc: Optional[Tuple[str, str]] = ("fixed", "free")
36
37     @abstractmethod
38     def evaluate_bc(
39         self,
40         beta: float,
41         factor: float,
42         z: Union[np.ndarray, float],
43         flag: str
44     ):
45         pass
46
47     @abstractmethod
48     def get_beta(self,
49                 omega: float) -> float:
50         pass
51
52
53 @dataclass
54 class BernulliBeam(Beam1D):
55
56     def mode_shapes(self):
57         pass
58
59     def determinant_boundary_cond(
60         self,
61         beta: float,
62         z_e: float,
63         z_s: float = 0.,
64         build: bool = False,
65     ):
66         mm = np.zeros((4, 4))
67
68         bc_s, bc_e = self.bc
69
70         mm[:2, :] = self.evaluate_bc(beta=beta, z=z_s, flag=bc_s)
71         mm[2:, :] = self.evaluate_bc(beta=beta, z=z_e, flag=bc_e)
72
73         mm /= np.max(np.abs(mm), axis=1).reshape(-1, 1)
74
75         if build:
76             return mm
77         else:
78             return np.linalg.det(mm)
79
80     def get_beta(self, omega: float):
81         return np.real((omega ** 2 * self.params["rho"] * self.params["A"] *
82                        self.params["L"]**4) / self.params["ei"]) ** (1 / 4)
83
84     def factor_continuity(self):
85         factor = 1 / (self.params["ei"])
86         return factor
87
88     def evaluate_bc(
89         self,
90         beta: float,
91         factor: float,
92         z: Union[np.ndarray, float],
93         flag: str
94     ):
95         if flag.lower() == "continuity+" or flag.lower() == "continuity-":
96             mm = np.zeros([4, 4])
97         else:
98             mm = np.zeros([2, 4])
99         if flag.lower() == "fixed":
100             mm[0, :] = w_z(beta, z=z)
101             mm[1, :] = w_z(beta, z=z, dz=1)

```

```

102         elif flag.lower() == "hinged":
103             mm[0, :] = w_z(beta, z=z)
104             mm[1, :] = w_z(beta, z=z, dz=2)
105         elif flag.lower() == "rot+transl":
106             mm[0, :] = w_z(beta, z=z, dz=2) - ((self.params["kr"] *
107             self.params["L"]) / self.params["ei"]) *
108             w_z(beta, z=z, dz=1)
109             mm[1, :] = w_z(beta, z=z, dz=3) + (self.params["kt"] *
110             self.params["L"]**3 / self.params["ei"]) *
111             w_z(beta, z=z)
112         elif flag.lower() == "rot":
113             mm[0, :] = w_z(beta, z=z, dz=2) -
114             (self.params["kr"] /
115             (self.params["ei"] / self.params["L"])) *
116             w_z(beta, z=z, dz=1)
117             mm[1, :] = w_z(beta, z=z)
118         elif flag.lower() == "transl":
119             mm[0, :] = w_z(beta, z=z, dz=1)
120             mm[1, :] = w_z(beta, z=z, dz=3) +
121             (self.params["kt"] * self.params["L"]**3 /
122             self.params["ei"]) * w_z(beta, z=z)
123         elif flag.lower() == "free":
124             mm[0, :] = w_z(beta, z=z, dz=2)
125             mm[1, :] = w_z(beta, z=z, dz=3)
126
127         elif flag.lower() == "continuity+":
128             mm[0, :] = w_z(beta, z=z)
129             mm[1, :] = w_z(beta, z=z, dz=1)
130             mm[2, :] = w_z(beta, z=z, dz=2)
131             mm[3, :] = w_z(beta, z=z, dz=3)
132         elif flag.lower() == "continuity-":
133             mm[0, :] = -w_z(beta, z=z)
134             mm[1, :] = -w_z(beta, z=z, dz=1)
135             mm[2, :] = -w_z(beta, z=z, dz=2) * self.params["ei"] * factor
136             mm[3, :] = -w_z(beta, z=z, dz=3) * self.params["ei"] * factor
137         return mm
138
139
140 @dataclass
141 class Model:
142     def update_dict(self, attr: str, new_dict: dict):
143         for k, val in new_dict.items():
144             getattr(self, attr).update[k] = val
145
146     def set_dict(self, attr: str, new_dict: dict):
147         setattr(self, attr, new_dict)
148
149     def update_model(self):
150         pass
151
152
153 @dataclass
154 class PieceWiseBeam1D:
155     """ """
156
157     beams: List[Beam1D]
158     bc: tuple
159
160     def initialize_beams(self):
161
162         beam = BernulliBeam
163
164     def mode_shapes(
165         self,
166         min_fn: float = 0.002,
167         max_fn: float = 100,
168         pts: Union[np.ndarray, int] = 100,
169         n_intervals: int = 100,
170         n_modes: int = 3
171     ):
172         n_b = len(self.beams)

```

```

173
174 heights = np.cumsum([0] + [beam.height for beam in self.beams])
175 min_omega, max_omega = (min_fn * 2 * np.pi, max_fn * 2 * np.pi)
176 omega_vec = np.logspace(np.log10(min_omega), np.log10(max_omega), n_intervals)
177 fun_v = np.zeros(omega_vec.shape)
178 nsol = 0
179 idx_change = []
180 for ii, om_ii in enumerate(omega_vec):
181     fun_v[ii] = self.determinant_boundary_cond(omega=om_ii, z=heights)
182     if ii > 0 and np.sign(fun_v[ii]) != np.sign(fun_v[ii - 1]):
183         nsol += 1
184         idx_change.append(ii - 1)
185     if nsol == n_modes:
186         break
187
188 omega_sol = []
189
190 for ii in idx_change:
191     if len(omega_sol) == n_modes:
192         break
193     sol = brentq(
194         f=self.determinant_boundary_cond, a=omega_vec[ii], b=omega_vec[ii + 1],
195         args=(heights)
196     )
197     omega_sol.append(sol)
198
199 if len(omega_sol) < n_modes:
200     print(
201         "Solution are less than required number of modes."
202         "Try a larger interval or larger bounds."
203     )
204
205 if len(omega_sol) == 0:
206     print("Failed in obtaining solutions!!")
207
208 omega = np.array(omega_sol)
209 fn = omega / (2 * np.pi)
210
211 if isinstance(pts, int):
212     pts_pieces = np.linspace(heights[:-1], heights[1:], pts, endpoint=False).T
213     pts = pts_pieces.ravel()
214     pts = np.insert(pts, len(pts), heights[-1])
215
216 phi = np.zeros((len(pts), len(omega)))
217 pts = np.asarray(pts)
218 pts = pts.flatten()
219
220 coeff = np.zeros(((n_b * 4), 1))
221
222 for ii, om_ii in enumerate(omega):
223     mm = self.determinant_boundary_cond(
224         omega=om_ii, z=heights, build=True
225     )
226     rtol = 0.1
227     not_converge = True
228     while not_converge:
229         coeff = nullspace(mm, atol=0, rtol=rtol).reshape(-1, 1)
230         if coeff.shape == (0, 1) or coeff.shape[0] > 4 * len(self.beams):
231             rtol /= 10
232         else:
233             not_converge = False
234
235     for ix, beam in enumerate(self.beams):
236         idx_z = (pts >= heights[ix]) & (pts <= heights[ix + 1])
237
238         beta_ = beam.get_beta(omega=om_ii)
239         z_ix = pts[idx_z]
240         phi[idx_z, ii] = np.sum(np.real(
241             w_z(beta_, z=z_ix) * coeff[4 * ix:(4 * ix + 4)]), axis=0
242         )
243         phi_ix = phi[idx_z, ii]

```

```

244     phi /= phi[np.argmax(np.abs(phi), axis=0), np.arange(phi.shape[1])]
245
246     self.fn = fn
247     self.phi = phi
248
249     return fn, phi, pts
250
251 def determinant_boundary_cond(
252     self,
253     omega: float,
254     z: np.ndarray,
255     build: bool = False,
256 ):
257     n_b = len(self.beams)
258     mm = np.zeros((4 * n_b, 4 * n_b))
259
260     bc_s, bc_e = self.bc
261
262     f__ = []
263     for ix, beam in enumerate(self.beams):
264         f = beam.factor_continuity()
265         f__.append(f)
266         if ix == 0:
267             f_ = f__[ix]
268         else:
269             f_ = f__[ix - 1]
270         bc_s_ix = (bc_s if ix == 0 else "continuity+")
271         bc_e_ix = (bc_e if ix == (n_b - 1) else "continuity-")
272
273         _beta = beam.get_beta(omega=omega)
274         _mm_s = beam.evaluate_bc(beta=_beta, factor=f_, z=z[ix], flag=bc_s_ix)
275         _mm_e = beam.evaluate_bc(beta=_beta, factor=f_, z=z[ix + 1], flag=bc_e_ix)
276
277         c_ix = np.arange(4 * ix, (4 * ix + 4))
278         r_st = (ix if ix == 0 else 4 * ix - 2)
279
280         _mm = np.vstack([_mm_s, _mm_e])
281         r_ix = np.arange(r_st, r_st + _mm.shape[0])
282         mm[np.ix_(r_ix, c_ix)] = _mm
283
284     mm /= np.max(np.abs(mm), axis=1).reshape(-1, 1)
285
286     if build:
287         return mm
288     else:
289         return np.linalg.det(mm)
290

```

Replication of the article by Taciroglu et al. [2]

```

1 %load_ext autoreload
2 %autoreload 2
3 from copy import deepcopy
4 from pyvibe.models.Timoshenko_verification_article import Timo, TimoPieceWiseBeam1D
5 import numpy as np
6 import matplotlib.pyplot as plt
7 %matplotlib inline
8
9 # Verification study Timoshenko beam model
10 In this notebook, a verification study is conducted to check the Timoshenko code written. The
    article "Efficient_model Updating of a multi-storey frame and its foundation stiffness
    from earthquake records using a timoshenko beam model" is replicated.
11
12 ## Parameters
13
14 # Dimension Parameters:
15 L = 44 #[m]
16 Lt = np.array([11,11,11,11]) #[m]
17 B = 23 #[m]
18 D = 21 #[m]

```

```

19 A = B * D #[m2]
20 I = 1/12 * B * D**3 #[m4]
21 H = Lt/L #[-]
22
23 # Material properties:
24 rho = np.array([400,400,400,400]) #[kg/m3]
25 E = np.array([6.6e8,6.6e8,6.6e8,6.6e8]) #[N/m2]
26 G = np.array([2.75e8,2.75e8,2.75e8,2.75e8]) #[N/m2]
27 k = 0.85 #[-]
28
29 # Springs:
30 kr = 2.048e12 #[Nm/rad]
31 kt = 1.841e10 #[N/m]
32
33 # Boundary conditions:
34 bc = ("rot+transl", "free")
35
36 #Ground propertiesGs
37 Gs = 2.68e8 #[N/m]
38 rhos = 2.7e3 #[kg/m3]
39 modes = 3
40
41 ## Procedure
42
43 timoshenko = []
44 for ix in range(len(H)):
45     timoshenko.append(Timo(params = {"E": E[ix], "I": I, "L": L,
46         "rho": rho[ix], "A": A, "kG": k*G[ix], "Gs":Gs, "rhos":rhos,
47         "B": B, "D": D, "kr": kr, "kt": kt},height= H[ix]))
48 pb = TimoPieceWiseBeam1D(beams=timoshenko, bc=bc)
49 fn, phi, pts = pb.modeshapes(n_modes=modes)
50
51 normalization_factors = phi[-1, :]
52 phi = phi / normalization_factors
53
54 ## Results
55
56 print("The first three natural frequencies (in Hz) of the Timoshenko beam are:")
57 for ii in range(3):
58     print(f"Mode_{ii+1}: ", fn[ii])
59
60 print(fn)
61
62 plt.figure(figsize=(4, 6))
63
64 # Plot mode shapes
65 plt.plot(phi[:,0], pts, color='black', label='Mode_1', linewidth=1.8)
66 plt.plot(phi[:,1], pts, '--', color='firebrick', label='Mode_2', linewidth=1.8)
67 plt.plot(phi[:,2], pts, '-.', color='mediumblue', label='Mode_3', linewidth=1.8)
68
69 # Set limits and labels
70 plt.xlim(-1.5, 1.5)
71 plt.ylim(0, 1.1)
72 plt.xlabel('Normalized displacement (-)')
73 plt.ylabel('Normalized height (-)')
74
75 # Add legend and customize grid
76 plt.legend()
77 plt.grid(True, which='both', color='grey', linestyle=':', linewidth=1) # Customize grid
78 plt.gca().set_axisbelow(True) # Ensure grid is behind other plot elements
79
80 plt.gca().spines['top'].set_linewidth(0.5)
81 plt.gca().spines['top'].set_color('black')
82 plt.gca().spines['right'].set_linewidth(0.5)
83 plt.gca().spines['right'].set_color('black')
84 plt.gca().spines['bottom'].set_linewidth(0.5)
85 plt.gca().spines['bottom'].set_color('black')
86 plt.gca().spines['left'].set_linewidth(0.5)
87 plt.gca().spines['left'].set_color('black')
88 plt.show()

```

## Comparison with an Euler-Bernoulli beam

```

1 %load_ext autoreload
2 %autoreload 2
3 from pyvibe.models.Timoshenko import Timo, TimoPieceWiseBeam1D
4 import numpy as np
5 from pyvibe.models.Euler_Bernoulli import PieceWiseBeam1D, BernulliBeam
6 from pyvibe.models.Shear import Shear1D, Shear
7
8 %matplotlib inline
9 # Dimension Parameters:
10 L = 44 #[m]
11 Lt = np.array([11,11,11,11]) #[m]
12 B = 23 #[m]
13 D = 21 #[m]
14 A = B * D #[m2]
15 I = 1/12 * B * D**3
16 H = Lt/L
17
18 # Material properties:
19 rho = np.array([400,400,400,400]) #[kg/m3]
20 E = np.array([6.6e8,6.6e8,6.6e8,6.6e8]) #[N/m2]
21 G = np.array([2.75e8,2.75e8,2.75e8,2.75e8]) #[N/m2]
22 k = 0.85 #[-]
23
24 # Springs
25 kr = 2.048e12 #[Nm/rad]
26 kt = 1.841e10 #[N/m]
27
28 # Boundary conditions:
29 bc = ("rot+transl", "free")
30
31 # Scalar boundaries
32 amount_points = 50
33 left = 1
34 right = 100
35 # MAC
36 def adjust_modes(phi1, phi2):
37     rmse_plus = np.sqrt(np.mean((phi1 - phi2) ** 2))
38     rmse_minus = np.sqrt(np.mean((phi1 + phi2) ** 2))
39     flip = np.ones_like(phi2)
40     flip[rmse_plus > rmse_minus] = -1
41     phi_flip = phi1 * flip
42     return phi_flip
43
44 def modal_assurance_criterion(phi1, phi2):
45     phi2 = adjust_modes(phi2, phi1)
46     mac = np.abs(np.conj(phi1).dot(phi2))**2 / \
47         (np.conj(phi1).dot(phi1) * np.conj(phi2).dot(phi2))
48     return mac
49 timoshenko = []
50 for ix in range(len(H)):
51     timoshenko.append(Timo(params={"E": E[ix], "I": I, "L": L,
52     "rho": rho[ix], "A": A, "kG": k * G[ix], "kr": kr, "kt": kt}, height=H[ix]))
53 model = TimoPieceWiseBeam1D(beams=timoshenko, bc = bc)
54 fn, phi, pts = model.modeshapes()
55 change = np.logspace(np.log10(left), np.log10(right), amount_points)
56 euler = []
57 for ix in range(len(H)):
58     euler.append(BernulliBeam(params={"ei": E[ix]*I, "L": L,
59     "rho": rho[ix], "A": A, "kr": kr, "kt": kt}, height=H[ix]))
60 model = PieceWiseBeam1D(beams=euler, bc = bc)
61 fn_e, phi_e, pts_e = model.mode_shapes()
62
63
64 def change_kG(change, E, I, L, rho, A, k, G, kr, kt, H,
65     bc):
66     f1 = []
67     f2 = []
68     f3 = []

```



```

69 m1 = []
70 m2 = []
71 m3 = []
72
73 for value in change:
74     timoshenko = []
75     for ix in range(len(H)):
76         timoshenko.append(Timo(params={"E": E[ix], "I": I, "L": L,
77             "rho": rho[ix], "A": A, "kG": k * G[ix] * value, "kr": kr,
78             "kt": kt}, height=H[ix]))
79     model = TimoPieceWiseBeam1D(beams=timoshenko, bc = bc)
80     fn, phi, pts = model.modeshapes()
81
82     f1.append(fn[0])
83     f2.append(fn[1])
84     f3.append(fn[2])
85     m1.append(modal_assurance_criterion(phi_e[:, 0], phi[:, 0]))
86     m2.append(modal_assurance_criterion(phi_e[:, 1], phi[:, 1]))
87     m3.append(modal_assurance_criterion(phi_e[:, 2], phi[:, 2]))
88
89     return f1, f2, f3, m1, m2, m3
90
91 f1_kG, f2_kG, f3_kG, m1_kG, m2_kG, m3_kG = change_kG(change, E, I, L, rho, A, k, G, kr,
92     kt, H, bc)
93
94 import numpy as np
95 import matplotlib.pyplot as plt
96
97 def plot_elements(f1_kG, f2_kG, f3_kG, change, fn_e, t):
98     plt.figure(figsize=(7, 5))
99
100     fn_e_0_array = np.full_like(change, fn_e[0])
101     fn_e_1_array = np.full_like(change, fn_e[1])
102     fn_e_2_array = np.full_like(change, fn_e[2])
103
104     plt.plot(change, fn_e_0_array, color='black', linestyle='dotted', linewidth=1.3, zorder
105         =1)
106     plt.plot(change, fn_e_1_array, color='black', linestyle='dotted', linewidth=1.3, zorder
107         =1)
108     plt.plot(change, fn_e_2_array, color='black', linestyle='dotted', linewidth=1.3, zorder
109         =1)
110
111     plt.text(change[0], fn_e_0_array[0] + 1, f'Euler_{\omega_1}_{fn_e[0]:.2f} Hz', fontsize
112         =10, color='black', zorder=2, fontname='Arial')
113     plt.text(change[0], fn_e_1_array[0] + 1, f'Euler_{\omega_2}_{fn_e[1]:.2f} Hz', fontsize
114         =10, color='black', zorder=2, fontname='Arial')
115     plt.text(change[0], fn_e_2_array[0] + 1, f'Euler_{\omega_3}_{fn_e[2]:.2f} Hz', fontsize
116         =10, color='black', zorder=2, fontname='Arial')
117
118     plt.scatter(change, f1_kG, color='black', linewidth=1.3,
119         label='Timoshenko_{\omega_1}$', s=9)
120     plt.scatter(change, f2_kG, color='firebrick', linewidth=1.3,
121         label='Timoshenko_{\omega_2}$', s=9)
122     plt.scatter(change, f3_kG, color='mediumblue', linewidth=1.3,
123         label='Timoshenko_{\omega_3}$', s=9)
124
125     plt.xlabel('Scalar applied on the initial value of parameter $G$ (-)', fontsize=12,
126         fontname='Arial')
127     plt.ylabel('Natural frequency $f_n$ (Hz)', fontsize=12, fontname='Arial')
128     plt.ylim(-1, 30)
129     plt.xlim(0.9, 110)
130     plt.gca().set_facecolor('white')
131     plt.grid(True, which='both', color='grey', linestyle=':', linewidth=0.5)
132     plt.xscale('log')
133     #plt.title('Model output Timoshenko and Euler-Bernoulli beam model', fontsize=14,
134         fontname='Arial')
135
136     plt.legend(fontsize=10, prop={'family': 'Arial'})
137     plt.gca().spines['top'].set_linewidth(0.5)
138     plt.gca().spines['top'].set_color('black')
139     plt.gca().spines['right'].set_linewidth(0.5)
140     plt.gca().spines['right'].set_color('black')

```

```

131 plt.gca().spines['bottom'].set_linewidth(0.5)
132 plt.gca().spines['bottom'].set_color('black')
133 plt.gca().spines['left'].set_linewidth(0.5)
134 plt.gca().spines['left'].set_color('black')
135 plt.gca().set_xticks([1, 10, 100])
136 plt.gca().set_xticklabels(['1', '10', '100'], fontname='Arial')
137 plt.show()
138 plot_elements(f1_kG, f2_kG, f3_kG, change, fn_e, 2)

```

## Models

```

1 %load_ext autoreload
2 %autoreload 2
3 from copy import deepcopy
4 from pyvibe.models.Timoshenko import Timo, TimoPieceWiseBeam1D, TimoPieceWiseBeam2D,
   UpdatingMethod, cost_func1D, cost_func2D, update_model
5 import numpy as np
6 from pyvibe.models.Euler_Bernoulli import PieceWiseBeam1D, BernulliBeam
7 import matplotlib.pyplot as plt
8 %matplotlib inline
9
10 ## Parameters
11
12 # Dimension Parameters:
13 L = 160.58 #[m]
14 Lt = np.array([5.735, 11.47, 26.640, 89.91, 26.825]) #[m]
15 B = 28 #[m]
16 D = 28 #[m]
17 A = B * D #[m2]
18 I_x = np.array([619.21, 948.47, 2727.22, 2335.82, 905.52]) #[m4]
19 I_y = np.array([1536.25, 959.01, 1245.46, 1202.07, 831.57]) #[m4]
20 I_yy = np.array([1444, 1500, 1393, 1389, 1293]) #[m4]
21 I_xx = np.array([1532, 1548, 2796, 2324, 760]) #[m4]
22 H = Lt/L #[-]
23
24 # Material properties:
25 rho = np.array([1933, 495, 486, 440, 383]) #[kg/m3]
26 E = np.array([38.2e9, 38.2e9, 38.2e9, 38.2e9, 32.8e9]) #[N/m2]
27 v = 0.2 #[-]
28 G = E/(2*(1+v)) #[N/m^2]
29 k = 0.85 #[-]
30
31 # Springs:
32 kr_x = 2380e9 #[Nm/rad]
33 kr_y = 2625e9 #[Nm/rad]
34 kt_y = 19e9 #[N/m]
35 kt_x = 4e9 #[N/m]
36
37 #boundary conditions:
38 bc = ("rot+transl", "free")
39
40 ## Procedure
41
42 timoshenko_x = []
43 for ix in range(len(H)):
44     timoshenko_x.append(Timo(params={"E": E[ix], "I": I_xx[ix], "L": L, "rho": rho[ix], "A": A,
45     , "kG": k * G[ix], "kr": kr_x, "kt": kt_x}, height=H[ix]))
46 timoshenko_y = []
47 for ix in range(len(H)):
48     timoshenko_y.append(Timo(params={"E": E[ix], "I": I_yy[ix], "L": L, "rho": rho[ix], "A": A,
49     , "kG": k * G[ix], "kr": kr_y, "kt": kt_y}, height=H[ix]))
50 model2D = TimoPieceWiseBeam2D(beams_x=timoshenko_x, beams_y=timoshenko_y, bc = bc)
51 fn_x, fn_y, phi_x, phi_y, pts_x, pts_y = model2D.modeshapes()
52
53 print("The first two natural frequencies (in Hz) in x-direction are:")
54 for ii in range(2):
55     print(f"Mode {ii+1}: ", fn_x[ii])
56
57 print("The first three natural frequencies (in Hz) in y-direction are:")
58 for ii in range(3):

```

```

57     print(f"Mode_{ii+1}_:", fn_y[ii])
58
59 euler_x = []
60 for ix in range(len(H)):
61     euler_x.append(BernulliBeam(params={"ei": E[ix]*I_xx[ix], "L": L,"rho": rho[ix], "A": A ,
62         "kr": kr_x, "kt": kt_x}, height=H[ix]))
63 model_x = PieceWiseBeam1D(beams=euler_x, bc = bc)
64 fn_xe, phi_xe, pts_xe = model_x.mode_shapes()
65 euler_y = []
66 for ix in range(len(H)):
67     euler_y.append(BernulliBeam(params={"ei": E[ix]*I_yy[ix], "L": L,"rho": rho[ix], "A": A ,
68         "kr": kr_y, "kt": kt_y}, height=H[ix]))
69 model_y = PieceWiseBeam1D(beams=euler_y, bc = bc)
70 fn_ye, phi_yie, pts_ye = model_y.mode_shapes()
71
72 print("The first two natural frequencies (in Hz) in x-direction are:")
73 for ii in range(2):
74     print(f"Mode_{ii+1}_:", fn_xe[ii])
75
76 print("The first three natural frequencies (in Hz) in y-direction are:")
77 for ii in range(3):
78     print(f"Mode_{ii+1}_:", fn_ye[ii])
79
80 # Dimension Parameters:
81 L = 160.58 #[m]
82 Lt = np.array([160.58]) #[m]
83 B = 28 #[m]
84 D = 28 #[m]
85 A = B * D #[m2]
86 I_yy = np.array([1384]) #[m4]
87 I_xx = np.array([2057]) #[m4]
88 H = Lt/L #[-]
89
90 # Material properties:
91 rho = np.array([545]) #[kg/m3]
92 E = np.array([37.3e9]) #[N/m2]
93 v = 0.2 #[-]
94 G = E/(2*(1+v)) #[N/m^2]
95 k = 0.85 #[-]
96
97 # Springs:
98 kr_x = 2380e9 #[Nm/rad]
99 kr_y = 2625e9 #[Nm/rad]
100 kt_y = 19e9 #[N/m]
101 kt_x = 4e9 #[N/m]
102
103 #boundary conditions:
104 bc = ("rot+transl", "free")
105
106 timoshenko_xu = []
107 for ix in range(len(H)):
108     timoshenko_xu.append(Timo(params={"E": E[ix], "I": I_xx[ix], "L": L,"rho": rho[ix], "A": A ,
109         "kG": k * G[ix], "kr": kr_x, "kt": kt_x}, height=H[ix]))
110 timoshenko_yu = []
111 for ix in range(len(H)):
112     timoshenko_yu.append(Timo(params={"E": E[ix], "I": I_yy[ix], "L": L,"rho": rho[ix], "A": A ,
113         "kG": k * G[ix], "kr": kr_y, "kt": kt_y}, height=H[ix]))
114 model2D = TimoPieceWiseBeam2D(beams_x=timoshenko_xu, beams_y=timoshenko_yu, bc = bc)
115 fn_xu, fn_yu, phi_xu, phi_yu, pts_xu, pts_yu = model2D.modeshapes()
116 print(timoshenko_yu)
117 print("The first two natural frequencies (in Hz) in x-direction are:")
118 for ii in range(2):
119     print(f"Mode_{ii+1}_:", fn_xu[ii])
120
121 print("The first three natural frequencies (in Hz) in y-direction are:")
122 for ii in range(3):
123     print(f"Mode_{ii+1}_:", fn_yu[ii])

```

## Input Parameters

```

1 from pyvibe.models.Timoshenko import Timo, TimoPieceWiseBeam1D, TimoPieceWiseBeam2D,
   UpdatingMethod, cost_func1D, cost_func2D, update_model
2 import numpy as np
3 import matplotlib.pyplot as plt
4
5 # MAC
6 def adjust_modes(phi1, phi2):
7     rmse_plus = np.sqrt(np.mean((phi1 - phi2) ** 2))
8     rmse_minus = np.sqrt(np.mean((phi1 + phi2) ** 2))
9     flip = np.ones_like(phi2)
10    flip[rmse_plus > rmse_minus] = -1
11    phi_flip = phi1 * flip
12    return phi_flip
13
14 def modal_assurance_criterion(phi1, phi2):
15     phi2 = adjust_modes(phi2, phi1)
16     mac = np.abs(np.conj(phi1).dot(phi2))**2 / \
17           (np.conj(phi1).dot(phi1) * np.conj(phi2).dot(phi2))
18     return mac
19 #
20 -----
21 # Parameter kr
22 def change_kr(change, fn_x, fn_y, phi_x, phi_y, E_x, E_y, I_x, I_y, L, rho, A, k, G_x, G_y,
23               kr_x, kr_y, kt_x, kt_y, H,
24               bc):
25     Change_f1_x = []
26     Change_f2_x = []
27     Change_f3_x = []
28     Change_f1_y = []
29     Change_f2_y = []
30     Change_f3_y = []
31     Change_m1_x = []
32     Change_m2_x = []
33     Change_m3_x = []
34     Change_m1_y = []
35     Change_m2_y = []
36     Change_m3_y = []
37
38     for value in change:
39         kr_new_x = kr_x * value
40         kr_new_y = kr_y * value
41         timoshenko_x = []
42         for ix in range(len(H)):
43             timoshenko_x.append(Timo(
44                 params={"E": E_x[ix], "I": I_x[ix], "L": L, "rho": rho[ix], "A": A[ix], "kG":
45                       k * G_x[ix], "kr": kr_new_x,
46                       "kt": kt_x}, height=H[ix]))
47         timoshenko_y = []
48         for ix in range(len(H)):
49             timoshenko_y.append(Timo(
50                 params={"E": E_y[ix], "I": I_y[ix], "L": L, "rho": rho[ix], "A": A[ix], "kG":
51                       k * G_y[ix], "kr": kr_new_y,
52                       "kt": kt_y}, height=H[ix]))
53
54     model2D = TimoPieceWiseBeam2D(beams_x=timoshenko_x, beams_y=timoshenko_y, bc=bc)
55     fn_x_kr, fn_y_kr, phi_x_kr, phi_y_kr, pts_x_kr, pts_y_kr = model2D.modeshapes()
56
57     Change_f1_x.append(((fn_x_kr[0] - fn_x[0]) / fn_x[0]) * 100)
58     Change_f2_x.append(((fn_x_kr[1] - fn_x[1]) / fn_x[1]) * 100)
59     Change_f3_x.append(((fn_x_kr[2] - fn_x[2]) / fn_x[2]) * 100)
60     Change_f1_y.append(((fn_y_kr[0] - fn_y[0]) / fn_y[0]) * 100)
61     Change_f2_y.append(((fn_y_kr[1] - fn_y[1]) / fn_y[1]) * 100)
62     Change_f3_y.append(((fn_y_kr[2] - fn_y[2]) / fn_y[2]) * 100)
63
64     Change_m1_x.append(modal_assurance_criterion(phi_x[:, 0], phi_x_kr[:, 0]))
65     Change_m2_x.append(modal_assurance_criterion(phi_x[:, 1], phi_x_kr[:, 1]))
66     Change_m3_x.append(modal_assurance_criterion(phi_x[:, 2], phi_x_kr[:, 2]))

```

```

63     Change_m1_y.append(modal_assurance_criterion(phi_y[:, 0], phi_y_kr[:, 0]))
64     Change_m2_y.append(modal_assurance_criterion(phi_y[:, 1], phi_y_kr[:, 1]))
65     Change_m3_y.append(modal_assurance_criterion(phi_y[:, 2], phi_y_kr[:, 2]))
66     return Change_f1_x, Change_f2_x, Change_f3_x, Change_f1_y, Change_f2_y, Change_f3_y,
        Change_m1_x, Change_m2_x, Change_m3_x, Change_m1_y, Change_m2_y, Change_m3_y
67
68 # Parameter kt
69 def change_kt(change, fn_x, fn_y, phi_x, phi_y, E_x, E_y, I_x, I_y, L, rho, A, k, G_x, G_y,
        kr_x, kr_y, kt_x, kt_y, H,
70             bc):
71     Change_f1_x = []
72     Change_f2_x = []
73     Change_f3_x = []
74     Change_f1_y = []
75     Change_f2_y = []
76     Change_f3_y = []
77     Change_m1_x = []
78     Change_m2_x = []
79     Change_m3_x = []
80     Change_m1_y = []
81     Change_m2_y = []
82     Change_m3_y = []
83
84     for value in change:
85         kt_new_x = kt_x * value
86         kt_new_y = kt_y * value
87         timoshenko_x = []
88         for ix in range(len(H)):
89             timoshenko_x.append(Timo(
90                 params={"E": E_x[ix], "I": I_x[ix], "L": L, "rho": rho[ix], "A": A[ix], "kG":
                        k * G_x[ix], "kr": kr_x,
91                         "kt": kt_new_x}, height=H[ix]))
92         timoshenko_y = []
93         for ix in range(len(H)):
94             timoshenko_y.append(Timo(
95                 params={"E": E_y[ix], "I": I_y[ix], "L": L, "rho": rho[ix], "A": A[ix], "kG":
                        k * G_y[ix], "kr": kr_y,
96                         "kt": kt_new_y}, height=H[ix]))
97
98     model2D = TimoPieceWiseBeam2D(beams_x=timoshenko_x, beams_y=timoshenko_y, bc=bc)
99     fn_x_kt, fn_y_kt, phi_x_kt, phi_y_kt, pts_x_kt, pts_y_kt = model2D.modeshapes()
100
101     Change_f1_x.append(((fn_x_kt[0] - fn_x[0]) / fn_x[0]) * 100)
102     Change_f2_x.append(((fn_x_kt[1] - fn_x[1]) / fn_x[1]) * 100)
103     Change_f3_x.append(((fn_x_kt[2] - fn_x[2]) / fn_x[2]) * 100)
104     Change_f1_y.append(((fn_y_kt[0] - fn_y[0]) / fn_y[0]) * 100)
105     Change_f2_y.append(((fn_y_kt[1] - fn_y[1]) / fn_y[1]) * 100)
106     Change_f3_y.append(((fn_y_kt[2] - fn_y[2]) / fn_y[2]) * 100)
107
108     Change_m1_x.append(modal_assurance_criterion(phi_x[:, 0], phi_x_kt[:, 0]))
109     Change_m2_x.append(modal_assurance_criterion(phi_x[:, 1], phi_x_kt[:, 1]))
110     Change_m3_x.append(modal_assurance_criterion(phi_x[:, 2], phi_x_kt[:, 2]))
111     Change_m1_y.append(modal_assurance_criterion(phi_y[:, 0], phi_y_kt[:, 0]))
112     Change_m2_y.append(modal_assurance_criterion(phi_y[:, 1], phi_y_kt[:, 1]))
113     Change_m3_y.append(modal_assurance_criterion(phi_y[:, 2], phi_y_kt[:, 2]))
114     return Change_f1_x, Change_f2_x, Change_f3_x, Change_f1_y, Change_f2_y, Change_f3_y,
        Change_m1_x, Change_m2_x, Change_m3_x, Change_m1_y, Change_m2_y, Change_m3_y
115
116 # Parameter I
117 def change_I(change, fn_x, fn_y, phi_x, phi_y, E_x, E_y, I_x, I_y, L, rho, A, k, G_x, G_y,
        kr_x, kr_y, kt_x, kt_y, H,
118             bc):
119     Change_f1_x = []
120     Change_f2_x = []
121     Change_f3_x = []
122     Change_f1_y = []
123     Change_f2_y = []
124     Change_f3_y = []
125     Change_m1_x = []
126     Change_m2_x = []
127     Change_m3_x = []

```

```

128 Change_m1_y = []
129 Change_m2_y = []
130 Change_m3_y = []
131
132 for value in change:
133     print(value)
134     I_new_x = I_x * value
135     I_new_y = I_y * value
136     timoshenko_x = []
137     for ix in range(len(H)):
138         timoshenko_x.append(Timo(
139             params={"E": E_x[ix], "I": I_new_x[ix], "L": L, "rho": rho[ix], "A": A[ix], "
140                     kG": k * G_x[ix], "kr": kr_x,
141                     "kt": kt_x}, height=H[ix]))
142     timoshenko_y = []
143     for ix in range(len(H)):
144         timoshenko_y.append(Timo(
145             params={"E": E_y[ix], "I": I_new_y[ix], "L": L, "rho": rho[ix], "A": A[ix], "
146                     kG": k * G_y[ix], "kr": kr_y,
147                     "kt": kt_y}, height=H[ix]))
148
149 model2D = TimoPieceWiseBeam2D(beams_x=timoshenko_x, beams_y=timoshenko_y, bc=bc)
150 fn_x_I, fn_y_I, phi_x_I, phi_y_I, pts_x_I, pts_y_I = model2D.modeshapes()
151 print(fn_x_I)
152
153 Change_f1_x.append(((fn_x_I[0] - fn_x[0]) / fn_x[0]) * 100)
154 Change_f2_x.append(((fn_x_I[1] - fn_x[1]) / fn_x[1]) * 100)
155 Change_f3_x.append(((fn_x_I[2] - fn_x[2]) / fn_x[2]) * 100)
156 Change_f1_y.append(((fn_y_I[0] - fn_y[0]) / fn_y[0]) * 100)
157 Change_f2_y.append(((fn_y_I[1] - fn_y[1]) / fn_y[1]) * 100)
158 Change_f3_y.append(((fn_y_I[2] - fn_y[2]) / fn_y[2]) * 100)
159
160 Change_m1_x.append(modal_assurance_criterion(phi_x[:, 0], phi_x_I[:, 0]))
161 Change_m2_x.append(modal_assurance_criterion(phi_x[:, 1], phi_x_I[:, 1]))
162 Change_m3_x.append(modal_assurance_criterion(phi_x[:, 2], phi_x_I[:, 2]))
163 Change_m1_y.append(modal_assurance_criterion(phi_y[:, 0], phi_y_I[:, 0]))
164 Change_m2_y.append(modal_assurance_criterion(phi_y[:, 1], phi_y_I[:, 1]))
165 Change_m3_y.append(modal_assurance_criterion(phi_y[:, 2], phi_y_I[:, 2]))
166
167 return Change_f1_x, Change_f2_x, Change_f3_x, Change_f1_y, Change_f2_y, Change_f3_y,
168        Change_m1_x, Change_m2_x, Change_m3_x, Change_m1_y, Change_m2_y, Change_m3_y
169
170 # Parameter E
171 def change_E(change, fn_x, fn_y, phi_x, phi_y, E_x, E_y, I_x, I_y, L, rho, A, k, G_x, G_y,
172              kr_x, kr_y, kt_x, kt_y, H,
173              bc):
174     Change_f1_x = []
175     Change_f2_x = []
176     Change_f3_x = []
177     Change_f1_y = []
178     Change_f2_y = []
179     Change_f3_y = []
180     Change_m1_x = []
181     Change_m2_x = []
182     Change_m3_x = []
183     Change_m1_y = []
184     Change_m2_y = []
185     Change_m3_y = []
186
187     for value in change:
188         E_new_x = E_x * value
189         E_new_y = E_y * value
190         timoshenko_x = []
191         for ix in range(len(H)):
192             timoshenko_x.append(Timo(
193                 params={"E": E_new_x[ix], "I": I_x[ix], "L": L, "rho": rho[ix], "A": A[ix], "

```

```

194         params={"E": E_new_y[ix], "I": I_y[ix], "L": L, "rho": rho[ix], "A": A[ix], "
195               kG": k * G_y[ix], "kr": kr_y,
196               "kt": kt_y}, height=H[ix]))
197
198     model2D = TimoPieceWiseBeam2D(beams_x=timoshenko_x, beams_y=timoshenko_y, bc=bc)
199     fn_x_E, fn_y_E, phi_x_E, phi_y_E, pts_x_E, pts_y_E = model2D.modeshapes()
200
201     Change_f1_x.append(((fn_x_E[0] - fn_x[0]) / fn_x[0]) * 100)
202     Change_f2_x.append(((fn_x_E[1] - fn_x[1]) / fn_x[1]) * 100)
203     Change_f3_x.append(((fn_x_E[2] - fn_x[2]) / fn_x[2]) * 100)
204     Change_f1_y.append(((fn_y_E[0] - fn_y[0]) / fn_y[0]) * 100)
205     Change_f2_y.append(((fn_y_E[1] - fn_y[1]) / fn_y[1]) * 100)
206     Change_f3_y.append(((fn_y_E[2] - fn_y[2]) / fn_y[2]) * 100)
207
208     Change_m1_x.append(modal_assurance_criterion(phi_x[:, 0], phi_x_E[:, 0]))
209     Change_m2_x.append(modal_assurance_criterion(phi_x[:, 1], phi_x_E[:, 1]))
210     Change_m3_x.append(modal_assurance_criterion(phi_x[:, 2], phi_x_E[:, 2]))
211     Change_m1_y.append(modal_assurance_criterion(phi_y[:, 0], phi_y_E[:, 0]))
212     Change_m2_y.append(modal_assurance_criterion(phi_y[:, 1], phi_y_E[:, 1]))
213     Change_m3_y.append(modal_assurance_criterion(phi_y[:, 2], phi_y_E[:, 2]))
214
215     return Change_f1_x, Change_f2_x, Change_f3_x, Change_f1_y, Change_f2_y, Change_f3_y,
216           Change_m1_x, Change_m2_x, Change_m3_x, Change_m1_y, Change_m2_y, Change_m3_y
217
218 # Parameter kG
219 def change_kG(change, fn_x, fn_y, phi_x, phi_y, E_x, E_y, I_x, I_y, L, rho, A, k, G_x, G_y,
220               kr_x, kr_y, kt_x, kt_y, H,
221               bc):
222     Change_f1_x = []
223     Change_f2_x = []
224     Change_f3_x = []
225     Change_f1_y = []
226     Change_f2_y = []
227     Change_f3_y = []
228     Change_m1_x = []
229     Change_m2_x = []
230     Change_m3_x = []
231     Change_m1_y = []
232     Change_m2_y = []
233     Change_m3_y = []
234
235     for value in change:
236         kG_new_x = k * G_x * value
237         kG_new_y = k * G_y * value
238         timoshenko_x = []
239         for ix in range(len(H)):
240             timoshenko_x.append(Timo(
241                 params={"E": E_x[ix], "I": I_x[ix], "L": L, "rho": rho[ix], "A": A[ix], "kG":
242                       kG_new_x[ix], "kr": kr_x,
243                       "kt": kt_x}, height=H[ix]))
244         timoshenko_y = []
245         for ix in range(len(H)):
246             timoshenko_y.append(Timo(
247                 params={"E": E_y[ix], "I": I_y[ix], "L": L, "rho": rho[ix], "A": A[ix], "kG":
248                       kG_new_y[ix], "kr": kr_y,
249                       "kt": kt_y}, height=H[ix]))
250
251     model2D = TimoPieceWiseBeam2D(beams_x=timoshenko_x, beams_y=timoshenko_y, bc=bc)
252     fn_x_kG, fn_y_kG, phi_x_kG, phi_y_kG, pts_x_kG, pts_y_kG = model2D.modeshapes()
253
254     Change_f1_x.append(((fn_x_kG[0] - fn_x[0]) / fn_x[0]) * 100)
255     Change_f2_x.append(((fn_x_kG[1] - fn_x[1]) / fn_x[1]) * 100)
256     Change_f3_x.append(((fn_x_kG[2] - fn_x[2]) / fn_x[2]) * 100)
257     Change_f1_y.append(((fn_y_kG[0] - fn_y[0]) / fn_y[0]) * 100)
258     Change_f2_y.append(((fn_y_kG[1] - fn_y[1]) / fn_y[1]) * 100)
259     Change_f3_y.append(((fn_y_kG[2] - fn_y[2]) / fn_y[2]) * 100)
260
261     Change_m1_x.append(modal_assurance_criterion(phi_x[:, 0], phi_x_kG[:, 0]))
262     Change_m2_x.append(modal_assurance_criterion(phi_x[:, 1], phi_x_kG[:, 1]))
263     Change_m3_x.append(modal_assurance_criterion(phi_x[:, 2], phi_x_kG[:, 2]))
264     Change_m1_y.append(modal_assurance_criterion(phi_y[:, 0], phi_y_kG[:, 0]))
265     Change_m2_y.append(modal_assurance_criterion(phi_y[:, 1], phi_y_kG[:, 1]))

```

```

260     Change_m3_y.append(modal_assurance_criterion(phi_y[:, 2], phi_y_kG[:, 2]))
261     return Change_f1_x, Change_f2_x, Change_f3_x, Change_f1_y, Change_f2_y, Change_f3_y,
262         Change_m1_x, Change_m2_x, Change_m3_x, Change_m1_y, Change_m2_y, Change_m3_y
263 # Parameter rho
264 def change_rho(change, fn_x, fn_y, phi_x, phi_y, E_x, E_y, I_x, I_y, L, rho, A, k, G_x, G_y,
265     kr_x, kr_y, kt_x, kt_y, H,
266     bc):
267     Change_f1_x = []
268     Change_f2_x = []
269     Change_f3_x = []
270     Change_f1_y = []
271     Change_f2_y = []
272     Change_f3_y = []
273     Change_m1_x = []
274     Change_m2_x = []
275     Change_m3_x = []
276     Change_m1_y = []
277     Change_m2_y = []
278     Change_m3_y = []
279     for value in change:
280         rho_new = rho * value
281         timoshenko_x = []
282         for ix in range(len(H)):
283             timoshenko_x.append(Timo(
284                 params={"E": E_x[ix], "I": I_x[ix], "L": L, "rho": rho_new[ix], "A": A[ix], "
285                     kG": k * G_x[ix], "kr": kr_x,
286                     "kt": kt_x}, height=H[ix]))
287         timoshenko_y = []
288         for ix in range(len(H)):
289             timoshenko_y.append(Timo(
290                 params={"E": E_y[ix], "I": I_y[ix], "L": L, "rho": rho_new[ix], "A": A[ix], "
291                     kG": k * G_y[ix], "kr": kr_y,
292                     "kt": kt_y}, height=H[ix]))
293         model2D = TimoPieceWiseBeam2D(beams_x=timoshenko_x, beams_y=timoshenko_y, bc=bc)
294         fn_x_rho, fn_y_rho, phi_x_rho, phi_y_rho, pts_x_rho, pts_y_rho = model2D.modeshapes()
295         Change_f1_x.append(((fn_x_rho[0] - fn_x[0]) / fn_x[0]) * 100)
296         Change_f2_x.append(((fn_x_rho[1] - fn_x[1]) / fn_x[1]) * 100)
297         Change_f3_x.append(((fn_x_rho[2] - fn_x[2]) / fn_x[2]) * 100)
298         Change_f1_y.append(((fn_y_rho[0] - fn_y[0]) / fn_y[0]) * 100)
299         Change_f2_y.append(((fn_y_rho[1] - fn_y[1]) / fn_y[1]) * 100)
300         Change_f3_y.append(((fn_y_rho[2] - fn_y[2]) / fn_y[2]) * 100)
301
302         Change_m1_x.append(modal_assurance_criterion(phi_x[:, 0], phi_x_rho[:, 0]))
303         Change_m2_x.append(modal_assurance_criterion(phi_x[:, 1], phi_x_rho[:, 1]))
304         Change_m3_x.append(modal_assurance_criterion(phi_x[:, 2], phi_x_rho[:, 2]))
305         Change_m1_y.append(modal_assurance_criterion(phi_y[:, 0], phi_y_rho[:, 0]))
306         Change_m2_y.append(modal_assurance_criterion(phi_y[:, 1], phi_y_rho[:, 1]))
307         Change_m3_y.append(modal_assurance_criterion(phi_y[:, 2], phi_y_rho[:, 2]))
308     return Change_f1_x, Change_f2_x, Change_f3_x, Change_f1_y, Change_f2_y, Change_f3_y,
309         Change_m1_x, Change_m2_x, Change_m3_x, Change_m1_y, Change_m2_y, Change_m3_y
310 -----
311
312 import matplotlib.pyplot as plt
313
314 import matplotlib.pyplot as plt
315 import numpy as np
316
317 def plot_per_parameter_influence(type, parameter, direction, parameter_change, *data):
318     plt.rcParams['font.family'] = 'Arial'
319
320     plt.figure(figsize=(7, 5))
321     colors = ['black', 'firebrick', 'steelblue']
322     linestyle = ['-', '-', '-']
323     mode = ['Mode1x', 'Mode1y', 'Mode2y']
324     for mod, color, linestyle, data_array in zip(mode, colors, linestyle, data):
325         if type == 'frequency':

```



```

326         if isinstance(data_array[0], (list, np.ndarray)):
327             abs_data_array = [[abs(val) for val in sublist] for sublist in data_array]
328         else:
329             abs_data_array = [abs(val) for val in data_array]
330         plt.plot(parameter_change, abs_data_array, label=mod, color=color, linestyle=
            linestyle, linewidth=1.2)
331     else:
332         plt.plot(parameter_change, data_array, label=mod, color=color, linestyle=
            linestyle, linewidth=1.2)
333     plt.xlabel(f'Scalarappliedoninitialvalueparameter{parameter}_(-)', fontsize=12)
334     if type == 'frequency':
335         plt.ylabel('Absolutechangenaturalfrequency$_{f_n}$(%)', fontsize=12)
336         plt.ylim(-1.5, 100)
337         plt.yticks(np.arange(0, 110, 10))
338         plt.legend(loc='upperright')
339     else:
340         plt.ylabel('Changemodeshape$_{MAC}$(-)', fontsize=12)
341         plt.ylim(0.70, 1.01)
342         plt.yticks(np.arange(0.70, 1.05, 0.05))
343         plt.legend(loc='lowerright')
344     plt.gca().spines['top'].set_linewidth(0.5)
345     plt.gca().spines['top'].set_color('black')
346     plt.gca().spines['right'].set_linewidth(0.5)
347     plt.gca().spines['right'].set_color('black')
348     plt.gca().spines['bottom'].set_linewidth(0.5)
349     plt.gca().spines['bottom'].set_color('black')
350     plt.gca().spines['left'].set_linewidth(0.5)
351     plt.gca().spines['left'].set_color('black')
352     plt.xscale('log')
353     plt.gca().set_xticks([0.1, 1, 10])
354     plt.gca().set_xticklabels(['0.1', '1', '10'])
355     plt.grid(True, color='grey', linestyle=':')
356     plt.show()
357
358 %load_ext autoreload
359 %autoreload 2
360 from pyvibe.models.Timoshenko import Timo, TimoPieceWiseBeam2D
361 from pyvibe.studies.influence_parameters import change_E, change_rho, change_kG, change_kr,
    change_I, change_kt, plot_per_parameter_influence
362 import numpy as np
363 import matplotlib.pyplot as plt
364
365 %matplotlib inline
366
367 ## Parameters
368
369 # Dimension Parameters:
370 L = 160.58 #[m]
371 Lt = np.array([5.735, 11.47, 26.640, 89.91, 26.825]) #[m]
372 B = 28 #[m]
373 D = 28 #[m]
374 A = B * D #[m2]
375 I_x = np.array([619.21, 948.47, 2727.22, 2335.82, 905.52]) #[m4]
376 I_y = np.array([1536.25, 959.01, 1245.46, 1202.07, 831.57]) #[m4]
377 I_yy = np.array([1444, 1500, 1393, 1389, 1293]) #[m4]
378 I_xx = np.array([1532, 1548, 2796, 2324, 760]) #[m4]
379 H = Lt/L #[-]
380
381 # Material properties:
382 rho = np.array([1933, 495, 486, 440, 383]) #[kg/m3]
383 E = np.array([38.2e9, 38.2e9, 38.2e9, 38.2e9, 32.8e9]) #[N/m2]
384 v = 0.2 #[-]
385 G = E/(2*(1+v)) #[N/m^2]
386 k = 0.85 #[-]
387
388 # Springs:
389 kr_x = 2380e9 #[Nm/rad]
390 kr_y = 2625e9 #[Nm/rad]
391 kt_y = 19e9 #[N/m]
392 kt_x = 4e9 #[N/m]
393

```

```

394 #boundary conditions:
395 bc = ("rot+transl", "free")
396
397 # Scalar boundaries
398 amount_points = 300
399 left = 0.1
400 right = 10
401
402 ## Procedure
403
404 timoshenko_x = []
405 for ix in range(len(H)):
406     timoshenko_x.append(Timo(params={"E": E[ix], "I": I_xx[ix], "L": L, "rho": rho[ix], "A": A
407         , "kG": k * G[ix], "kr": kr_x, "kt": kt_x}, height=H[ix]))
408 timoshenko_y = []
409 for ix in range(len(H)):
410     timoshenko_y.append(Timo(params={"E": E[ix], "I": I_yy[ix], "L": L, "rho": rho[ix], "A": A
411         , "kG": k * G[ix], "kr": kr_y, "kt": kt_y}, height=H[ix]))
412 model2D = TimoPieceWiseBeam2D(beams_x=timoshenko_x, beams_y=timoshenko_y, bc = bc)
413 fn_x, fn_y, phi_x, phi_y, pts_x, pts_y = model2D.modeshapes()
414 change = np.logspace(np.log10(left), np.log10(right), amount_points)
415
416 Change_f1_x_kr, Change_f2_x_kr, Change_f3_x_kr, Change_f1_y_kr, Change_f2_y_kr,
417 Change_f3_y_kr, Change_m1_x_kr, Change_m2_x_kr, Change_m3_x_kr, Change_m1_y_kr,
418 Change_m2_y_kr, Change_m3_y_kr = change_kr(change, fn_x, fn_y, phi_x, phi_y, E, E, I_xx,
419 I_yy, L, rho, A, k, G, G, kr_x, kr_y, kt_x, kt_y, H, bc)
420
421 Change_f1_x_kt, Change_f2_x_kt, Change_f3_x_kt, Change_f1_y_kt, Change_f2_y_kt,
422 Change_f3_y_kt, Change_m1_x_kt, Change_m2_x_kt, Change_m3_x_kt, Change_m1_y_kt,
423 Change_m2_y_kt, Change_m3_y_kt = change_kt(change, fn_x, fn_y, phi_x, phi_y, E, E, I_xx,
424 I_yy, L, rho, A, k, G, G, kr_x, kr_y, kt_x, kt_y, H, bc)
425
426 Change_f1_x_I, Change_f2_x_I, Change_f3_x_I, Change_f1_y_I, Change_f2_y_I, Change_f3_y_I,
427 Change_m1_x_I, Change_m2_x_I, Change_m3_x_I, Change_m1_y_I, Change_m2_y_I, Change_m3_y_I
428 = change_I(change, fn_x, fn_y, phi_x, phi_y, E, E, I_xx, I_yy, L, rho, A, k, G, G, kr_x,
429 kr_y, kt_x, kt_y, H, bc)
430
431 Change_f1_x_E, Change_f2_x_E, Change_f3_x_E, Change_f1_y_E, Change_f2_y_E, Change_f3_y_E,
432 Change_m1_x_E, Change_m2_x_E, Change_m3_x_E, Change_m1_y_E, Change_m2_y_E, Change_m3_y_E
433 = change_E(change, fn_x, fn_y, phi_x, phi_y, E, E, I_xx, I_yy, L, rho, A, k, G, G, kr_x,
434 kr_y, kt_x, kt_y, H, bc)
435
436 Change_f1_x_kG, Change_f2_x_kG, Change_f3_x_kG, Change_f1_y_kG, Change_f2_y_kG,
437 Change_f3_y_kG, Change_m1_x_kG, Change_m2_x_kG, Change_m3_x_kG, Change_m1_y_kG,
438 Change_m2_y_kG, Change_m3_y_kG = change_kG(change, fn_x, fn_y, phi_x, phi_y, E, E, I_xx,
439 I_yy, L, rho, A, k, G, G, kr_x, kr_y, kt_x, kt_y, H, bc)
440
441 Change_f1_x_rho, Change_f2_x_rho, Change_f3_x_rho, Change_f1_y_rho, Change_f2_y_rho,
442 Change_f3_y_rho, Change_m1_x_rho, Change_m2_x_rho, Change_m3_x_rho, Change_m1_y_rho,
443 Change_m2_y_rho, Change_m3_y_rho = change_rho(change, fn_x, fn_y, phi_x, phi_y, E, E,
444 I_xx, I_yy, L, rho, A, k, G, G, kr_x, kr_y, kt_x, kt_y, H, bc)
445
446 ## Results per parameter
447
448 plot_per_parameter_influence('frequency', 'kr', 'x', change, Change_f1_x_kr, Change_f2_x_kr)
449 plot_per_parameter_influence('mode', 'kr', 'x', change, Change_m1_x_kr, Change_m2_x_kr)
450 plot_per_parameter_influence('frequency', 'kr', 'y', change, Change_f1_y_kr, Change_f2_y_kr,
451 Change_f3_y_kr)
452 plot_per_parameter_influence('mode', 'kr', 'y', change, Change_m1_y_kr, Change_m2_y_kr,
453 Change_m3_y_kr)
454
455 plot_per_parameter_influence('frequency', 'kt', 'x', change, Change_f1_x_kt, Change_f2_x_kt)
456 plot_per_parameter_influence('mode', 'kt', 'x', change, Change_m1_x_kt, Change_m2_x_kt)
457 plot_per_parameter_influence('frequency', 'kt', 'y', change, Change_f1_y_kt, Change_f2_y_kt,
458 Change_f3_y_kt)
459 plot_per_parameter_influence('mode', 'kt', 'y', change, Change_m1_y_kt, Change_m2_y_kt,
460 Change_m3_y_kt)
461
462 plot_per_parameter_influence('frequency', 'I', 'x', change, Change_f1_x_I, Change_f2_x_I)
463 plot_per_parameter_influence('mode', 'I', 'x', change, Change_m1_x_I, Change_m2_x_I)
464 plot_per_parameter_influence('frequency', 'I', 'y', change, Change_f1_y_I, Change_f2_y_I,

```

```

    Change_f3_y_I)
441 plot_per_parameter_influence('mode','I', 'y', change, Change_m1_y_I, Change_m2_y_I,
    Change_m3_y_I)
442
443 plot_per_parameter_influence('frequency','E','x', change, Change_f1_x_E, Change_f2_x_E)
444 plot_per_parameter_influence('mode','E','x', change, Change_m1_x_E, Change_m2_x_E)
445 plot_per_parameter_influence('frequency','E','y', change, Change_f1_y_E, Change_f2_y_E,
    Change_f3_y_E)
446 plot_per_parameter_influence('mode','E','y', change, Change_m1_y_E, Change_m2_y_E,
    Change_m3_y_E)
447
448 plot_per_parameter_influence('frequency','kG','x', change, Change_f1_x_kG, Change_f2_x_kG)
449 plot_per_parameter_influence('mode','kG','x', change, Change_m1_x_kG, Change_m2_x_kG)
450 plot_per_parameter_influence('frequency','kG','y', change, Change_f1_y_kG, Change_f2_y_kG,
    Change_f3_y_kG)
451 plot_per_parameter_influence('mode','kG','y', change, Change_m1_y_kG, Change_m2_y_kG,
    Change_m3_y_kG)
452
453 plot_per_parameter_influence('frequency','rho', 'x', change, Change_f1_x_rho, Change_f2_x_rho
    )
454 plot_per_parameter_influence('mode','rho', 'x', change, Change_m1_x_rho, Change_m2_x_rho)
455 plot_per_parameter_influence('frequency','rho', 'y', change, Change_f1_y_rho, Change_f2_y_rho
    , Change_f3_y_rho)
456 plot_per_parameter_influence('mode','rho', 'y', change, Change_m1_y_rho, Change_m2_y_rho,
    Change_m3_y_rho)

```

### Optimization Algorithm

```

1 %load_ext autoreload
2 %autoreload 2
3 from copy import deepcopy
4 from pyvibe.models.Timoshenko import Timo, TimoPieceWiseBeam2D, UpdatingMethod, cost_func2D,
    update_model
5 import numpy as np
6 import seaborn as sns
7 import matplotlib.pyplot as plt
8 %matplotlib inline
9 import pandas as pd
10
11 ## Parameters
12
13 # Dimension Parameters:
14 L = 153.87 #[m]
15 Lt = np.array([3.15, 22.65, 46.8, 81.27]) #[m]
16 A = np.array([1320, 1320, 1320, 825]) #[m]
17 I_yy = np.array([4312, 4312, 4744, 1402]) #[m4]
18 I_xx = np.array([34350, 34350, 35105, 10685]) #[m4]
19 H = Lt/L #[-]
20 # Material properties:
21 rho = np.array([5070, 552, 426, 371]) #[kg/m3]
22 E = np.array([11e9, 11e9, 22e9, 33e9]) #[N/m2]
23 v = 0.2 #[-]
24 G = E/(2*(1+v))#[N/m^2]
25 k = 0.85 #[-]
26
27 # Springs:
28 kr_y = 6486e9 #[Nm/rad]
29 kr_x = 50371e9 #[Nm/rad]
30 kt_x = 9.22e9 #[N/m]
31 kt_y = 9.22e9 #[N/m]
32
33 #boundary conditions:
34 bc = ("rot+transl", "free")
35 scaler = {"kr":0.4, "kt":9.0, "I":1.5, "E":7.0, "rho":2.0} #The parameters chosen are from
    the conclusion of sensitivity study 2
36 zz_trgt = np.array([0, 69, 108.60, 144.60]) #The three locations where there are
    measurements taken of the New Orleans
37 z_trgt = zz_trgt/L
38
39 ## Procedure

```

```

40
41 timoshenko_x = []
42 for ix in range(len(H)):
43     timoshenko_x.append(Timo(params={"E": E[ix], "I": I_xx[ix], "L": L, "rho": rho[ix], "A": A[
44         ix], "kG": k * G[ix], "kr": kr_x, "kt": kt_x}, height=H[ix]))
45 timoshenko_y = []
46 for ix in range(len(H)):
47     timoshenko_y.append(Timo(params={"E": E[ix], "I": I_yy[ix], "L": L, "rho": rho[ix], "A": A[
48         ix], "kG": k * G[ix], "kr": kr_y, "kt": kt_y}, height=H[ix]))
49 model = TimoPieceWiseBeam2D(beams_x=timoshenko_x, beams_y=timoshenko_y, bc = bc)
50
51 model.scale_params(scalers=scaler)
52 fn_x, fn_y, phi_x, phi_y, pts_x, pts_y = model.modeshapes(pts=z_trgt, n_modes_x = 1,
53     n_modes_y = 2)
54 y_hat = {"z": z_trgt, "fn_x": deepcopy(model.fn_x), "fn_y": deepcopy(model.fn_y), "modes_x":
55     deepcopy(model.phi_x), "modes_y": deepcopy(model.phi_y)}
56
57 parameters_to_update = {p:[1/10, 9.8] for p in list(scaler.keys())}
58 logscale = False
59
60 print(parameters_to_update)
61
62 def process_iteration(npts):
63     results = update_model(
64         model=model,
65         cost_func=cost_func2D,
66         parameters_to_update=parameters_to_update,
67         y_hat=y_hat,
68         npts=npts,
69         sim_type=UpdatingMethod.CMC,
70         verbose=False,
71         logscale=logscale)
72
73     parameters_columns = [cc for cc in results.columns if "_st" in cc or "_opt" in cc]
74     plot_cols = [a + "_opt" for a in parameters_to_update.keys()]
75     map_opt2name = {pcl: k for k, pcl in zip(parameters_to_update.keys(), plot_cols)}
76     df_opt = results.copy()
77     df_opt.dropna(inplace=True)
78
79     if logscale:
80         df_opt[parameters_columns] = df_opt[parameters_columns].apply(np.exp)
81
82     top_lowest_convergence = df_opt.sort_values(by='Convergence').head(1)
83     return (npts,
84         top_lowest_convergence["kr_opt"].iloc[0],
85         top_lowest_convergence["kt_opt"].iloc[0],
86         top_lowest_convergence["I_opt"].iloc[0],
87         top_lowest_convergence["E_opt"].iloc[0],
88         top_lowest_convergence["Convergence"].iloc[0],
89         top_lowest_convergence["rho_opt"].iloc[0])
90
91 kr = []
92 kt = []
93 I = []
94 E = []
95 rho = []
96 convergence = []
97 startpoints = []
98 npts_values = range(25, 525, 25)
99
100 for npts in npts_values:
101     result = process_iteration(npts)
102     startpoints.append(result[0])
103     kr.append(result[1])
104     kt.append(result[2])
105     I.append(result[3])
106     E.append(result[4])
107     convergence.append(result[5])
108     rho.append(result[6])
109
110 import os

```



```

    ')
173 plt.plot(dfs[0], productCMC, color='firebrick', label = 'CMC',marker='^', markersize=6,
    linewidth=0.8,markeredgewidth=0.5, markeredgewidth=0.5, markeredgewidth=0.5, markeredgewidth=0.5, markeredgewidth=0.5,
174 plt.plot(dfs[0], productCMCr, color='goldenrod', label='CMC with  $\rho$ ', marker='D',
    markersize=4, markeredgewidth=0.5, markeredgewidth=0.5, markeredgewidth=0.5, markeredgewidth=0.5, markeredgewidth=0.5,
    zorder =1)
175
176 # Customize the grid and spines
177 plt.grid(True, which='both', linestyle=':', linewidth='0.5', color='white')
178 ax = plt.gca()
179 ax.spines['top'].set_linewidth(0.5)
180 ax.spines['top'].set_color('white')
181 ax.spines['right'].set_linewidth(0.5)
182 ax.spines['right'].set_color('white')
183 ax.spines['bottom'].set_linewidth(0.5)
184 ax.spines['bottom'].set_color('black')
185 ax.spines['left'].set_linewidth(0.5)
186 ax.spines['left'].set_color('black')
187
188 # Set tick positions and limits
189 plt.xticks(np.arange(0, 501, 100), fontname='Arial')
190
191 plt.ylim(7.5,13.5)
192 plt.yticks(np.arange(7.5, 13.6, 1.5), fontname='Arial')
193 ax.yaxis.set_major_formatter(plt.FormatStrFormatter('%0.2f'))
194 # Set the y-axis height for the bottom spine
195 y_axis_height = 10.5
196 ax.spines['bottom'].set_position(('data', y_axis_height))
197
198 # Manually adjust x-axis label position
199 ax.xaxis.set_label_coords(-0.5, 0.0) # Adjust x-axis label position
200
201 # Ensure a tick mark at the end of x-axis without a corresponding value
202 xticks = np.arange(100, 501, 100)
203 plt.xlim(15, 525)
204 xticks = np.append(xticks, 525) # Add the endpoint where you want the tick without a label
205 ax.set_xticks(xticks)
206 ax.set_xticklabels([str(int(x)) if x != 525 else '' for x in xticks]) # Set labels; use ''
    for the endpoint
207
208 # Customize the ticks
209 plt.tick_params(axis='y', which='major', direction='out', length=2, width=1, colors='black')
210 plt.tick_params(axis='x', which='major', direction='out', length=2, width=1, colors='black')
211 plt.gca().yaxis.set_ticks_position('left')
212 plt.gca().xaxis.set_ticks_position('bottom')# Ensure ticks are on the left side of the plot
213 plt.tick_params(axis='y', which='major', direction='out', length=2, width=1, colors='black')
214 plt.tick_params(axis='x', which='major', direction='inout', length=4, width=1, colors='black')
    )
215 # Set labels and legend
216 plt.ylabel('Scalar(-)', fontsize=10, fontname='Arial')
217 plt.xlabel('Amount of candidate solutions(-)', fontsize=10, fontname='Arial')
218 plt.legend(prop={'family': 'Arial'}, loc='upper right')
219
220 # Show the plot
221 plt.show()
222
223 ## Results
224
225 import matplotlib.pyplot as plt
226 import numpy as np
227
228 plt.figure(figsize=(7, 5))
229
230 # Plot the data
231 plt.plot(dfs[0], dfs[1], color='lightsteelblue', label='PSO', marker='s', markersize=4.5,
    markeredgewidth=0.5, markeredgewidth=0.5, markeredgewidth=0.5, markeredgewidth=0.5, markeredgewidth=0.5,
232 #plt.plot(dfs[0], dfs[11], color='steelblue', label='GA', marker='o', markersize=5, linewidth
    =0.8, markeredgewidth=0.5, markeredgewidth=0.5, markeredgewidth=0.5, markeredgewidth=0.5, markeredgewidth=0.5,
233 plt.plot(dfs[0], dfs[6], color='black', label='DE', marker='v', markersize=6, linewidth=0.8,
    markeredgewidth=0.5, markeredgewidth=0.5, markeredgewidth=0.5, markeredgewidth=0.5, markeredgewidth=0.5,
    markerfacecolor='white', linestyle='-')
234 plt.plot(dfs[0], dfs[16], color='firebrick', label='CMC', marker='^', markersize=6, linewidth

```

```

    =0.8, markeredgewidth=0.5, markeredgewidth='black', zorder=3)
235 plt.plot(dfs[0], dfs[21], color='goldenrod', label='CMCwith $\rho$ ', marker='D', markersize
    =4, markeredgewidth=0.5, markeredgewidth='black', linewidth=0.8, linestyle='-')
236
237 # Customize the grid and spines
238 plt.grid(True, which='both', linestyle=':', linewidth='0.5', color='white')
239 ax = plt.gca()
240 ax.spines['top'].set_linewidth(0.5)
241 ax.spines['top'].set_color('white')
242 ax.spines['right'].set_linewidth(0.5)
243 ax.spines['right'].set_color('white')
244 ax.spines['bottom'].set_linewidth(0.5)
245 ax.spines['bottom'].set_color('black')
246 ax.spines['left'].set_linewidth(0.5)
247 ax.spines['left'].set_color('black')
248 ax.yaxis.set_major_formatter(plt.FormatStrFormatter('%0.2f'))
249 # Set tick positions and limits
250 plt.xticks(np.arange(0, 501, 100), fontname='Arial')
251
252 plt.yticks(np.arange(0, 6.01, 1.5), fontname='Arial')
253 plt.ylim(0, 6.0)
254
255 # Set the y-axis height for the bottom spine
256 y_axis_height = 1.5
257 ax.spines['bottom'].set_position(('data', y_axis_height))
258
259 # Manually adjust x-axis label position
260 ax.xaxis.set_label_coords(-0.5, 0.01) # Adjust x-axis label position
261
262 # Ensure a tick mark at the end of x-axis without a corresponding value
263 xticks = np.arange(100, 501, 100)
264 plt.xlim(15, 525)
265 xticks = np.append(xticks, 525) # Add the endpoint where you want the tick without a label
266 ax.set_xticks(xticks)
267 ax.set_xticklabels([str(int(x)) if x != 525 else '' for x in xticks]) # Set labels; use ''
    for the endpoint
268
269 # Customize the ticks
270 plt.tick_params(axis='y', which='major', direction='out', length=2, width=1, colors='black')
271 plt.tick_params(axis='x', which='major', direction='out', length=2, width=1, colors='black')
272 plt.gca().yaxis.set_ticks_position('left')
273 plt.gca().xaxis.set_ticks_position('bottom') # Ensure ticks are on the left side of the plot
274 plt.tick_params(axis='y', which='major', direction='out', length=2, width=1, colors='black')
275 plt.tick_params(axis='x', which='major', direction='inout', length=4, width=1, colors='black')
276
277 # Set labels and legend
278 plt.ylabel('Scalar(-)', fontsize=10, fontname='Arial')
279 plt.xlabel('Amount of candidate solutions(-)', fontsize=10, fontname='Arial')
280 plt.legend(prop={'family': 'Arial'}, loc='upper right')
281
282 # Show the plot
283 plt.show()
284
285 import matplotlib.pyplot as plt
286 import numpy as np
287
288 plt.figure(figsize=(7, 5))
289
290 # Plot the data
291 plt.plot(dfs[0], dfs[2], color='lightsteelblue', label='PSO', marker='s', markersize=4.5,
    markeredgewidth=0.5, markeredgewidth='black', linewidth=1, linestyle='-')
292 #plt.plot(dfs[0], dfs[12], color='steelblue', label='GA', marker='o', markersize=5, linewidth
    =0.8, markeredgewidth=0.5, markeredgewidth='black', linestyle='-')
293 plt.plot(dfs[0], dfs[7], color='black', label='DE', marker='v', markersize=6, linewidth=0.8,
    markeredgewidth=0.5, markeredgewidth='black', markerfacecolor='white', linestyle='-')
294 plt.plot(dfs[0], dfs[17], color='firebrick', label='CMC', marker='^', markersize=6, linewidth
    =0.8, markeredgewidth=0.5, markeredgewidth='black', zorder=3)
295 plt.plot(dfs[0], dfs[22], color='goldenrod', label='CMCwith $\rho$ ', marker='D', markersize
    =4, markeredgewidth=0.5, markeredgewidth='black', linewidth=0.8, linestyle='-')
296
297 # Customize the grid and spines

```



```

297 plt.grid(True, which='both', linestyle=':', linewidth='0.5', color='white')
298 ax = plt.gca()
299 ax.spines['top'].set_linewidth(0.5)
300 ax.spines['top'].set_color('white')
301 ax.spines['right'].set_linewidth(0.5)
302 ax.spines['right'].set_color('white')
303 ax.spines['bottom'].set_linewidth(0.5)
304 ax.spines['bottom'].set_color('black')
305 ax.spines['left'].set_linewidth(0.5)
306 ax.spines['left'].set_color('black')
307 ax.yaxis.set_major_formatter(plt.FormatStrFormatter('%.2f'))
308 # Set tick positions and limits
309 plt.xticks(np.arange(0, 501, 100), fontname='Arial')
310
311 plt.yticks(np.arange(1, 11.01, 2), fontname='Arial')
312 plt.ylim(1, 11.0)
313
314 # Set the y-axis height for the bottom spine
315 y_axis_height = 7.0
316 ax.spines['bottom'].set_position(('data', y_axis_height))
317
318 # Manually adjust x-axis label position
319 ax.xaxis.set_label_coords(-0.5, 0.01) # Adjust x-axis label position
320
321 # Ensure a tick mark at the end of x-axis without a corresponding value
322 xticks = np.arange(100, 501, 100)
323 plt.xlim(15, 525)
324 xticks = np.append(xticks, 525) # Add the endpoint where you want the tick without a label
325 ax.set_xticks(xticks)
326 ax.set_xticklabels([str(int(x)) if x != 525 else '' for x in xticks]) # Set labels; use ''
    for the endpoint
327
328 # Customize the ticks
329 plt.tick_params(axis='y', which='major', direction='out', length=2, width=1, colors='black')
330 plt.tick_params(axis='x', which='major', direction='out', length=2, width=1, colors='black')
331 plt.gca().yaxis.set_ticks_position('left')
332 plt.gca().xaxis.set_ticks_position('bottom') # Ensure ticks are on the left side of the plot
333 plt.tick_params(axis='y', which='major', direction='out', length=2, width=1, colors='black')
334 plt.tick_params(axis='x', which='major', direction='out', length=2, width=1, colors='black')
335 # Set labels and legend
336 plt.ylabel('Scalar(-)', fontsize=10, fontname='Arial')
337 plt.xlabel('Amount of candidate solutions(-)', fontsize=10, fontname='Arial')
338 plt.legend(prop={'family': 'Arial'}, loc='upper right')
339
340 # Show the plot
341 plt.show()
342
343 plt.figure(figsize=(7, 5))
344
345 plt.plot(dfs[0], dfs[3], color='lightsteelblue', label = 'PSO', marker='s', markersize=4.5,
    markeredgewidth=0.5, markeredgecolor='black', linewidth=1, linestyle='-')
346 #plt.plot(dfs[0], productGA, color='steelblue', label = 'GA', marker='o', markersize=5,
    linewidth=0.8, markeredgewidth=0.5, markeredgecolor='black', linestyle='-')
347 plt.plot(dfs[0], dfs[8], color='black', label = 'DE', marker='v', markersize=6, linewidth=0.8,
    markeredgewidth=0.5, markeredgecolor='black', markerfacecolor='white', linestyle='-')
348 plt.plot(dfs[0], dfs[18], color='firebrick', label = 'CMC', marker='^', markersize=6,
    linewidth=0.8, markeredgewidth=0.5, markeredgecolor='black', zorder=3)
349 plt.plot(dfs[0], dfs[23], color='goldenrod', label='CMC with  $\rho$ ', marker='D', markersize
    =4, markeredgewidth=0.5, markeredgecolor='black', linewidth=0.8, linestyle='-', zorder =1)
350
351 # Customize the grid and spines
352 plt.grid(True, which='both', linestyle=':', linewidth='0.5', color='white')
353 ax = plt.gca()
354 ax.spines['top'].set_linewidth(0.5)
355 ax.spines['top'].set_color('white')
356 ax.spines['right'].set_linewidth(0.5)
357 ax.spines['right'].set_color('white')
358 ax.spines['bottom'].set_linewidth(0.5)
359 ax.spines['bottom'].set_color('black')
360 ax.spines['left'].set_linewidth(0.5)
361 ax.spines['left'].set_color('black')

```



```

362
363 # Set tick positions and limits
364 plt.xticks(np.arange(0, 501, 100), fontname='Arial')
365
366 plt.ylim(0.32,0.42)
367 plt.yticks(np.arange(0.32, 0.421, 0.02), fontname='Arial')
368
369 # Set the y-axis height for the bottom spine
370 y_axis_height = 0.4
371 ax.spines['bottom'].set_position(('data', y_axis_height))
372
373 # Manually adjust x-axis label position
374 ax.xaxis.set_label_coords(-0.5, 0.01) # Adjust x-axis label position
375
376 # Ensure a tick mark at the end of x-axis without a corresponding value
377 xticks = np.arange(100, 501, 100)
378 plt.xlim(15, 525)
379 xticks = np.append(xticks, 525) # Add the endpoint where you want the tick without a label
380 ax.set_xticks(xticks)
381 ax.set_xticklabels([str(int(x)) if x != 525 else '' for x in xticks]) # Set labels; use ''
    for the endpoint
382
383 # Customize the ticks
384 plt.tick_params(axis='y', which='major', direction='out', length=2, width=1, colors='black')
385 plt.tick_params(axis='x', which='major', direction='out', length=2, width=1, colors='black')
386 plt.gca().yaxis.set_ticks_position('left')
387 plt.gca().xaxis.set_ticks_position('bottom') # Ensure ticks are on the left side of the plot
388 plt.tick_params(axis='y', which='major', direction='out', length=2, width=1, colors='black')
389 plt.tick_params(axis='x', which='major', direction='inout', length=4, width=1, colors='black'
    )
390
391 # Set labels and legend
392 plt.ylabel('Scalar(-)', fontsize=10, fontname='Arial')
393 plt.xlabel('Amount of candidate solutions(-)', fontsize=10, fontname='Arial')
394 plt.legend(prop={'family': 'Arial'}, loc='lower right')
395
396 # Show the plot
397 plt.show()
398
399 plt.figure(figsize=(7, 5))
400
401 plt.plot(dfs[0], dfs[4], color='lightsteelblue', label = 'PSO', marker='s', markersize=4.5,
    markeredgewidth=0.5, markeredgewidth=0.5, markeredgewidth=0.5, linewidth=1, linestyle='-')
402 #plt.plot(dfs[0], productGA, color='steelblue', label = 'GA', marker='o', markersize=5,
    linewidth=0.8, markeredgewidth=0.5, markeredgewidth=0.5, markeredgewidth=0.5, linewidth=0.8,
403 plt.plot(dfs[0], dfs[9], color='black', label = 'DE', marker='v', markersize=6, linewidth=0.8,
    markeredgewidth=0.5, markeredgewidth=0.5, markeredgewidth=0.5, linewidth=0.8,
404 plt.plot(dfs[0], dfs[19], color='firebrick', label = 'CMC', marker='^', markersize=6,
    linewidth=0.8, markeredgewidth=0.5, markeredgewidth=0.5, markeredgewidth=0.5, linewidth=0.8,
405 plt.plot(dfs[0], dfs[24], color='goldenrod', label='CMC with  $\rho$ ', marker='D', markersize
    =4, markeredgewidth=0.5, markeredgewidth=0.5, markeredgewidth=0.5, linewidth=0.8, linestyle='-', zorder =1)
406
407 # Customize the grid and spines
408 plt.grid(True, which='both', linestyle=':', linewidth='0.5', color='white')
409 ax = plt.gca()
410 ax.spines['top'].set_linewidth(0.5)
411 ax.spines['top'].set_color('white')
412 ax.spines['right'].set_linewidth(0.5)
413 ax.spines['right'].set_color('white')
414 ax.spines['bottom'].set_linewidth(0.5)
415 ax.spines['bottom'].set_color('black')
416 ax.spines['left'].set_linewidth(0.5)
417 ax.spines['left'].set_color('black')
418
419 # Set tick positions and limits
420 plt.xticks(np.arange(0, 501, 100), fontname='Arial')
421
422 plt.ylim(7.5,9.9)
423 plt.yticks(np.arange(7.5, 10.01, 0.5), fontname='Arial')
424
425 # Set the y-axis height for the bottom spine
426 y_axis_height = 9.0

```

```

426 ax.spines['bottom'].set_position(('data', y_axis_height))
427 ax.yaxis.set_major_formatter(plt.FormatStrFormatter('%.2f'))
428 # Manually adjust x-axis label position
429 ax.xaxis.set_label_coords(-0.5, 0.01) # Adjust x-axis label position
430
431 # Ensure a tick mark at the end of x-axis without a corresponding value
432 xticks = np.arange(100, 501, 100)
433 plt.xlim(15, 525)
434 xticks = np.append(xticks, 525) # Add the endpoint where you want the tick without a label
435 ax.set_xticks(xticks)
436 ax.set_xticklabels([str(int(x)) if x != 525 else '' for x in xticks]) # Set labels; use ''
    for the endpoint
437
438 # Customize the ticks
439 plt.tick_params(axis='y', which='major', direction='out', length=2, width=1, colors='black')
440 plt.tick_params(axis='x', which='major', direction='out', length=2, width=1, colors='black')
441 plt.gca().yaxis.set_ticks_position('left')
442 plt.gca().xaxis.set_ticks_position('bottom') # Ensure ticks are on the left side of the plot
443 plt.tick_params(axis='y', which='major', direction='out', length=2, width=1, colors='black')
444 plt.tick_params(axis='x', which='major', direction='inout', length=4, width=1, colors='black'
    )
445 # Set labels and legend
446 plt.ylabel('Scalar(-)', fontsize=10, fontname='Arial')
447 plt.xlabel('Amount of candidate solutions(-)', fontsize=10, fontname='Arial')
448 plt.legend(prop={'family': 'Arial'}, loc='lower right')
449
450 # Show the plot
451 plt.show()

```

Amount of Measured Modal Properties

```

1 %load_ext autoreload
2 %autoreload 2
3 from copy import deepcopy
4 from pyvibe.models.Timoshenko import Timo, TimoPieceWiseBeam2D, UpdatingMethod, cost_func2D,
    update_model
5 import numpy as np
6 import seaborn as sns
7 import matplotlib.pyplot as plt
8 %matplotlib inline
9 import pandas as pd
10
11 ## Parameters
12
13 # Dimension Parameters:
14 L = 153.87 #[m]
15 Lt = np.array([3.15, 22.65, 46.8, 81.27]) #[m]
16 A = np.array([1320, 1320, 1320, 825]) #[m]
17 I_yy = np.array([4312, 4312, 4744, 1402]) #[m4]
18 I_xx = np.array([34350, 34350, 35105, 10685]) #[m4]
19 H = Lt/L #[-]
20 # Material properties:
21 rho = np.array([5070, 552, 426, 371]) #[kg/m3]
22 E = np.array([11e9, 11e9, 22e9, 33e9]) #[N/m2]
23 v = 0.2 #[-]
24 G = E/(2*(1+v))#[N/m^2]
25 k = 0.85 #[-]
26
27 # Springs:
28 kr_y = 6486e9 #[Nm/rad]
29 kr_x = 50371e9 #[Nm/rad]
30 kt_x = 9.22e9 #[N/m]
31 kt_y = 9.22e9 #[N/m]
32
33 #boundary conditions:
34 bc = ("rot+transl", "free")
35 scaler = {"kr":0.4, "kt":9.0, "I":1.5, "E":7.0}
36 zz_trgt = np.array([0, 69, 108.60, 144.60])
37 z_trgt = zz_trgt/L
38

```

```

39 ## Procedure
40
41 timoshenko_x = []
42 for ix in range(len(H)):
43     timoshenko_x.append(Timo(params={"E": E[ix], "I": I_xx[ix], "L": L, "rho": rho[ix], "A": A[
44         ix], "kG": k * G[ix], "kr": kr_x, "kt": kt_x}, height=H[ix]))
45 timoshenko_y = []
46 for ix in range(len(H)):
47     timoshenko_y.append(Timo(params={"E": E[ix], "I": I_yy[ix], "L": L, "rho": rho[ix], "A": A[
48         ix], "kG": k * G[ix], "kr": kr_y, "kt": kt_y}, height=H[ix]))
49 model = TimoPieceWiseBeam2D(beams_x=timoshenko_x, beams_y=timoshenko_y, bc = bc)
50
51 model.scale_params(scalers=scaler)
52 fn_x, fn_y, phi_x, phi_y, pts_x, pts_y = model.modeshapes(pts=z_trgt, n_modes_x = 1,
53     n_modes_y = 1)
54 y_hat = {"z": z_trgt, "fn_x": deepcopy(model.fn_x), "fn_y": deepcopy(model.fn_y), "modes_x":
55     deepcopy(model.phi_x), "modes_y": deepcopy(model.phi_y)}
56
57 parameters_to_update = {p:[1/10, 9.8] for p in list(scaler.keys())}
58 logscale = False
59
60 print(parameters_to_update)
61
62 def process_iteration(npts):
63     results = update_model(
64         model=model,
65         cost_func=cost_func2D,
66         parameters_to_update=parameters_to_update,
67         y_hat=y_hat,
68         npts=npts,
69         sim_type=UpdatingMethod.CMC,
70         verbose=False,
71         logscale=logscale)
72
73     parameters_columns = [cc for cc in results.columns if "_st" in cc or "_opt" in cc]
74     plot_cols = [a + "_opt" for a in parameters_to_update.keys()]
75     map_opt2name = {pcl: k for k, pcl in zip(parameters_to_update.keys(), plot_cols)}
76     df_opt = results.copy()
77     df_opt.dropna(inplace=True)
78
79     if logscale:
80         df_opt[parameters_columns] = df_opt[parameters_columns].apply(np.exp)
81
82     top_lowest_convergence = df_opt.sort_values(by='Convergence').head(1)
83     return (npts,
84         top_lowest_convergence["kr_opt"].iloc[0],
85         top_lowest_convergence["kt_opt"].iloc[0],
86         top_lowest_convergence["I_opt"].iloc[0],
87         top_lowest_convergence["E_opt"].iloc[0],
88         top_lowest_convergence["Convergence"].iloc[0])
89
90 kr = []
91 kt = []
92 I = []
93 E = []
94 convergence = []
95 startpoints = []
96 npts_values = range(25, 525, 25)
97
98 for npts in npts_values:
99     result = process_iteration(npts)
100     startpoints.append(result[0])
101     kr.append(result[1])
102     kt.append(result[2])
103     I.append(result[3])
104     E.append(result[4])
105     convergence.append(result[5])
106
107 import os
108 import pandas as pd
109 directory = r'C:\Users\ritfeldis\OneDrive\TNO\Documents\Master_thesis\Isa_Ritfeld(4892534)

```

```

166 \5. Python\5.3\Results\DelfstePoort'
167 os.makedirs(directory, exist_ok=True)
168 file_names = [
169     'CMC_1_I.csv',
170     'CMC_1_E.csv',
171     'CMC_1_kr.csv',
172     'CMC_1_kt.csv',
173     'CMC_1_convergence.csv',
174     'CMC_1_startpoints.csv'
175 ]
176 data_lists = [I, E, kr, kt, convergence, startpoints]
177 parameter_names = ['I', 'E', 'kr', 'kt', 'convergence', 'startpoints']
178
179 for file_name, data, parameter in zip(file_names, data_lists, parameter_names):
180     df = pd.DataFrame({f'value_of_{parameter}': data})
181     file_path = os.path.join(directory, file_name)
182     df.to_csv(file_path, index=False)
183
184 %load_ext autoreload
185 %autoreload 2
186 from copy import deepcopy
187 from pyvibe.models.Timoshenko import Timo, TimoPieceWiseBeam2D, UpdatingMethod, cost_func2D,
188     update_model
189 import numpy as np
190 import seaborn as sns
191 import matplotlib.pyplot as plt
192 import pandas as pd
193 import os
194 %matplotlib inline
195
196 ## Parameters
197
198 directory = r'C:\Users\ritfeldis\OneDrive\TNO\Documents\Master_thesis_Isa_Ritfeld(4892534)
199 \5. Python\5.3\Results\DelfstePoort'
200 file_names = [
201     'combined_CMC_1_startpoints.csv', #0
202     'combined_CMC_1_I.csv', #1
203     'combined_CMC_1_E.csv', #2
204     'combined_CMC_1_kr.csv', #3
205     'combined_CMC_1_kt.csv', #4
206     'combined_CMC_1_convergence.csv', #5
207     'combined_CMC_3_I.csv', #6
208     'combined_CMC_3_E.csv', #7
209     'combined_CMC_3_kr.csv', #8
210     'combined_CMC_3_kt.csv', #9
211     'combined_CMC_3_convergence.csv', #10
212 ]
213 dfs = []
214 for file_name in file_names:
215     file_path = os.path.join(directory, file_name)
216     df = pd.read_csv(file_path)
217     dfs.append(df)
218
219 productCMC1 = dfs[1]['value_of_I'] * dfs[2]['value_of_E']
220 productCMC3 = dfs[6]['value_of_I'] * dfs[7]['value_of_E']
221
222 plt.figure(figsize=(7, 5))
223
224 plt.plot(dfs[0], productCMC1, color='lightsteelblue', label = '1 in x and 1 in y', marker='s',
225     markersize=4.5, markeredgewidth=0.5, markeredgewidth=0.5, markeredgewidth=0.5, markeredgewidth=0.5,
226     markeredgewidth=0.5, markeredgewidth=0.5, markeredgewidth=0.5, markeredgewidth=0.5,
227     markeredgewidth=0.5, markeredgewidth=0.5, markeredgewidth=0.5, markeredgewidth=0.5,
228     markeredgewidth=0.5, markeredgewidth=0.5, markeredgewidth=0.5, markeredgewidth=0.5,
229     markeredgewidth=0.5, markeredgewidth=0.5, markeredgewidth=0.5, markeredgewidth=0.5,
230     markeredgewidth=0.5, markeredgewidth=0.5, markeredgewidth=0.5, markeredgewidth=0.5,
231     markeredgewidth=0.5, markeredgewidth=0.5, markeredgewidth=0.5, markeredgewidth=0.5,
232     markeredgewidth=0.5, markeredgewidth=0.5, markeredgewidth=0.5, markeredgewidth=0.5,
233     markeredgewidth=0.5, markeredgewidth=0.5, markeredgewidth=0.5, markeredgewidth=0.5,
234     markeredgewidth=0.5, markeredgewidth=0.5, markeredgewidth=0.5, markeredgewidth=0.5,
235     markeredgewidth=0.5, markeredgewidth=0.5, markeredgewidth=0.5, markeredgewidth=0.5,
236     markeredgewidth=0.5, markeredgewidth=0.5, markeredgewidth=0.5, markeredgewidth=0.5,
237     markeredgewidth=0.5, markeredgewidth=0.5, markeredgewidth=0.5, markeredgewidth=0.5,
238     markeredgewidth=0.5, markeredgewidth=0.5, markeredgewidth=0.5, markeredgewidth=0.5,
239     markeredgewidth=0.5, markeredgewidth=0.5, markeredgewidth=0.5, markeredgewidth=0.5,
240     markeredgewidth=0.5, markeredgewidth=0.5, markeredgewidth=0.5, markeredgewidth=0.5,
241     markeredgewidth=0.5, markeredgewidth=0.5, markeredgewidth=0.5, markeredgewidth=0.5,
242     markeredgewidth=0.5, markeredgewidth=0.5, markeredgewidth=0.5, markeredgewidth=0.5,
243     markeredgewidth=0.5, markeredgewidth=0.5, markeredgewidth=0.5, markeredgewidth=0.5,
244     markeredgewidth=0.5, markeredgewidth=0.5, markeredgewidth=0.5, markeredgewidth=0.5,
245     markeredgewidth=0.5, markeredgewidth=0.5, markeredgewidth=0.5, markeredgewidth=0.5,
246     markeredgewidth=0.5, markeredgewidth=0.5, markeredgewidth=0.5, markeredgewidth=0.5,
247     markeredgewidth=0.5, markeredgewidth=0.5, markeredgewidth=0.5, markeredgewidth=0.5,
248     markeredgewidth=0.5, markeredgewidth=0.5, markeredgewidth=0.5, markeredgewidth=0.5,
249     markeredgewidth=0.5, markeredgewidth=0.5, markeredgewidth=0.5, markeredgewidth=0.5,
250     markeredgewidth=0.5, markeredgewidth=0.5, markeredgewidth=0.5, markeredgewidth=0.5,
251     markeredgewidth=0.5, markeredgewidth=0.5, markeredgewidth=0.5, markeredgewidth=0.5,
252     markeredgewidth=0.5, markeredgewidth=0.5, markeredgewidth=0.5, markeredgewidth=0.5,
253     markeredgewidth=0.5, markeredgewidth=0.5, markeredgewidth=0.5, markeredgewidth=0.5,
254     markeredgewidth=0.5, markeredgewidth=0.5, markeredgewidth=0.5, markeredgewidth=0.5,
255     markeredgewidth=0.5, markeredgewidth=0.5, markeredgewidth=0.5, markeredgewidth=0.5,
256     markeredgewidth=0.5, markeredgewidth=0.5, markeredgewidth=0.5, markeredgewidth=0.5,
257     markeredgewidth=0.5, markeredgewidth=0.5, markeredgewidth=0.5, markeredgewidth=0.5,
258     markeredgewidth=0.5, markeredgewidth=0.5, markeredgewidth=0.5, markeredgewidth=0.5,
259     markeredgewidth=0.5, markeredgewidth=0.5, markeredgewidth=0.5, markeredgewidth=0.5,
260     markeredgewidth=0.5, markeredgewidth=0.5, markeredgewidth=0.5, markeredgewidth=0.5,
261     markeredgewidth=0.5, markeredgewidth=0.5, markeredgewidth=0.5, markeredgewidth=0.5,
262     markeredgewidth=0.5, markeredgewidth=0.5, markeredgewidth=0.5, markeredgewidth=0.5,
263     markeredgewidth=0.5, markeredgewidth=0.5, markeredgewidth=0.5, markeredgewidth=0.5,
264     markeredgewidth=0.5, markeredgewidth=0.5, markeredgewidth=0.5, markeredgewidth=0.5,
265     markeredgewidth=0.5, markeredgewidth=0.5, markeredgewidth=0.5, markeredgewidth=0.5,
266     markeredgewidth=0.5, markeredgewidth=0.5, markeredgewidth=0.5, markeredgewidth=0.5,
267     markeredgewidth=0.5, markeredgewidth=0.5, markeredgewidth=0.5, markeredgewidth=0.5,
268     markeredgewidth=0.5, markeredgewidth=0.5, markeredgewidth=0.5, markeredgewidth=0.5,
269     markeredgewidth=0.5, markeredgewidth=0.5, markeredgewidth=0.5, markeredgewidth=0.5,
270     markeredgewidth=0.5, markeredgewidth=0.5, markeredgewidth=0.5, markeredgewidth=0.5,
271     markeredgewidth=0.5, markeredgewidth=0.5, markeredgewidth=0.5, markeredgewidth=0.5,
272     markeredgewidth=0.5, markeredgewidth=0.5, markeredgewidth=0.5, markeredgewidth=0.5,
273     markeredgewidth=0.5, markeredgewidth=0.5, markeredgewidth=0.5, markeredgewidth=0.5,
274     markeredgewidth=0.5, markeredgewidth=0.5, markeredgewidth=0.5, markeredgewidth=0.5,
275     markeredgewidth=0.5, markeredgewidth=0.5, markeredgewidth=0.5, markeredgewidth=0.5,
276     markeredgewidth=0.5, markeredgewidth=0.5, markeredgewidth=0.5, markeredgewidth=0.5,
277     markeredgewidth=0.5, markeredgewidth=0.5, markeredgewidth=0.5, markeredgewidth=0.5,
278     markeredgewidth=0.5, markeredgewidth=0.5, markeredgewidth=0.5, markeredgewidth=0.5,
279     markeredgewidth=0.5, markeredgewidth=0.5, markeredgewidth=0.5, markeredgewidth=0.5,
280     markeredgewidth=0.5, markeredgewidth=0.5, markeredgewidth=0.5, markeredgewidth=0.5,
281     markeredgewidth=0.5, markeredgewidth=0.5, markeredgewidth=0.5, markeredgewidth=0.5,
282     markeredgewidth=0.5, markeredgewidth=0.5, markeredgewidth=0.5, markeredgewidth=0.5,
283     markeredgewidth=0.5, markeredgewidth=0.5, markeredgewidth=0.5, markeredgewidth=0.5,
284     markeredgewidth=0.5, markeredgewidth=0.5, markeredgewidth=0.5, markeredgewidth=0.5,
285     markeredgewidth=0.5, markeredgewidth=0.5, markeredgewidth=0.5, markeredgewidth=0.5,
286     markeredgewidth=0.5, markeredgewidth=0.5, markeredgewidth=0.5, markeredgewidth=0.5,
287     markeredgewidth=0.5, markeredgewidth=0.5, markeredgewidth=0.5, markeredgewidth=0.5,
288     markeredgewidth=0.5, markeredgewidth=0.5, markeredgewidth=0.5, markeredgewidth=0.5,
289     markeredgewidth=0.5, markeredgewidth=0.5, markeredgewidth=0.5, markeredgewidth=0.5,
290     markeredgewidth=0.5, markeredgewidth=0.5, markeredgewidth=0.5, markeredgewidth=0.5,
291     markeredgewidth=0.5, markeredgewidth=0.5, markeredgewidth=0.5, markeredgewidth=0.5,
292     markeredgewidth=0.5, markeredgewidth=0.5, markeredgewidth=0.5, markeredgewidth=0.5,
293     markeredgewidth=0.5, markeredgewidth=0.5, markeredgewidth=0.5, markeredgewidth=0.5,
294     markeredgewidth=0.5, markeredgewidth=0.5, markeredgewidth=0.5, markeredgewidth=0.5,
295     markeredgewidth=0.5, markeredgewidth=0.5, markeredgewidth=0.5, markeredgewidth=0.5,
296     markeredgewidth=0.5, markeredgewidth=0.5, markeredgewidth=0.5, markeredgewidth=0.5,
297     markeredgewidth=0.5, markeredgewidth=0.5, markeredgewidth=0.5, markeredgewidth=0.5,
298     markeredgewidth=0.5, markeredgewidth=0.5, markeredgewidth=0.5, markeredgewidth=0.5,
299     markeredgewidth=0.5, markeredgewidth=0.5, markeredgewidth=0.5, markeredgewidth=0.5,
300     markeredgewidth=0.5, markeredgewidth=0.5, markeredgewidth=0.5, markeredgewidth=0.5,
301     markeredgewidth=0.5, markeredgewidth=0.5, markeredgewidth=0.5, markeredgewidth=0.5,
302     markeredgewidth=0.5, markeredgewidth=0.5, markeredgewidth=0.5, markeredgewidth=0.5,
303     markeredgewidth=0.5, markeredgewidth=0.5, markeredgewidth=0.5, markeredgewidth=0.5,
304     markeredgewidth=0.5, markeredgewidth=0.5, markeredgewidth=0.5, markeredgewidth=0.5,
```

```

167 # Customize the grid and spines
168 plt.grid(True, which='both', linestyle=':', linewidth='0.5', color='white')
169 ax = plt.gca()
170 ax.spines['top'].set_linewidth(0.5)
171 ax.spines['top'].set_color('white')
172 ax.spines['right'].set_linewidth(0.5)
173 ax.spines['right'].set_color('white')
174 ax.spines['bottom'].set_linewidth(0.5)
175 ax.spines['bottom'].set_color('black')
176 ax.spines['left'].set_linewidth(0.5)
177 ax.spines['left'].set_color('black')
178
179 # Set tick positions and limits
180 plt.xticks(np.arange(0, 501, 100), fontname='Arial')
181
182 plt.ylim(9.80, 10.6)
183 plt.yticks(np.arange(4.5, 19.6, 3), fontname='Arial')
184 ax.yaxis.set_major_formatter(plt.FormatStrFormatter('% .2f'))
185 # Set the y-axis height for the bottom spine
186 y_axis_height = 10.5
187 ax.spines['bottom'].set_position(('data', y_axis_height))
188
189 # Manually adjust x-axis label position
190 ax.xaxis.set_label_coords(-0.5, 0.0) # Adjust x-axis label position
191
192 # Ensure a tick mark at the end of x-axis without a corresponding value
193 xticks = np.arange(100, 501, 100)
194 plt.xlim(15, 525)
195 xticks = np.append(xticks, 525) # Add the endpoint where you want the tick without a label
196 ax.set_xticks(xticks)
197 ax.set_xticklabels([str(int(x)) if x != 525 else '' for x in xticks]) # Set labels; use ''
    for the endpoint
198
199 # Customize the ticks
200 plt.tick_params(axis='y', which='major', direction='out', length=2, width=1, colors='black')
201 plt.tick_params(axis='x', which='major', direction='out', length=2, width=1, colors='black')
202 plt.gca().yaxis.set_ticks_position('left')
203 plt.gca().xaxis.set_ticks_position('bottom') # Ensure ticks are on the left side of the plot
204 plt.tick_params(axis='y', which='major', direction='out', length=2, width=1, colors='black')
205 plt.tick_params(axis='x', which='major', direction='inout', length=4, width=1, colors='black'
    )
206
207 # Set labels and legend
208 plt.ylabel('Scalar(-)', fontsize=10, fontname='Arial')
209 plt.xlabel('Amount of candidate solutions(-)', fontsize=10, fontname='Arial')
210 plt.legend(prop={'family': 'Arial'}, loc='lower right')
211
212 # Show the plot
213 plt.show()
214
215 plt.figure(figsize=(7, 5))
216
217 plt.plot(dfs[0], dfs[1], color='lightsteelblue', label = '1 in x and 1 in y', marker='s',
    markersize=4.5, markeredgewidth=0.5, markeredgecolor='black', linewidth=1, linestyle='-')
218 #plt.plot(dfs[0], productGA, color='steelblue', label = 'GA', marker='o', markersize=5,
    linewidth=0.8, markeredgewidth=0.5, markeredgecolor='black', linestyle='-')
219 #plt.plot(dfs[0], dfs[6], color='black', label = '2 in x and 2 in y', marker='v', markersize
    =6, linewidth=0.8, markeredgewidth=0.5, markeredgecolor='black', markerfacecolor='white',
    linestyle='-')
220 plt.plot(dfs[0], dfs[6], color='firebrick', label = '1 in x and 2 in y', marker='^',
    markersize=6, linewidth=0.8, markeredgewidth=0.5, markeredgecolor='black', zorder=3)
221 #plt.plot(dfs[0], productCMC, color='goldenrod', label='CMC with $\rho$', marker='D',
    markersize=4, markeredgewidth=0.5, markeredgecolor='black', linewidth=0.8, linestyle='-',
    zorder = 1)
222
223 # Customize the grid and spines
224 plt.grid(True, which='both', linestyle=':', linewidth='0.5', color='white')
225 ax = plt.gca()
226 ax.spines['top'].set_linewidth(0.5)
227 ax.spines['top'].set_color('white')
228 ax.spines['right'].set_linewidth(0.5)
229 ax.spines['right'].set_color('white')
230 ax.spines['bottom'].set_linewidth(0.5)

```

```

229 ax.spines['bottom'].set_color('black')
230 ax.spines['left'].set_linewidth(0.5)
231 ax.spines['left'].set_color('black')
232 ax.yaxis.set_major_formatter(plt.FormatStrFormatter('% .2f'))
233 # Set tick positions and limits
234 plt.xticks(np.arange(0, 501, 100), fontname='Arial')
235
236 ax.set_yticks(np.arange(0.5, 2.6, 0.5))
237 ax.set_ylim(0.5, 2.5)
238
239 # Set the y-axis height for the bottom spine
240 y_axis_height = 1.5
241 ax.spines['bottom'].set_position(('data', y_axis_height))
242
243 # Manually adjust x-axis label position
244 ax.xaxis.set_label_coords(-0.5, 0.0) # Adjust x-axis label position
245
246 # Ensure a tick mark at the end of x-axis without a corresponding value
247 xticks = np.arange(100, 501, 100)
248 plt.xlim(15, 525)
249 xticks = np.append(xticks, 525) # Add the endpoint where you want the tick without a label
250 ax.set_xticks(xticks)
251 ax.set_xticklabels([str(int(x)) if x != 525 else '' for x in xticks]) # Set labels; use ''
    for the endpoint
252
253 # Customize the ticks
254 plt.tick_params(axis='y', which='major', direction='out', length=2, width=1, colors='black')
255 plt.tick_params(axis='x', which='major', direction='out', length=2, width=1, colors='black')
256 plt.gca().yaxis.set_ticks_position('left')
257 plt.gca().xaxis.set_ticks_position('bottom')# Ensure ticks are on the left side of the plot
258 plt.tick_params(axis='y', which='major', direction='out', length=2, width=1, colors='black')
259 plt.tick_params(axis='x', which='major', direction='inout', length=4, width=1, colors='black'
    )
260 # Set labels and legend
261 plt.ylabel('Scalar(-)', fontsize=10, fontname='Arial')
262 plt.xlabel('Amount of candidate solutions(-)', fontsize=10, fontname='Arial')
263 plt.legend(prop={'family': 'Arial'}, loc='upper right')
264
265 # Show the plot
266 plt.show()
267
268 plt.figure(figsize=(7, 5))
269
270 plt.plot(dfs[0], dfs[2], color='lightsteelblue', label = '1 in x and 1 in y',marker='s',
    markersize=4.5,markededgewidth=0.5, markededgecolor='black', linewidth=1, linestyle='-')
271 #plt.plot(dfs[0], productGA, color='steelblue', label = 'GA',marker='o', markersize=5,
    linewidth=0.8,markededgewidth=0.5, markededgecolor='black', linestyle='-')
272 #plt.plot(dfs[0], dfs[7], color='black', label = '2 in x and 2 in y',marker='v', markersize
    =6, linewidth=0.8, markededgewidth=0.5, markededgecolor='black', markerfacecolor='white',
    linestyle='-')
273 plt.plot(dfs[0], dfs[7], color='firebrick', label = '1 in x and 2 in y',marker='^',
    markersize=6, linewidth=0.8,markededgewidth=0.5, markededgecolor='black', zorder=3)
274 #plt.plot(dfs[0], productCMCr, color='goldenrod', label='CMC with  $\rho$ ', marker='D',
    markersize=4, markededgewidth=0.5, markededgecolor='black', linewidth=0.8, linestyle='-',
    zorder =1)
275
276 # Customize the grid and spines
277 plt.grid(True, which='both', linestyle(':', linewidth='0.5', color='white')
278 ax = plt.gca()
279 ax.spines['top'].set_linewidth(0.5)
280 ax.spines['top'].set_color('white')
281 ax.spines['right'].set_linewidth(0.5)
282 ax.spines['right'].set_color('white')
283 ax.spines['bottom'].set_linewidth(0.5)
284 ax.spines['bottom'].set_color('black')
285 ax.spines['left'].set_linewidth(0.5)
286 ax.spines['left'].set_color('black')
287
288 # Set tick positions and limits
289 plt.xticks(np.arange(0, 501, 100), fontname='Arial')
290

```

```

291 ax.set_yticks(np.arange(3, 11.1, 2))
292 ax.set_ylim(3, 11)
293
294 # Set the y-axis height for the bottom spine
295 y_axis_height = 7.0
296 ax.spines['bottom'].set_position(('data', y_axis_height))
297
298 # Manually adjust x-axis label position
299 ax.xaxis.set_label_coords(-0.5, 0.0) # Adjust x-axis label position
300 ax.yaxis.set_major_formatter(plt.FormatStrFormatter('% .2f'))
301 # Ensure a tick mark at the end of x-axis without a corresponding value
302 xticks = np.arange(100, 501, 100)
303 plt.xlim(15, 525)
304 xticks = np.append(xticks, 525) # Add the endpoint where you want the tick without a label
305 ax.set_xticks(xticks)
306 ax.set_xticklabels([str(int(x)) if x != 525 else '' for x in xticks]) # Set labels; use ''
    for the endpoint
307
308 # Customize the ticks
309 plt.tick_params(axis='y', which='major', direction='out', length=2, width=1, colors='black')
310 plt.tick_params(axis='x', which='major', direction='out', length=2, width=1, colors='black')
311 plt.gca().yaxis.set_ticks_position('left')
312 plt.gca().xaxis.set_ticks_position('bottom')# Ensure ticks are on the left side of the plot
313 plt.tick_params(axis='y', which='major', direction='out', length=2, width=1, colors='black')
314 plt.tick_params(axis='x', which='major', direction='inout', length=4, width=1, colors='black'
    )
315
316 # Set labels and legend
317 plt.ylabel('Scalar(-)', fontsize=10, fontname='Arial')
318 plt.xlabel('Amount of candidate solutions(-)', fontsize=10, fontname='Arial')
319 plt.legend(prop={'family': 'Arial'}, loc='upper right')
320
321 # Show the plot
322 plt.show()
323
324 plt.figure(figsize=(7, 5))
325 plt.plot(dfs[0], dfs[3], color='lightsteelblue', label = '1 in x and 1 in y', marker='s',
    markersize=4.5, markeredgewidth=0.5, markeredgecolor='black', linewidth=1, linestyle='-')
326 #plt.plot(dfs[0], productGA, color='steelblue', label = 'GA', marker='o', markersize=5,
    linewidth=0.8, markeredgewidth=0.5, markeredgecolor='black', linestyle='-')
327 #plt.plot(dfs[0], dfs[13], color='black', label = '2 in x and 2 in y', marker='v', markersize
    =6, linewidth=0.8, markeredgewidth=0.5, markeredgecolor='black', markerfacecolor='white',
    linestyle='-')
328 plt.plot(dfs[0], dfs[8], color='firebrick', label = '1 in x and 2 in y', marker='^',
    markersize=6, linewidth=0.8, markeredgewidth=0.5, markeredgecolor='black', zorder=3)
329 #plt.plot(dfs[0], productCMCr, color='goldenrod', label='CMC with  $\rho$ ', marker='D',
    markersize=4, markeredgewidth=0.5, markeredgecolor='black', linewidth=0.8, linestyle='-',
    zorder=1)
330
331 # Customize the grid and spines
332 plt.grid(True, which='both', linestyle=':', linewidth='0.5', color='white')
333 ax = plt.gca()
334 ax.spines['top'].set_linewidth(0.5)
335 ax.spines['top'].set_color('white')
336 ax.spines['right'].set_linewidth(0.5)
337 ax.spines['right'].set_color('white')
338 ax.spines['bottom'].set_linewidth(0.5)
339 ax.spines['bottom'].set_color('black')
340 ax.spines['left'].set_linewidth(0.5)
341 ax.spines['left'].set_color('black')
342
343 # Set tick positions and limits
344 plt.xticks(np.arange(0, 501, 100), fontname='Arial')
345
346 #ax.set_yticks(np.arange(0, 14.1, 2))
347 #ax.set_ylim(0.0, 14)
348 ax.set_yticks(np.arange(0.36, 0.481, 0.02))
349 #ax.set_ylim(0.3990, 0.403)
350 # Set the y-axis height for the bottom spine
351 y_axis_height = 0.4
352 ax.spines['bottom'].set_position(('data', y_axis_height))
353 ax.yaxis.set_major_formatter(plt.FormatStrFormatter('% .2f'))

```



```

333 # Manually adjust x-axis label position
334 ax.xaxis.set_label_coords(-0.5, 0.0) # Adjust x-axis label position
335
336 # Ensure a tick mark at the end of x-axis without a corresponding value
337 xticks = np.arange(100, 501, 100)
338 plt.xlim(15, 525)
339 xticks = np.append(xticks, 525) # Add the endpoint where you want the tick without a label
340 ax.set_xticks(xticks)
341 ax.set_xticklabels([str(int(x)) if x != 525 else '' for x in xticks]) # Set labels; use ''
    for the endpoint
342
343 # Customize the ticks
344 plt.tick_params(axis='y', which='major', direction='out', length=2, width=1, colors='black')
345 plt.tick_params(axis='x', which='major', direction='out', length=2, width=1, colors='black')
346 plt.gca().yaxis.set_ticks_position('left')
347 plt.gca().xaxis.set_ticks_position('bottom')# Ensure ticks are on the left side of the plot
348 plt.tick_params(axis='y', which='major', direction='out', length=2, width=1, colors='black')
349 plt.tick_params(axis='x', which='major', direction='inout', length=4, width=1, colors='black'
    )
350
351 # Set labels and legend
352 plt.ylabel('Scalar_(-)', fontsize=10, fontname='Arial')
353 plt.xlabel('Amount_of_candidate_solutions_(-)', fontsize=10, fontname='Arial')
354 plt.legend(prop={'family': 'Arial'}, loc='upper_right')
355
356 # Show the plot
357 plt.show()
358
359 plt.figure(figsize=(7, 5))
360
361 plt.plot(dfs[0], dfs[4], color='lightsteelblue', label = '1_in_x_and_1_in_y',marker='s',
    markersize=4.5,markeredgewidth=0.5, markeredgcolor='black', linewidth=1, linestyle='-')
362 #plt.plot(dfs[0], productGA, color='steelblue', label = 'GA',marker='o', markersize=5,
    linewidth=0.8,markedgedwidth=0.5, markeredgcolor='black',  linestyle='-')
363 #plt.plot(dfs[0], dfs[14], color='black', label = '2 in x and 2 in y',marker='v', markersize
    =6, linewidth=0.8, markedgedwidth=0.5, markeredgcolor='black', markerfacecolor='white',
    linestyle='-')
364 plt.plot(dfs[0], dfs[9], color='firebrick', label = '1_in_x_and_2_in_y',marker='^',
    markersize=6, linewidth=0.8,markedgedwidth=0.5, markeredgcolor='black', zorder=3)
365 #plt.plot(dfs[0], productCMCr, color='goldenrod', label='CMC with $\rho$', marker='D',
    markersize=4, markedgedwidth=0.5, markeredgcolor='black', linewidth=0.8, linestyle='-',
    zorder =1)
366
367 # Customize the grid and spines
368 plt.grid(True, which='both', linestyle=':', linewidth='0.5', color='white')
369 ax = plt.gca()
370 ax.spines['top'].set_linewidth(0.5)
371 ax.spines['top'].set_color('white')
372 ax.spines['right'].set_linewidth(0.5)
373 ax.spines['right'].set_color('white')
374 ax.spines['bottom'].set_linewidth(0.5)
375 ax.spines['bottom'].set_color('black')
376 ax.spines['left'].set_linewidth(0.5)
377 ax.spines['left'].set_color('black')
378
379 # Set tick positions and limits
380 plt.xticks(np.arange(0, 501, 100), fontname='Arial')
381
382 #ax.set_yticks(np.arange(0, 14.1, 2))
383 #ax.set_ylim(0.0, 14)
384 ax.set_yticks(np.arange(7, 11.1, 1))
385 ax.set_ylim(7, 11)
386 # Set the y-axis height for the bottom spine
387 y_axis_height = 9.0
388 ax.spines['bottom'].set_position(('data', y_axis_height))
389
390 # Manually adjust x-axis label position
391 ax.xaxis.set_label_coords(-0.5, 0.0) # Adjust x-axis label position
392 ax.yaxis.set_major_formatter(plt.FormatStrFormatter('%2f'))
393 # Ensure a tick mark at the end of x-axis without a corresponding value
394 xticks = np.arange(100, 501, 100)
395 plt.xlim(15, 525)

```



```

415 xticks = np.append(xticks, 525) # Add the endpoint where you want the tick without a label
416 ax.set_xticks(xticks)
417 ax.set_xticklabels([str(int(x)) if x != 525 else '' for x in xticks]) # Set labels; use ''
    for the endpoint
418
419 # Customize the ticks
420 plt.tick_params(axis='y', which='major', direction='out', length=2, width=1, colors='black')
421 plt.tick_params(axis='x', which='major', direction='out', length=2, width=1, colors='black')
422 plt.gca().yaxis.set_ticks_position('left')
423 plt.gca().xaxis.set_ticks_position('bottom') # Ensure ticks are on the left side of the plot
424 plt.tick_params(axis='y', which='major', direction='out', length=2, width=1, colors='black')
425 plt.tick_params(axis='x', which='major', direction='inout', length=4, width=1, colors='black'
    )
426 # Set labels and legend
427 plt.ylabel('Scalar(-)', fontsize=10, fontname='Arial')
428 plt.xlabel('Amount of candidate solutions(-)', fontsize=10, fontname='Arial')
429 plt.legend(prop={'family': 'Arial'}, loc='upper right')
430
431 # Show the plot
432 plt.show()

```

## Measurement Uncertainties

```

1 import time
2 %load_ext autoreload
3 %autoreload 2
4 from copy import deepcopy
5 from pyvibe.models.Timoshenko import Timo, TimoPieceWiseBeam1D, TimoPieceWiseBeam2D,
    UpdatingMethod, cost_func1D, cost_func2D, update_model
6 %matplotlib inline
7 import matplotlib.pyplot as plt
8 import numpy as np
9
10 # Dimension Parameters:
11 L = 153.87 #[m]
12 Lt = np.array([3.15, 22.65, 46.8, 81.27]) #[m]
13 A = np.array([1320, 1320, 1320, 825]) #[m]
14 I_yy = np.array([4312, 4312, 4744, 1402]) #[m4]
15 I_xx = np.array([34350, 34350, 35105, 10685]) #[m4]
16 H = Lt/L #[-]
17 # Material properties:
18 rho = np.array([5070, 552, 426, 371]) #[kg/m3]
19 E = np.array([11e9, 11e9, 22e9, 33e9]) #[N/m2]
20 v = 0.2 #[-]
21 G = E/(2*(1+v))#[N/m^2]
22 k = 0.85 #[-]
23
24 # Springs:
25 kr_y = 6486e9 #[Nm/rad]
26 kr_x = 50371e9 #[Nm/rad]
27 kt_x = 9.22e9 #[N/m]
28 kt_y = 9.22e9 #[N/m]
29
30 #boundary conditions:
31 bc = ("rot+transl", "free")
32 scaler = {"kr":0.4, "kt":9.0, "I":1.5, "E":7.0}
33 zz_trgt = np.array([0, 69, 108.60, 144.60])
34 z_trgt = zz_trgt/L
35
36 #Startpoints
37 startpoints = 200
38
39 # Number of iterations
40 num_iterations = 10
41
42 timoshenko_x = []
43 for ix in range(len(H)):
44     timoshenko_x.append(Timo(params={"E": E[ix], "I": I_xx[ix], "L": L, "rho": rho[ix], "A": A[
        ix], "kG": k * G[ix], "kr": kr_x, "kt": kt_x}, height=H[ix]))
45 timoshenko_y = []

```

---

```

46 for ix in range(len(H)):
47     timoshenko_y.append(Timo(params={"E": E[ix], "I": I_yy[ix], "L": L, "rho": rho[ix], "A": A[
48         ix], "kG": k * G[ix], "kr": kr_y, "kt": kt_y}, height=H[ix]))
49 model = TimoPieceWiseBeam2D(beams_x=timoshenko_x, beams_y=timoshenko_y, bc = bc)
50 import numpy as np
51
52 def apply_random_errors(input_data, max_error=0.2, seed=None):
53     if seed is not None:
54         np.random.seed(seed)
55
56     if input_data.ndim == 1:
57         data_with_errors = np.copy(input_data)
58         scaling_factors = np.random.uniform(1 - max_error, 1 + max_error, input_data.shape)
59         data_with_errors *= scaling_factors
60
61     elif input_data.ndim == 2:
62         data_with_errors = np.copy(input_data)
63         scaling_factors = np.random.uniform(1 - max_error, 1 + max_error, input_data.shape)
64         data_with_errors *= scaling_factors
65
66     else:
67         raise ValueError("Input data must be either a 1D or 2D array.")
68
69     return data_with_errors
70
71 model.scale_params(scalers=scaler)
72 fn_x, fn_y, phi_x, phi_y, pts_x, pts_y = model.modeshapes(pts=z_trgt, n_modes_x = 1,
73     n_modes_y=2)
74 parameters_to_update = {p:[1/10, 9.8] for p in list(scaler.keys())}
75 logscale = False
76
77 def process_iteration(npts):
78     results = update_model(
79         model=model,
80         cost_func=cost_func2D,
81         parameters_to_update=parameters_to_update,
82         y_hat=y_hat,
83         npts=npts,
84         sim_type=UpdatingMethod.CMC,
85         verbose=False,
86         logscale=logscale)
87
88     parameters_columns = [cc for cc in results.columns if "_st" in cc or "_opt" in cc]
89     plot_cols = [a + "_opt" for a in parameters_to_update.keys()]
90     map_opt2name = {pcl: k for k, pcl in zip(parameters_to_update.keys(), plot_cols)}
91     df_opt = results.dropna().copy()
92     df_opt.dropna(inplace=True)
93
94     if logscale:
95         df_opt[parameters_columns] = df_opt[parameters_columns].apply(np.exp)
96
97     top_lowest_convergence = df_opt.sort_values(by='Convergence').head(1)
98     return (npts,
99         top_lowest_convergence["kr_opt"].iloc[0],
100         top_lowest_convergence["kt_opt"].iloc[0],
101         top_lowest_convergence["I_opt"].iloc[0],
102         top_lowest_convergence["E_opt"].iloc[0],
103         top_lowest_convergence["Convergence"].iloc[0])
104
105 kr = []
106 kt = []
107 I = []
108 E = []
109 convergence = []
110
111 for i in range(num_iterations):
112     seed_fn_xt = i * 4
113     seed_phi_xt = i * 4 + 1
114     seed_fn_yt = i * 4 + 2
115     seed_phi_yt = i * 4 + 3

```

```

115
116     fn_xt = apply_random_errors(deepcopy(fn_x), max_error=0.1, seed=seed_fn_xt)
117     phi_xt = apply_random_errors(deepcopy(phi_x), max_error=0.2, seed=seed_phi_xt)
118     fn_yt = apply_random_errors(deepcopy(fn_y), max_error=0.1, seed=seed_fn_yt)
119     phi_yt = apply_random_errors(deepcopy(phi_y), max_error=0.2, seed=seed_phi_yt)
120
121     y_hat = {"z": z_trgt, "fn_x": deepcopy(fn_xt), "fn_y": deepcopy(fn_yt), "modes_x":
122             deepcopy(phi_xt), "modes_y": deepcopy(phi_yt)}
123
124     result = process_iteration(startpoints)
125
126     kr.append(result[1])
127     kt.append(result[2])
128     I.append(result[3])
129     E.append(result[4])
130     convergence.append(result[5])
131
132 import os
133 import pandas as pd
134 directory = r'C:\Users\ritfeldis\OneDrive-TNO\Documents\Master_thesis_Isa_Ritfeld(4892534)
135 \5_Python\5.3_Results\DelfstePoort'
136 os.makedirs(directory, exist_ok=True)
137 file_names = [
138     'Measure_I.csv',
139     'Measure_E.csv',
140     'Measure_kr.csv',
141     'Measure_kt.csv',
142     'Measure_convergence.csv',
143 ]
144 data_lists = [I, E, kr, kt, convergence]
145 parameter_names = ['I', 'E', 'kr', 'kt', 'convergence']
146
147 for file_name, data, parameter in zip(file_names, data_lists, parameter_names):
148     df = pd.DataFrame({'value_of_{parameter}': data})
149     file_path = os.path.join(directory, file_name)
150     df.to_csv(file_path, index=False)
151
152 directory = r'C:\Users\ritfeldis\OneDrive-TNO\Documents\Master_thesis_Isa_Ritfeld(4892534)
153 \5_Python\5.3_Results\DelfstePoort'
154 file_names = [
155     'Measure_I.csv',
156     'Measure_E.csv',
157     'Measure_kr.csv',
158     'Measure_kt.csv',
159     'Measure_convergence.csv',
160 ]
161 dfs = []
162 for file_name in file_names:
163     file_path = os.path.join(directory, file_name)
164     df = pd.read_csv(file_path)
165     dfs.append(df)
166
167 import os
168 import pandas as pd
169 import matplotlib.pyplot as plt
170
171 # Define the directory and file names
172 directory = r'C:\Users\ritfeldis\OneDrive-TNO\Documents\Master_thesis_Isa_Ritfeld(4892534)
173 \5_Python\5.3_Results\DelfstePoort'
174 file_names = [
175     'Measure_I.csv',
176     'Measure_E.csv',
177     'Measure_kr.csv',
178     'Measure_kt.csv',
179     'Measure_convergence.csv',
180 ]
181 column_names = ['I', 'E', 'kr', 'kt', 'convergence']
182
183 # Load the data into a list of DataFrames
184 dfs = []
185 for file_name, col_name in zip(file_names, column_names):

```

```

182     file_path = os.path.join(directory, file_name)
183     df = pd.read_csv(file_path)
184     df.columns = [col_name]
185     dfs.append(df)
186
187 # Merge all DataFrames on the index
188 merged_df = pd.concat(dfs, axis=1)
189
190 # Add 'EI_opt' column which is the product of 'E' and 'I'
191 merged_df['EI'] = merged_df['E'] * merged_df['I']
192 print(merged_df['EI'])
193 # Define the additional solution
194 additional_solution = {
195     'kr': 0.4,
196     'kt': 9.0,
197     'E': 7.0,
198     'I': 1.5,
199     'EI': 7.0 * 1.5
200 }
201
202 # Define columns to plot
203 columns = ['kr', 'kt', 'E', 'I', 'EI']
204
205 # Define colors for each column
206 colors = {
207     'kr': 'steelblue',
208     'kt': 'lightsteelblue',
209     'I': 'firebrick',
210     'E': 'goldenrod',
211     'EI': 'white'
212 }
213
214 # Define marker styles for each column
215 markers = {
216     'kr': 'o',      # circle
217     'kt': 's',      # square
218     'I': '^',       # triangle up
219     'E': 'D',       # diamond
220     'EI': 'v'       # plus (filled)
221 }
222
223 # Define marker sizes for each column
224 marker_sizes = {
225     'kr': 50,        # Adjust marker size as needed
226     'kt': 43,
227     'I': 60,
228     'E': 35,
229     'EI': 60
230 }
231
232 # Define the additional solution color
233 additional_color = 'red'
234
235 # Set up the plot
236 plt.figure(figsize=(9, 6))
237
238 # Create scatter plot for each column with thin black edges
239 for col in columns:
240     plt.scatter(
241         [col] * len(merged_df),
242         merged_df[col],
243         label=col,
244         color=colors[col],
245         marker=markers[col],
246         edgecolor='black',
247         linewidth=0.5,
248         s=marker_sizes[col],
249         zorder=2
250     )
251
252 # Connect the dots for each row

```

```

253 for i in range(len(merged_df)):
254     plt.plot(columns, merged_df.iloc[i][columns], color='dimgrey', linestyle='-', linewidth
                =0.5, zorder=1)
255
256 # Plot the additional solution with corresponding markers and additional color
257 for col in columns:
258     plt.scatter(
259         [col],
260         [additional_solution[col]],
261         color=additional_color,
262         marker=markers[col],
263         edgecolor='black',
264         linewidth=0.5,
265         s=marker_sizes[col],
266         zorder=3
267     )
268 plt.plot(columns, list(additional_solution.values()), color=additional_color, linestyle='-',
            linewidth=0.5, zorder=3)
269
270 # Add labels and title
271 plt.xlabel('Optimized updating parameters', fontsize=12)
272 plt.ylabel('Scalar(-)', fontsize=12)
273 plt.xticks(rotation=45)
274
275 # Customize grid
276 plt.grid(True, which='major', axis='x', linestyle='-', linewidth=0.5, color='black') # Major
            vertical grid lines
277
278 # Hide major horizontal grid lines
279 plt.grid(True, which='major', axis='y', linestyle='-', linewidth=0.5, color='white')
280
281 # Add ticks where vertical grid lines intersect horizontal lines
282 for col in columns:
283     y_vals = [0.1, 1, 10, 20]
284     for y_val in y_vals:
285         plt.scatter(col, y_val, color='black', marker='_', linewidths=1.7, s=40, zorder=0)
286
287 for col in columns:
288     y_vals = [0.2,0.3,0.4, 0.5,0.6,0.7,0.8,0.9, 2,3,4,5,6,7,8,9]
289     for y_val in y_vals:
290         plt.scatter(col, y_val, color='black', marker='_', linewidths=0.5, s=2, zorder=0)
291
292 # Adjust spines
293 plt.gca().spines['top'].set_linewidth(0.5)
294 plt.gca().spines['top'].set_color('white')
295 plt.gca().spines['right'].set_linewidth(0.5)
296 plt.gca().spines['right'].set_color('white')
297 plt.gca().spines['bottom'].set_linewidth(0.5)
298 plt.gca().spines['bottom'].set_color('white')
299 plt.gca().spines['left'].set_linewidth(0.5)
300 plt.gca().spines['left'].set_color('black')
301 plt.gca().yaxis.set_ticks_position('left') # Ensure ticks are on the left side of the plot
302 plt.tick_params(axis='y', which='major', direction='out', length=5, width=1, colors='black')
            # Customize tick parameters
303
304 # Set y-scale and limits
305 plt.yscale('log')
306 plt.ylim(0.1, 20)
307 plt.gca().set_yticks([0.1, 1, 10, 20])
308 plt.gca().set_yticklabels(['0.1', '1', '10', '20'])
309
310 # Show legend for main columns only
311 plt.legend(columns, bbox_to_anchor=(0.05, 1), loc='upper left', prop={'family': 'Arial', '
            size': 10})
312
313 # Display the plot
314 plt.tight_layout()
315 plt.show()

```

## Application

```

1 %load_ext autoreload
2 %autoreload 2
3 from copy import deepcopy
4 from pyvibe.models.Timoshenko import Timo, TimoPieceWiseBeam1D, TimoPieceWiseBeam2D,
   UpdatingMethod, cost_func1D, cost_func2D, update_model
5 import numpy as np
6 import matplotlib.pyplot as plt
7 import seaborn as sns
8 import scipy.stats as stats
9 import math
10 import pandas as pd
11 %matplotlib inline
12
13 # Parameters
14
15 # Dimension Parameters:
16 L = 160.58 #[m]
17 Lt = np.array([5.735, 11.47, 26.640, 89.91, 26.825]) #[m]
18 B = 28 #[m]
19 D = 28 #[m]
20 A = B * D #[m2]
21 I_x = np.array([619.21, 948.47, 2727.22, 2335.82, 905.52]) #[m4]
22 I_y = np.array([1536.25, 959.01, 1245.46, 1202.07, 831.57]) #[m4]
23 I_yy = np.array([1444, 1500, 1393, 1389, 1293]) #[m4]
24 I_xx = np.array([1532, 1548, 2796, 2324, 760]) #[m4]
25 H = Lt/L #[-]
26
27 # Material properties:
28 rho = np.array([1933, 495, 486, 440, 383]) #[kg/m3]
29 E = np.array([38.2e9, 38.2e9, 38.2e9, 38.2e9, 32.8e9]) #[N/m2]
30 v = 0.2 #[-]
31 G = E/(2*(1+v)) #[N/m^2]
32 k = 0.85 #[-]
33
34 # Springs:
35 kr_x = 2380e9 #[Nm/rad]
36 kr_y = 2625e9 #[Nm/rad]
37 kt_y = 19e9 #[N/m]
38 kt_x = 4e9 #[N/m]
39
40 #boundary conditions:
41 bc = ("rot+transl", "free")
42
43 scaler = {"kr":0.4, "kt":9.0, "I":1.5, "E":7.0} #The parameters chosen are from the
   conclusion of sensitivity study 2
44 zz_trgt = np.array([57.165, 120.435, 153.735]) #The three locations where there are
   measurements taken of the New Orleans
45 z_trgt = zz_trgt/L
46
47 #Startpoints
48 startpoints = 400
49
50 fnx = np.array([0.282, 1.527])
51 phixx = [
52     [-0.30, 0.89],
53     [-0.71, -0.23],
54     [-1, -1]
55 ]
56 phix = np.array(phixx)
57
58 # Procedure
59
60 def adjust_modes(phi1, phi2):
61     rmse_plus = np.sqrt(np.mean((phi1 - phi2) ** 2))
62     rmse_minus = np.sqrt(np.mean((phi1 + phi2) ** 2))
63     flip = np.ones_like(phi2)
64     flip[rmse_plus > rmse_minus] = -1
65     phi_flip = phi1 * flip

```

```

66     return phi_flip
67
68 def modal_assurance_criterion(phi1, phi2):
69     phi2 = adjust_modes(phi2, phi1)
70     mac = np.abs(np.conj(phi1).dot(phi2))**2 / \
71         (np.conj(phi1).dot(phi1) * np.conj(phi2).dot(phi2))
72     return mac
73
74 timoshenko_x = []
75 for ix in range(len(H)):
76     timoshenko_x.append(Timo(params={"E": E[ix], "I": I_xx[ix], "L": L, "rho": rho[ix], "A": A
77         , "kG": k * G[ix], "kr": kr_x, "kt": kt_x}, height=H[ix]))
78 model_x = TimoPieceWiseBeam1D(beams=timoshenko_x, bc = bc)
79
80 y_hat_x = {"z": z_trgt, "fn": deepcopy(fnx), "modes": deepcopy(phix)}
81 parameters_to_update_x = {p: [1/10, 10] for p in list(scaler.keys())} #maybe change this?
82 print(parameters_to_update_x)
83 logscale = False
84
85 results_x = update_model(
86     model=model_x,
87     cost_func=cost_func1D,
88     parameters_to_update=parameters_to_update_x,
89     y_hat=y_hat_x,
90     npts=startpoints,
91     sim_type=UpdatingMethod.CMC,
92     verbose=False,
93     logscale=logscale
94 )
95 parameters_columns = [cc for cc in results_x.columns if "_st" in cc or "_opt" in cc]
96 plot_cols = [a + "_opt" for a in parameters_to_update_x.keys()]
97 map_opt2name = {pcl: k for k, pcl in zip(parameters_to_update_x.keys(), plot_cols)}
98 df_opt = results_x.copy()
99 df_opt.dropna(inplace=True)
100
101 if logscale:
102     df_opt[parameters_columns] = df_opt[parameters_columns].apply(np.exp)
103
104 # Results
105 t_x = results_x.sort_values(by='Convergence').head(startpoints)
106
107 quantile_threshold = t_x['Convergence'].quantile(0.125)
108 test_x = t_x[t_x['Convergence'] <= quantile_threshold]
109 print(test_x)
110
111 # Find the minimum and maximum values in each _opt column
112 min_values = test_x[[col for col in test_x.columns if col.endswith('_opt')]].min()
113 max_values = test_x[[col for col in test_x.columns if col.endswith('_opt')]].max()
114
115 # Update the boundaries
116 new_boundaries = {
117     'kr': [min_values['kr_opt'], max_values['kr_opt']],
118     'kt': [min_values['kt_opt'], max_values['kt_opt']],
119     'I': [min_values['I_opt'], max_values['I_opt']],
120     'E': [min_values['E_opt'], max_values['E_opt']]
121 }
122 print("New Boundaries:")
123 print(new_boundaries)
124
125 results_repeated2_x = update_model(
126     model=model_x,
127     cost_func=cost_func1D,
128     parameters_to_update=new_boundaries,
129     y_hat=y_hat_x,
130     npts=startpoints,
131     sim_type=UpdatingMethod.CMC,
132     verbose=False,
133     logscale=logscale
134 )
135 parameters_columns = [cc for cc in results_repeated2_x.columns if "_st" in cc or "_opt" in cc

```

```

    ]
136 plot_cols = [a + "_opt" for a in new_boundaries.keys()]
137 map_opt2name = {pcl: k for k, pcl in zip(new_boundaries.keys(), plot_cols)}
138 df_opt = results_repeated2_x.copy()
139 df_opt.dropna(inplace=True)
140
141 if logscale:
142     df_opt[parameters_columns] = df_opt[parameters_columns].apply(np.exp)
143
144 t2_x = results_repeated2_x.sort_values(by='Convergence').head(startpoints)
145 print(t2_x)
146
147 quantile_threshold = t2_x['Convergence'].quantile(0.125)
148 test2_x = t2_x[t2_x['Convergence'] <= quantile_threshold]
149 print(test2_x)
150
151 results_output_x = []
152 for ii in range(len(test2_x)):
153     scaler_x = {
154         "kr": test2_x["kr_opt"].iloc[ii],
155         "kt": test2_x["kt_opt"].iloc[ii],
156         "I": test2_x["I_opt"].iloc[ii],
157         "E": test2_x["E_opt"].iloc[ii]
158     }
159     model_x.scale_parms(scalers=scaler_x)
160     fn_x, phi_x, pts_x = model_x.modeshapes(pts=z_trgt, n_modes=2)
161
162     mac_0 = modal_assurance_criterion(phi_x[:, 0], phix[:, 0])
163     mac_1 = modal_assurance_criterion(phi_x[:, 1], phix[:, 1])
164
165     results_output_x.append({
166         "index": ii,
167         "fn_x_1": fn_x[0],
168         "fn_x_2": fn_x[1],
169         "mac_1": mac_0,
170         "mac_2": mac_1
171     })
172
173 results_df = pd.DataFrame(results_output_x)
174 stats_results = {}
175 z_value = 1.645
176
177 for col in ['fn_x_1', 'fn_x_2', 'mac_1', 'mac_2']:
178     mean_val = results_df[col].mean()
179     median_val = results_df[col].median()
180     max_val = results_df[col].max()
181     min_val = results_df[col].min()
182
183     stats_results[col] = {
184         "Median": median_val,
185         "Mean": mean_val,
186         "lower_bound": min_val,
187         "upper_bound": max_val
188     }
189 summary_df = pd.DataFrame(columns=["Measured_fnx", "Median", "Mean", "Lower_Bound", "Upper_
    Bound"])
190 for param, values in stats_results.items():
191     if "fn_x_1" in param:
192         measured_fnx = fnx[0]
193     elif "fn_x_2" in param:
194         measured_fnx = fnx[1]
195     else:
196         measured_fnx = np.nan
197
198     summary_df.loc[param] = [measured_fnx, values["Median"], values["Mean"], values["
        lower_bound"],
199                             values["upper_bound"]]
200 print(summary_df)
201
202 def calculate_summary_stats(df, columns, direction):
203     print(f"{direction}-direction")

```



```

204
205 kr_summary_stats = []
206 kt_summary_stats = []
207
208 for column in columns:
209     if "kr_opt" in column:
210         mean_val = df[column].mean()
211         median_val = df[column].median()
212         sem_val = df[column].sem()
213         std_dev = df[column].std()
214         z_value = 1.645
215
216         lower_boundkr = df[column].min()
217         upper_boundkr = df[column].max()
218
219         cv = round((std_dev / mean_val) * 100, 2)
220
221         kr_summary_stats.append(pd.DataFrame({
222             'Design': 1,
223             'Median': median_val,
224             'Lower_Bound': lower_boundkr,
225             'Upper_Bound': upper_boundkr,
226             'COV(%)': cv
227         }, index=["kr_opt"]))
228
229     elif "kt_opt" in column:
230         mean_val = df[column].mean()
231         median_val = df[column].median()
232         sem_val = df[column].sem()
233         std_dev = df[column].std()
234         z_value = 1.645
235
236         lower_boundkt = df[column].min()
237         upper_boundkt = df[column].max()
238
239         cv = round((std_dev / mean_val) * 100, 2)
240
241         kt_summary_stats.append(pd.DataFrame({
242             'Design': 1,
243             'Median': median_val,
244             'Lower_Bound': lower_boundkt,
245             'Upper_Bound': upper_boundkt,
246             'COV(%)': cv
247         }, index=["kt_opt"]))
248
249 kr_summary_stats_df = pd.concat(kr_summary_stats, axis=0)
250 kt_summary_stats_df = pd.concat(kt_summary_stats, axis=0)
251
252 print("Summary_statistics_for_boundary_conditions:")
253 print(kr_summary_stats_df)
254 print(kt_summary_stats_df.to_string(header=None))
255
256 for ii in range(len(E)):
257     print("")
258     print(f"Summary_statistics_for_discrete_element_{ii}")
259     summary_stats_list = []
260
261     for column in columns:
262         if "kr_opt" in column or "kt_opt" in column:
263             continue
264
265         mean_val = df[column].mean()
266         median_val = df[column].median()
267         sem_val = df[column].sem()
268         std_dev = df[column].std()
269         z_value = 1.645
270
271         lower_bound = df[column].min()
272         upper_bound = df[column].max()
273
274         cv = round((std_dev / mean_val) * 100, 2)

```

```

275
276     if 'y' in direction:
277         if "E_opt" in column:
278             summary_stats = pd.DataFrame({
279                 'Design': 1,
280                 'Median': median_val,
281                 'Lower_Bound': lower_bound ,
282                 'Upper_Bound': upper_bound,
283                 'COV(%)': cv
284             }, index=["E_opt"])
285             summary_stats_list.append(summary_stats)
286         elif "I_opt" in column:
287             summary_stats = pd.DataFrame({
288                 'Design': 1,
289                 'Median': median_val,
290                 'Lower_Bound': lower_bound ,
291                 'Upper_Bound': upper_bound,
292                 'COV(%)': cv
293             }, index=["I_opt"])
294             summary_stats_list.append(summary_stats)
295             new_column = df["E_opt"] * df["I_opt"]
296             mean_val_new = new_column.mean()
297             median_val_new = new_column.median()
298             std_dev_new = new_column.std()
299
300             lower_bound_new = new_column.min()
301             upper_bound_new = new_column.max()
302
303             cv_new = round((std_dev_new / mean_val_new) * 100, 2)
304
305             summary_stats = pd.DataFrame({
306                 'Design': 1,
307                 'Median': median_val_new,
308                 'Lower_Bound': lower_bound_new ,
309                 'Upper_Bound': upper_bound_new,
310                 'COV(%)': cv_new
311             }, index=["EI_opt"])
312             summary_stats_list.append(summary_stats)
313
314         summary_stats_df = pd.concat(summary_stats_list, axis=0)
315         print(summary_stats_df)
316
317 parameters_columns_y = [cc for cc in test2_x.columns if "_opt" in cc]
318 summary_stats_y = calculate_summary_stats(test2_x, parameters_columns_y, "y")
319
320 import matplotlib.pyplot as plt
321
322 # Assuming test2_x is already defined and has the columns 'kr_opt', 'kt_opt', 'I_opt', 'E_opt
323     ', and 'EI_opt'
324
325 # Add a new column 'EI_opt' which is the product of 'E_opt' and 'I_opt'
326 test2_x['EI_opt'] = test2_x['E_opt'] * test2_x['I_opt']
327
328 # Columns to plot
329 columns = ['kr_opt', 'kt_opt', 'E_opt', 'I_opt', 'EI_opt']
330
331 # Define colors for each column
332 colors = {
333     'kr_opt': 'steelblue',
334     'kt_opt': 'lightsteelblue',
335     'I_opt': 'firebrick',
336     'E_opt': 'goldenrod',
337     'EI_opt': 'white'
338 }
339
340 # Define marker styles for each column
341 markers = {
342     'kr_opt': 'o',      # circle
343     'kt_opt': 's',      # square
344     'I_opt': '^',       # triangle up
345     'E_opt': 'D',       # diamond

```

```

345     'EI_opt': 'v'          # plus (filled)
346 }
347
348 # Define marker sizes for each column
349 marker_sizes = {
350     'kr_opt': 50,          # Adjust marker size as needed
351     'kt_opt': 43,
352     'I_opt': 60,
353     'E_opt': 35,
354     'EI_opt': 60
355 }
356
357 # Set up the plot
358 plt.figure(figsize=(9, 6))
359
360 # Create scatter plot for each column with thin black edges
361 for col in columns:
362     plt.scatter(
363         [col] * len(test2_x),
364         test2_x[col],
365         label=col,
366         color=colors[col],
367         marker=markers[col],
368         edgecolor='black',
369         linewidth=0.5,
370         s=marker_sizes[col],
371         zorder=2
372     )
373
374 # Connect the dots for each row
375 for i in range(len(test2_x)):
376     plt.plot(columns, test2_x.iloc[i][columns], color='dimgrey', linestyle='-', linewidth
              =0.5, zorder=1)
377
378 # Add labels and title
379 plt.xlabel('Optimized updating parameters in x-direction', fontsize=12)
380 plt.ylabel('Scalar (-)', fontsize=12)
381 plt.xticks(rotation=45)
382
383 # Customize grid
384 plt.grid(True, which='major', axis='x', linestyle='-', linewidth=0.5, color='black') # Major
              vertical grid lines
385
386 # Hide major horizontal grid lines
387 plt.grid(True, which='major', axis='y', linestyle='-', linewidth=0.5, color='white')
388
389 # Add ticks where vertical grid lines intersect horizontal lines
390 for col in columns:
391     y_vals = [0.1, 1, 10]
392     for y_val in y_vals:
393         plt.scatter(col, y_val, color='black', marker='_', linewidths=1.7, s=40, zorder=0)
394
395 for col in columns:
396     y_vals = [0.2,0.3,0.4, 0.5,0.6,0.7,0.8,0.9, 2,3,4,5,6,7,8,9]
397     for y_val in y_vals:
398         plt.scatter(col, y_val, color='black', marker='_', linewidths=0.5, s=2, zorder=0)
399
400 # Adjust spines
401 plt.gca().spines['top'].set_linewidth(0.5)
402 plt.gca().spines['top'].set_color('white')
403 plt.gca().spines['right'].set_linewidth(0.5)
404 plt.gca().spines['right'].set_color('white')
405 plt.gca().spines['bottom'].set_linewidth(0.5)
406 plt.gca().spines['bottom'].set_color('white')
407 plt.gca().spines['left'].set_linewidth(0.5)
408 plt.gca().spines['left'].set_color('black')
409 plt.gca().yaxis.set_ticks_position('left') # Ensure ticks are on the left side of the plot
410 plt.tick_params(axis='y', which='major', direction='out', length=5, width=1, colors='black')
              # Customize tick parameters
411 # Set y-scale and limits
412 plt.yscale('log')

```

```

413 plt.ylim(0.1, 10)
414 plt.gca().set_yticks([0.1, 1, 10])
415 plt.gca().set_yticklabels(['0.1', '1', '10'])
416 #plt.tick_params(axis='y', which='major', direction='inout', length=6, width=1, colors='black')
417 #plt.gca().yaxis.grid(True, which='minor', linestyle='-', linewidth=0.25)
418 # Show legend
419 plt.legend(columns, bbox_to_anchor=(0.05, 1), loc='upper_left', prop={'family': 'Arial', 'size': 10})
420
421 # Display the plot
422 plt.tight_layout()
423 plt.show()
424
425 # Dimension Parameters:
426 L = 160.58 #[m]
427 Lt = np.array([5.735, 11.47, 26.640, 89.91, 26.825]) #[m]
428 B = 28 #[m]
429 D = 28 #[m]
430 A = B * D #[m2]
431 I_x = np.array([619.21, 948.47, 2727.22, 2335.82, 905.52]) #[m4]
432 I_y = np.array([1536.25, 959.01, 1245.46, 1202.07, 831.57]) #[m4]
433 I_yy = np.array([1444, 1500, 1393, 1389, 1293]) #[m4]
434 I_xx = np.array([1532, 1548, 2796, 2324, 760]) #[m4]
435 H = Lt/L #[-]
436
437 # Material properties:
438 rho = np.array([1933, 495, 486, 440, 383]) #[kg/m3]
439 E = np.array([38.2e9, 38.2e9, 38.2e9, 38.2e9, 32.8e9]) #[N/m2]
440 v = 0.2 #[-]
441 G = E/(2*(1+v)) #[N/m^2]
442 k = 0.85 #[-]
443
444 # Springs:
445 kr_x = 2380e9 #[Nm/rad]
446 kr_y = 2625e9 #[Nm/rad]
447 kt_y = 19e9 #[N/m]
448 kt_x = 4e9 #[N/m]
449
450 #boundary conditions:
451 bc = ("rot+transl", "free")
452
453 scaler = {"kr":0.4, "kt":9.0, "I":1.5, "E":7.0} #The parameters chosen are from the
         conclusion of sensitivity study 2
454 zz_trgt = np.array([57.165, 120.435, 153.735]) #The three locations where there are
         measurements taken of the New Orleans
455 z_trgt = zz_trgt/L
456
457 #Startpoints
458 startpoints = 400
459
460 fny = np.array([0.291, 1.332])
461 phiyy = [
462     [-0.29, -0.90],
463     [-0.78, 0.19],
464     [-1, 1]
465 ]
466 phiy = np.array(phiyy)
467
468 # Procedure
469
470 def adjust_modes(phi1, phi2):
471     rmse_plus = np.sqrt(np.mean((phi1 - phi2) ** 2))
472     rmse_minus = np.sqrt(np.mean((phi1 + phi2) ** 2))
473     flip = np.ones_like(phi2)
474     flip[rmse_plus > rmse_minus] = -1
475     phi_flip = phi1 * flip
476     return phi_flip
477
478 def modal_assurance_criterion(phi1, phi2):
479     phi2 = adjust_modes(phi2, phi1)

```

```

480     mac = np.abs(np.conj(phi1).dot(phi2))**2 / \
481         (np.conj(phi1).dot(phi1) * np.conj(phi2).dot(phi2))
482     return mac
483
484 timoshenko_y = []
485 for iy in range(len(H)):
486     timoshenko_y.append(Timo(params={"E": E[iy], "I": I_yy[iy], "L": L, "rho": rho[iy], "A": A
487         , "kG": k * G[iy], "kr": kr_y, "kt": kt_y}, height=H[iy]))
488 model_y = TimoPieceWiseBeam1D(beams=timoshenko_y, bc = bc)
489
490 y_hat_y = {"z": z_trgt, "fn": deepcopy(fny), "modes": deepcopy(phiy)}
491 parameters_to_update_y = {p: [1/10, 10] for p in list(scaler.keys())}
492 logscale = False
493
494 results_y = update_model(
495     model=model_y,
496     cost_func=cost_func1D,
497     parameters_to_update=parameters_to_update_y,
498     y_hat=y_hat_y,
499     npts=startpoints,
500     sim_type=UpdatingMethod.CMC,
501     verbose=False,
502     logscale=logscale
503 )
504 parameters_columns = [cc for cc in results_y.columns if "_st" in cc or "_opt" in cc]
505 plot_cols = [a + "_opt" for a in parameters_to_update_y.keys()]
506 map_opt2name = {pcl: k for k, pcl in zip(parameters_to_update_y.keys(), plot_cols)}
507 df_opt = results_y.copy()
508 df_opt.dropna(inplace=True)
509
510 if logscale:
511     df_opt[parameters_columns] = df_opt[parameters_columns].apply(np.exp)
512
513 # Results
514
515 t_y = results_y.sort_values(by='Convergence').head(startpoints)
516
517 quantile_threshold = t_y['Convergence'].quantile(0.125)
518 test_y = t_y[t_y['Convergence'] <= quantile_threshold]
519 print(test_y)
520
521 # Find the minimum and maximum values in each _opt column
522 min_values = test_y[[col for col in test_y.columns if col.endswith('_opt')]].min()
523 max_values = test_y[[col for col in test_y.columns if col.endswith('_opt')]].max()
524
525 # Update the boundaries
526 new_boundaries = {
527     'kr': [min_values['kr_opt'], max_values['kr_opt']],
528     'kt': [min_values['kt_opt'], max_values['kt_opt']],
529     'I': [min_values['I_opt'], max_values['I_opt']],
530     'E': [min_values['E_opt'], max_values['E_opt']]
531 }
532 print("New_Boundaries:")
533 print(new_boundaries)
534
535 results_repeated2_y = update_model(
536     model=model_y,
537     cost_func=cost_func1D,
538     parameters_to_update=new_boundaries,
539     y_hat=y_hat_y,
540     npts=startpoints,
541     sim_type=UpdatingMethod.CMC,
542     verbose=False,
543     logscale=logscale
544 )
545 parameters_columns = [cc for cc in results_repeated2_y.columns if "_st" in cc or "_opt" in cc]
546 plot_cols = [a + "_opt" for a in new_boundaries.keys()]
547 map_opt2name = {pcl: k for k, pcl in zip(new_boundaries.keys(), plot_cols)}
548 df_opt = results_repeated2_y.copy()
549 df_opt.dropna(inplace=True)

```

```

549
550 if logscale:
551     df_opt[parameters_columns] = df_opt[parameters_columns].apply(np.exp)
552
553 t2_y = results_repeated2_y.sort_values(by='Convergence').head(startpoints)
554 print(t2_y)
555
556 quantile_threshold = t2_y['Convergence'].quantile(0.125)
557 test2_y = t2_y[t2_y['Convergence'] <= quantile_threshold]
558 print(test2_y)
559
560 results_output_y = []
561 for ii in range(len(test2_y)):
562     scaler_y = {
563         "kr": test2_y["kr_opt"].iloc[ii],
564         "kt": test2_y["kt_opt"].iloc[ii],
565         "I": test2_y["I_opt"].iloc[ii],
566         "E": test2_y["E_opt"].iloc[ii]
567     }
568     model_y.scale_parms(scalers=scaler_y)
569     fn_y, phi_y, pts_y = model_y.modeshapes(pts=z_trgt, n_modes=2)
570
571     mac_0 = modal_assurance_criterion(phi_y[:, 0], phi_y[:, 0])
572     mac_1 = modal_assurance_criterion(phi_y[:, 1], phi_y[:, 1])
573
574     results_output_y.append({
575         "index": ii,
576         "fn_y_1": fn_y[0],
577         "fn_y_2": fn_y[1],
578         "mac_1": mac_0,
579         "mac_2": mac_1,
580     })
581
582 results_df = pd.DataFrame(results_output_y)
583 stats_results = {}
584 z_value = 1.645
585
586 for col in ['fn_y_1', 'fn_y_2', 'mac_1', 'mac_2']:
587     mean_val = results_df[col].mean()
588     median_val = results_df[col].median()
589     max_val = results_df[col].max()
590     min_val = results_df[col].min()
591
592     stats_results[col] = {
593         "Median": median_val,
594         "Mean": mean_val,
595         "lower_bound": min_val,
596         "upper_bound": max_val
597     }
598
599 summary_df = pd.DataFrame(columns=["Measured_fny", "Median", "Mean", "Lower_Bound", "Upper_
600 Bound"])
601 for param, values in stats_results.items():
602     if "fn_y_1" in param:
603         measured_fny = fny[0]
604     elif "fn_y_2" in param:
605         measured_fny = fny[1]
606     elif "fn_y_3" in param:
607         measured_fny = fny[2]
608     else:
609         measured_fny = np.nan
610
611     summary_df.loc[param] = [measured_fny, values["Median"], values["Mean"], values["
612 lower_bound"], values["upper_bound"]]
613 print(summary_df)
614
615 def calculate_summary_stats(df, columns, direction):
616     print(f"{direction}-direction")
617
618     kr_summary_stats = []
619     kt_summary_stats = []

```

```

618
619     for column in columns:
620         if "kr_opt" in column:
621             mean_val = df[column].mean()
622             median_val = df[column].median()
623             sem_val = df[column].sem()
624             std_dev = df[column].std()
625             z_value = 1.645
626
627             lower_boundkr = df[column].min()
628             upper_boundkr = df[column].max()
629
630             cv = round((std_dev / mean_val) * 100, 2)
631
632             kr_summary_stats.append(pd.DataFrame({
633                 'Design': 1,
634                 'Median': median_val,
635                 'Lower_Bound': lower_boundkr,
636                 'Upper_Bound': upper_boundkr,
637                 'COV(%)': cv
638             }, index=["kr_opt"]))
639
640         elif "kt_opt" in column:
641             mean_val = df[column].mean()
642             median_val = df[column].median()
643             sem_val = df[column].sem()
644             std_dev = df[column].std()
645             z_value = 1.645
646
647             lower_boundkt = df[column].min()
648             upper_boundkt = df[column].max()
649
650             cv = round((std_dev / mean_val) * 100, 2)
651
652             kt_summary_stats.append(pd.DataFrame({
653                 'Design': 1,
654                 'Median': median_val,
655                 'Lower_Bound': lower_boundkt,
656                 'Upper_Bound': upper_boundkt,
657                 'COV(%)': cv
658             }, index=["kt_opt"]))
659
660     kr_summary_stats_df = pd.concat(kr_summary_stats, axis=0)
661     kt_summary_stats_df = pd.concat(kt_summary_stats, axis=0)
662
663     print("Summary statistics for boundary conditions:")
664     print(kr_summary_stats_df)
665     print(kt_summary_stats_df.to_string(header=None))
666
667     for ii in range(len(E)):
668         print("")
669         print(f"Summary statistics for discrete element {ii}")
670         summary_stats_list = []
671
672         for column in columns:
673             if "kr_opt" in column or "kt_opt" in column:
674                 continue
675
676             mean_val = df[column].mean()
677             median_val = df[column].median()
678             sem_val = df[column].sem()
679             std_dev = df[column].std()
680             z_value = 1.645
681
682             lower_bound = df[column].min()
683             upper_bound = df[column].max()
684
685             cv = round((std_dev / mean_val) * 100, 2)
686
687             if 'y' in direction:
688                 if "E_opt" in column:

```

```

689         summary_stats = pd.DataFrame({
690             'Design': 1,
691             'Median': median_val,
692             'Lower_Bound': lower_bound ,
693             'Upper_Bound': upper_bound,
694             'COV(%)': cv
695         }, index=["E_opt"])
696         summary_stats_list.append(summary_stats)
697     elif "I_opt" in column:
698         summary_stats = pd.DataFrame({
699             'Design': 1,
700             'Median': median_val,
701             'Lower_Bound': lower_bound ,
702             'Upper_Bound': upper_bound,
703             'COV(%)': cv
704         }, index=["I_opt"])
705         summary_stats_list.append(summary_stats)
706         new_column = df["E_opt"] * df["I_opt"]
707         mean_val_new = new_column.mean()
708         median_val_new = new_column.median()
709         std_dev_new = new_column.std()
710
711         lower_bound_new = new_column.min()
712         upper_bound_new = new_column.max()
713
714         cv_new = round((std_dev_new / mean_val_new) * 100, 2)
715
716         summary_stats = pd.DataFrame({
717             'Design': 1,
718             'Median': median_val_new,
719             'Lower_Bound': lower_bound_new ,
720             'Upper_Bound': upper_bound_new,
721             'COV(%)': cv_new
722         }, index=["EI_opt"])
723         summary_stats_list.append(summary_stats)
724
725     summary_stats_df = pd.concat(summary_stats_list, axis=0)
726     print(summary_stats_df)
727
728 parameters_columns_y = [cc for cc in test2_y.columns if "_opt" in cc]
729 summary_stats_y = calculate_summary_stats(test2_y, parameters_columns_y, "y")
730
731 import matplotlib.pyplot as plt
732
733 # Assuming test2_y is already defined and has the columns 'kr_opt', 'kt_opt', 'I_opt', 'E_opt',
734 # and 'EI_opt'
735
736 # Add a new column 'EI_opt' which is the product of 'E_opt' and 'I_opt'
737 test2_y['EI_opt'] = test2_y['E_opt'] * test2_y['I_opt']
738
739 # Columns to plot
740 columns = ['kr_opt', 'kt_opt', 'E_opt', 'I_opt', 'EI_opt']
741
742 # Define colors for each column
743 colors = {
744     'kr_opt': 'steelblue',
745     'kt_opt': 'lightsteelblue',
746     'I_opt': 'firebrick',
747     'E_opt': 'goldenrod',
748     'EI_opt': 'white'
749 }
750
751 # Define marker styles for each column
752 markers = {
753     'kr_opt': 'o',      # circle
754     'kt_opt': 's',      # square
755     'I_opt': '^',       # triangle up
756     'E_opt': 'D',       # diamond
757     'EI_opt': 'v'       # plus (filled)
758 }

```



```

759 # Define marker sizes for each column
760 marker_sizes = {
761     'kr_opt': 50,          # Adjust marker size as needed
762     'kt_opt': 43,
763     'I_opt': 60,
764     'E_opt': 35,
765     'EI_opt': 60
766 }
767
768 # Set up the plot
769 plt.figure(figsize=(9, 6))
770
771 # Create scatter plot for each column with thin black edges
772 for col in columns:
773     plt.scatter(
774         [col] * len(test2_y),
775         test2_y[col],
776         label=col,
777         color=colors[col],
778         marker=markers[col],
779         edgecolor='black',
780         linewidth=0.5,
781         s=marker_sizes[col],
782         zorder=2
783     )
784
785 # Connect the dots for each row
786 for i in range(len(test2_y)):
787     plt.plot(columns, test2_y.iloc[i][columns], color='dimgrey', linestyle='-', linewidth
788              =0.5, zorder=1)
789
790 # Add labels and title
791 plt.xlabel('Optimized updating parameters in y-direction', fontsize=12)
792 plt.ylabel('Scalar (-)', fontsize=12)
793 plt.xticks(rotation=45)
794
795 # Customize grid
796 plt.grid(True, which='major', axis='x', linestyle='-', linewidth=0.5, color='black') # Major
797     vertical grid lines
798
799 # Hide major horizontal grid lines
800 plt.grid(True, which='major', axis='y', linestyle='-', linewidth=0.5, color='white')
801
802 # Add ticks where vertical grid lines intersect horizontal lines
803 for col in columns:
804     y_vals = [0.1, 1, 10]
805     for y_val in y_vals:
806         plt.scatter(col, y_val, color='black', marker='_', linewidths=1.7, s=40, zorder=0)
807
808 for col in columns:
809     y_vals = [0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 2, 3, 4, 5, 6, 7, 8, 9]
810     for y_val in y_vals:
811         plt.scatter(col, y_val, color='black', marker='_', linewidths=0.5, s=2, zorder=0)
812
813 # Adjust spines
814 plt.gca().spines['top'].set_linewidth(0.5)
815 plt.gca().spines['top'].set_color('white')
816 plt.gca().spines['right'].set_linewidth(0.5)
817 plt.gca().spines['right'].set_color('white')
818 plt.gca().spines['bottom'].set_linewidth(0.5)
819 plt.gca().spines['bottom'].set_color('white')
820 plt.gca().spines['left'].set_linewidth(0.5)
821 plt.gca().spines['left'].set_color('black')
822 plt.gca().yaxis.set_ticks_position('left') # Ensure ticks are on the left side of the plot
823 plt.tick_params(axis='y', which='major', direction='out', length=5, width=1, colors='black')
824     # Customize tick parameters
825
826 # Set y-scale and limits
827 plt.yscale('log')
828 plt.ylim(0.1, 10)
829 plt.gca().set_yticks([0.1, 1, 10])
830 plt.gca().set_yticklabels(['0.1', '1', '10'])

```

```

827 plt.tick_params(axis='y', which='major', direction='inout', length=6, width=1, colors='black
      ')
828 plt.gca().yaxis.grid(True, which='minor', linestyle='--', linewidth=0.25)
829 # Show legend
830 plt.legend(columns, bbox_to_anchor=(0.05, 1), loc='upper_left', prop={'family': 'Arial', '
      size': 10})
831
832 # Display the plot
833 plt.tight_layout()
834 plt.show()
835
836 # Dimension Parameters:
837 L = 160.58 #[m]
838 Lt = np.array([5.735, 11.47, 26.640, 89.91, 26.825]) #[m]
839 B = 28 #[m]
840 D = 28 #[m]
841 A = B * D #[m2]
842 I_x = np.array([619.21, 948.47, 2727.22, 2335.82, 905.52]) #[m4]
843 I_y = np.array([1536.25, 959.01, 1245.46, 1202.07, 831.57]) #[m4]
844 I_yy = np.array([1444, 1500, 1393, 1389, 1293]) #[m4]
845 I_xx = np.array([1532, 1548, 2796, 2324, 760]) #[m4]
846 H = Lt/L #[-]
847
848 # Material properties:
849 rho = np.array([1933, 495, 486, 440, 383]) #[kg/m3]
850 E = np.array([38.2e9, 38.2e9, 38.2e9, 38.2e9, 32.8e9]) #[N/m2]
851 v = 0.2 #[-]
852 G = E/(2*(1+v)) #[N/m^2]
853 k = 0.85 #[-]
854
855 # Springs:
856 kr_x = 2380e9 #[Nm/rad]
857 kr_y = 2625e9 #[Nm/rad]
858 kt_y = 19e9 #[N/m]
859 kt_x = 4e9 #[N/m]
860
861 #boundary conditions:
862 bc = ("rot+transl", "free")
863
864 scaler = {"kr":0.4, "kt":9.0, "I":1.5, "E":7.0} #The parameters chosen are from the
      conclusion of sensitivity study 2
865 zz_trgt = np.array([57.165, 120.435, 153.735]) #The three locations where there are
      measurements taken of the New Orleans
866 z_trgt = zz_trgt/L
867
868 #Startpoints
869 startpoints = 400
870
871 fny = np.array([0.291, 1.332, 2.771])
872 phiyy = [
873     [-0.29, -0.90, -0.73],
874     [-0.78, 0.19, 0.83],
875     [-1, 1, -1]
876 ]
877 phiy = np.array(phiyy)
878
879 # Procedure
880
881 def adjust_modes(phi1, phi2):
882     rmse_plus = np.sqrt(np.mean((phi1 - phi2) ** 2))
883     rmse_minus = np.sqrt(np.mean((phi1 + phi2) ** 2))
884     flip = np.ones_like(phi2)
885     flip[rmse_plus > rmse_minus] = -1
886     phi_flip = phi1 * flip
887     return phi_flip
888
889 def modal_assurance_criterion(phi1, phi2):
890     phi2 = adjust_modes(phi2, phi1)
891     mac = np.abs(np.conj(phi1).dot(phi2))**2 / \
892         (np.conj(phi1).dot(phi1) * np.conj(phi2).dot(phi2))
893     return mac

```

```

894
895 timoshenko_y = []
896 for iy in range(len(H)):
897     timoshenko_y.append(Timo(params={"E": E[iy], "I": I_yy[iy], "L": L, "rho": rho[iy], "A": A
898     , "kG": k * G[iy], "kr": kr_y, "kt": kt_y}, height=H[iy]))
899 model_y = TimoPieceWiseBeam1D(beams=timoshenko_y, bc = bc)
900
901 y_hat_y = {"z": z_trgt, "fn": deepcopy(fny), "modes": deepcopy(phiy)}
902 parameters_to_update_y = {p: [1/10, 10] for p in list(scaler.keys())}
903 logscale = False
904
905 results_y = update_model(
906     model=model_y,
907     cost_func=cost_func1D,
908     parameters_to_update=parameters_to_update_y,
909     y_hat=y_hat_y,
910     npts=startpoints,
911     sim_type=UpdatingMethod.CMC,
912     verbose=False,
913     logscale=logscale
914 )
915 parameters_columns = [cc for cc in results_y.columns if "_st" in cc or "_opt" in cc]
916 plot_cols = [a + "_opt" for a in parameters_to_update_y.keys()]
917 map_opt2name = {pcl: k for k, pcl in zip(parameters_to_update_y.keys(), plot_cols)}
918 df_opt = results_y.copy()
919 df_opt.dropna(inplace=True)
920
921 if logscale:
922     df_opt[parameters_columns] = df_opt[parameters_columns].apply(np.exp)
923
924 # Results
925
926 t_y = results_y.sort_values(by='Convergence').head(startpoints)
927
928 quantile_threshold = t_y['Convergence'].quantile(0.125)
929 test_y = t_y[t_y['Convergence'] <= quantile_threshold]
930 print(test_y)
931
932 # Find the minimum and maximum values in each _opt column
933 min_values = test_y[[col for col in test_y.columns if col.endswith('_opt')]].min()
934 max_values = test_y[[col for col in test_y.columns if col.endswith('_opt')]].max()
935
936 # Update the boundaries
937 new_boundaries = {
938     'kr': [min_values['kr_opt'], max_values['kr_opt']],
939     'kt': [min_values['kt_opt'], max_values['kt_opt']],
940     'I': [min_values['I_opt'], max_values['I_opt']],
941     'E': [min_values['E_opt'], max_values['E_opt']]
942 }
943 print("New Boundaries:")
944 print(new_boundaries)
945
946 results_repeated2_y = update_model(
947     model=model_y,
948     cost_func=cost_func1D,
949     parameters_to_update=new_boundaries,
950     y_hat=y_hat_y,
951     npts=startpoints,
952     sim_type=UpdatingMethod.CMC,
953     verbose=False,
954     logscale=logscale
955 )
956 parameters_columns = [cc for cc in results_repeated2_y.columns if "_st" in cc or "_opt" in cc]
957 plot_cols = [a + "_opt" for a in new_boundaries.keys()]
958 map_opt2name = {pcl: k for k, pcl in zip(new_boundaries.keys(), plot_cols)}
959 df_opt = results_repeated2_y.copy()
960 df_opt.dropna(inplace=True)
961
962 if logscale:
963     df_opt[parameters_columns] = df_opt[parameters_columns].apply(np.exp)

```

```

963
964 t2_y = results_repeated2_y.sort_values(by='Convergence').head(startpoints)
965 print(t2_y)
966
967 quantile_threshold = t2_y['Convergence'].quantile(0.125)
968 test2_y = t2_y[t2_y['Convergence'] <= quantile_threshold]
969 print(test2_y)
970
971 results_output_y = []
972 for ii in range(len(test2_y)):
973     scaler_y = {
974         "kr": test2_y["kr_opt"].iloc[ii],
975         "kt": test2_y["kt_opt"].iloc[ii],
976         "I": test2_y["I_opt"].iloc[ii],
977         "E": test2_y["E_opt"].iloc[ii]
978     }
979     model_y.scale_parms(scalers=scaler_y)
980     fn_y, phi_y, pts_y = model_y.modeshapes(pts=z_trgt, n_modes=3)
981
982     mac_0 = modal_assurance_criterion(phi_y[:, 0], phi_y[:, 0])
983     mac_1 = modal_assurance_criterion(phi_y[:, 1], phi_y[:, 1])
984     mac_2 = modal_assurance_criterion(phi_y[:, 2], phi_y[:, 2])
985
986     results_output_y.append({
987         "index": ii,
988         "fn_y_1": fn_y[0],
989         "fn_y_2": fn_y[1],
990         "fn_y_3": fn_y[2],
991         "mac_1": mac_0,
992         "mac_2": mac_1,
993         "mac_3": mac_2
994     })
995
996 results_df = pd.DataFrame(results_output_y)
997 stats_results = {}
998 z_value = 1.645
999
1000 for col in ['fn_y_1', 'fn_y_2', 'fn_y_3', 'mac_1', 'mac_2', 'mac_3']:
1001     mean_val = results_df[col].mean()
1002     median_val = results_df[col].median()
1003     max_val = results_df[col].max()
1004     min_val = results_df[col].min()
1005
1006     stats_results[col] = {
1007         "Median": median_val,
1008         "Mean": mean_val,
1009         "lower_bound": min_val,
1010         "upper_bound": max_val
1011     }
1012
1013 summary_df = pd.DataFrame(columns=["Measured_fny", "Median", "Mean", "Lower_Bound", "Upper_
    Bound"])
1014 for param, values in stats_results.items():
1015     if "fn_y_1" in param:
1016         measured_fny = fny[0]
1017     elif "fn_y_2" in param:
1018         measured_fny = fny[1]
1019     elif "fn_y_3" in param:
1020         measured_fny = fny[2]
1021     else:
1022         measured_fny = np.nan
1023
1024     summary_df.loc[param] = [measured_fny, values["Median"], values["Mean"], values["
        lower_bound"], values["upper_bound"]]
1025 print(summary_df)
1026
1027 def calculate_summary_stats(df, columns, direction):
1028     print(f"{direction}-direction")
1029
1030     kr_summary_stats = []
1031     kt_summary_stats = []

```

```

1032
1033     for column in columns:
1034         if "kr_opt" in column:
1035             mean_val = df[column].mean()
1036             median_val = df[column].median()
1037             sem_val = df[column].sem()
1038             std_dev = df[column].std()
1039             z_value = 1.645
1040
1041             lower_boundkr = df[column].min()
1042             upper_boundkr = df[column].max()
1043
1044             cv = round((std_dev / mean_val) * 100, 2)
1045
1046             kr_summary_stats.append(pd.DataFrame({
1047                 'Design': 1,
1048                 'Median': median_val,
1049                 'Lower_Bound': lower_boundkr,
1050                 'Upper_Bound': upper_boundkr,
1051                 'COV(%)': cv
1052             }, index=["kr_opt"]))
1053
1054         elif "kt_opt" in column:
1055             mean_val = df[column].mean()
1056             median_val = df[column].median()
1057             sem_val = df[column].sem()
1058             std_dev = df[column].std()
1059             z_value = 1.645
1060
1061             lower_boundkt = df[column].min()
1062             upper_boundkt = df[column].max()
1063
1064             cv = round((std_dev / mean_val) * 100, 2)
1065
1066             kt_summary_stats.append(pd.DataFrame({
1067                 'Design': 1,
1068                 'Median': median_val,
1069                 'Lower_Bound': lower_boundkt,
1070                 'Upper_Bound': upper_boundkt,
1071                 'COV(%)': cv
1072             }, index=["kt_opt"]))
1073
1074     kr_summary_stats_df = pd.concat(kr_summary_stats, axis=0)
1075     kt_summary_stats_df = pd.concat(kt_summary_stats, axis=0)
1076
1077     print("Summary statistics for boundary conditions:")
1078     print(kr_summary_stats_df)
1079     print(kt_summary_stats_df.to_string(header=None))
1080
1081     for ii in range(len(E)):
1082         print("")
1083         print(f"Summary statistics for discrete element {ii}")
1084         summary_stats_list = []
1085
1086         for column in columns:
1087             if "kr_opt" in column or "kt_opt" in column:
1088                 continue
1089
1090             mean_val = df[column].mean()
1091             median_val = df[column].median()
1092             sem_val = df[column].sem()
1093             std_dev = df[column].std()
1094             z_value = 1.645
1095
1096             lower_bound = df[column].min()
1097             upper_bound = df[column].max()
1098
1099             cv = round((std_dev / mean_val) * 100, 2)
1100
1101             if 'y' in direction:
1102                 if "E_opt" in column:

```

```

1103         summary_stats = pd.DataFrame({
1104             'Design': 1,
1105             'Median': median_val,
1106             'Lower_Bound': lower_bound,
1107             'Upper_Bound': upper_bound,
1108             'COV(%)': cv
1109         }, index=["E_opt"])
1110         summary_stats_list.append(summary_stats)
1111     elif "I_opt" in column:
1112         summary_stats = pd.DataFrame({
1113             'Design': 1,
1114             'Median': median_val,
1115             'Lower_Bound': lower_bound,
1116             'Upper_Bound': upper_bound,
1117             'COV(%)': cv
1118         }, index=["I_opt"])
1119         summary_stats_list.append(summary_stats)
1120         new_column = df["E_opt"] * df["I_opt"]
1121         mean_val_new = new_column.mean()
1122         median_val_new = new_column.median()
1123         std_dev_new = new_column.std()
1124
1125         lower_bound_new = new_column.min()
1126         upper_bound_new = new_column.max()
1127
1128         cv_new = round((std_dev_new / mean_val_new) * 100, 2)
1129
1130         summary_stats = pd.DataFrame({
1131             'Design': 1,
1132             'Median': median_val_new,
1133             'Lower_Bound': lower_bound_new,
1134             'Upper_Bound': upper_bound_new,
1135             'COV(%)': cv_new
1136         }, index=["EI_opt"])
1137         summary_stats_list.append(summary_stats)
1138
1139     summary_stats_df = pd.concat(summary_stats_list, axis=0)
1140     print(summary_stats_df)
1141
1142 parameters_columns_y = [cc for cc in test2_y.columns if "_opt" in cc]
1143 summary_stats_y = calculate_summary_stats(test2_y, parameters_columns_y, "y")

```

## Appendix Timoshenko beam model

```

1 %load_ext autoreload
2 %autoreload 2
3 from copy import deepcopy
4 from pyvibe.models.Timoshenko import Timo, TimoPieceWiseBeam1D, TimoPieceWiseBeam2D,
   UpdatingMethod, cost_func1D, update_model
5 import numpy as np
6 import matplotlib.pyplot as plt
7
8 %matplotlib inline
9 ## Parameters
10
11 # Dimension Parameters:
12 L = 2 #[m]
13 Lt = np.array([1,1]) #[m]
14 B = 0.1 #[m]
15 D = 0.1 #[m]
16
17 A = B * D #[m2]
18 I = 1/12 * B * D**3 #[m4]
19 H = Lt/L #[-]
20
21 # Material properties:
22 rho = np.array([8000,8000]) #[kg/m3]
23 E = np.array([260e9,260e9]) #[N/m2]
24 G = np.array([100e9,100e9]) #[N/m2]
25 k = 5/6 #[-]

```

```

26
27 #boudary conditions:
28 bc = ("hinged", "hinged")
29
30 ## Procedure
31
32 timoshenko = []
33 for ix in range(len(H)):
34     timoshenko.append(Timo(params = {"E": E[ix], "I": I, "L": L, "rho": rho[ix], "A": A, "kG"
35                                     : k*G[ix]},height= H[ix]))
36 pb = TimoPieceWiseBeam1D(beams=timoshenko, bc=bc)
37 fn, phi, pts = pb.modeshapes(max_fn=30000, n_intervals=10000, n_modes= 50)
38
39 ## Results
40
41 print("The first 50 natural frequencies (in rad/s) of the Timoshenko beam are:")
42 for ii in range(50):
43     print(f"Mode {ii+1}: ", fn[ii] * 2 * np.pi)
44
45 plt.figure(figsize=(6, 6))
46 plt.plot(pts, phi[:,0], color='blue', label='Mode_1')
47 plt.plot(pts, phi[:,1], '--', color='red', label='Mode_2')
48 plt.plot(pts, -phi[:,2], '-.', color='green', label='Mode_3')
49 plt.plot(pts, -phi[:,3], '--', color='magenta', label='Mode_4')
50 plt.plot(pts, phi[:,4], '-.', color='black', label='Mode_5')
51
52 plt.ylim(-2, 2)
53
54 plt.ylabel('Normalized displacement (-)')
55 plt.xlabel('Normalized length (-)')
56 #plt.title('The mode shape of mode 1,2,3,4,and 5')
57 plt.xlim(0,1)
58 plt.yticks([-2,-1.6, -1.2, -0.8, -0.4, 0, 0.4, 0.8, 1.2, 1.6, 2.0])
59 plt.xticks([0,0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0])
60 plt.tick_params(axis='y', which='both', direction='inout', length=6, width=1, colors='black')
61 plt.legend(frameon=True, loc='lower_left')
62 plt.grid(True)
63 plt.show()
64
65 plt.figure(figsize=(6, 6))
66 plt.plot(pts, -phi[:,25], color='blue', label='Mode_26')
67 plt.ylim(-0.5, 0.5)
68 plt.xlim(0,1)
69 plt.ylabel('Nodal displacement (-)')
70 plt.xlabel('Normalized length (-)')
71 #plt.title('The mode shape of mode 26 (at transition frequency $\\omega_c$)')
72 plt.yticks([-0.5,-0.4, -0.3, -0.2, -0.1, 0, 0.1, 0.2, 0.3, 0.4, 0.5])
73 plt.xticks([0,0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0])
74 plt.legend(frameon=True, loc='lower_left')
75 plt.grid(True)
76 plt.show()
77
78 plt.figure(figsize=(6, 6))
79 plt.plot(pts, phi[:,26], color='blue', label='Mode_27')
80 plt.plot(pts, phi[:,27], '--', color='red', label='Mode_28')
81 plt.plot(pts, -phi[:,29], '--', color='magenta', label='Mode_30')
82 plt.plot(pts, -phi[:,30], '-.', color='black', label='Mode_31')
83
84 plt.ylim(-2, 2)
85
86 plt.ylabel('Nodal displacement (-)')
87 plt.xlabel('Normalized length (-)')
88 #plt.title('The mode shape of mode 27, 28, 30 and 31')
89 plt.xlim(0,1)
90 plt.yticks([-2,-1.6, -1.2, -0.8, -0.4, 0, 0.4, 0.8, 1.2, 1.6, 2.0])
91 plt.xticks([0,0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0])
92 plt.legend(frameon=True, loc='lower_left')
93 plt.grid(True)
94 plt.show()
95

```

```

96 plt.figure(figsize=(6, 6))
97 plt.plot(pts, phi[:,28], color='blue', label='Mode_29')
98 plt.plot(pts, -phi[:,31], '--', color='red', label='Mode_32')
99
100 plt.ylim(-2, 2)
101 plt.xlim(0,1)
102 plt.yticks([-2,-1.6, -1.2, -0.8, -0.4, 0, 0.4, 0.8, 1.2, 1.6, 2.0])
103 plt.xticks([0,0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0])
104
105 plt.ylabel('Nodal displacement (-)')
106 plt.xlabel('Normalized length (-)')
107 plt.title('The mode shapes of modes 29 and 32')
108 plt.legend(frameon=True, loc='lower_left')
109 plt.tick_params(axis='y', which='major', direction='inout', length=6, width=1, colors='black'
110 )
110 plt.tick_params(axis='y', which='minor', direction='inout', length=4, width=0.5, colors='gray'
111 ')
111 plt.grid(True)
112 plt.show()
113
114 import matplotlib.pyplot as plt
115
116
117 # Plotting the data
118 plt.figure(figsize=(6, 6))
119 plt.plot(pts, phi[:,28], color='blue', label='Mode_29')
120 plt.plot(pts, -phi[:,31], '--', color='red', label='Mode_32')
121
122 plt.ylim(-2, 2)
123 plt.xlim(0, 1)
124 plt.yticks([-2, -1.6, -1.2, -0.8, -0.4, 0, 0.4, 0.8, 1.2, 1.6, 2.0])
125 plt.xticks([0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0])
126
127 plt.ylabel('Nodal displacement (-)')
128 plt.xlabel('Normalized length (-)')
129 plt.legend(frameon=True, loc='lower_left')
130 plt.grid(True)
131
132 # Customizing tick parameters for y-axis only
133 plt.tick_params(axis='y', which='major', direction='inout', length=6, width=1, colors='black'
134 )
134 plt.tick_params(axis='y', which='minor', direction='inout', length=4, width=0.5, colors='gray'
135 ')
135
136 plt.show()
137
138
139 import matplotlib.pyplot as plt
140 import numpy as np
141
142 # Generate some sample data
143 x = np.linspace(0, 2*np.pi, 100)
144 y = np.sin(x)
145
146 # Plotting the data
147 plt.figure(figsize=(8, 6))
148 plt.plot(x, y, color='blue', label='sin(x)')
149
150 plt.ylim(-1.2, 1.2)
151 plt.xlim(0, 2*np.pi)
152 plt.yticks(np.linspace(-1, 1, 5)) # Set y-axis ticks at specific intervals
153
154 plt.xlabel('x')
155 plt.ylabel('sin(x)')
156 plt.title('Sine Function')
157
158 plt.legend()
159
160 # Customizing y-axis tick parameters
161 plt.tick_params(axis='y', which='major', direction='inout', length=6, width=1, colors='black'
162 )

```



```

162 plt.show()
163
164 %load_ext autoreload
165 %autoreload 2
166 from copy import deepcopy
167 from pyvibe.models.Timoshenko_phi import Timo, TimoPieceWiseBeam1D, TimoPieceWiseBeam2D,
    UpdatingMethod, cost_func1D, update_model
168 import numpy as np
169 import matplotlib.pyplot as plt
170
171 %matplotlib inline
172 ## Parameters
173
174 # Dimension Parameters:
175 L = 2 #[m]
176 Lt = np.array([1,1]) #[m]
177 B = 0.1 #[m]
178 D = 0.1 #[m]
179
180 A = B * D #[m2]
181 I = 1/12 * B * D**3 #[m4]
182 H = Lt/L #[-]
183
184 # Material properties:
185 rho = np.array([8000,8000]) #[kg/m3]
186 E = np.array([260e9,260e9]) #[N/m2]
187 G = np.array([100e9,100e9]) #[N/m2]
188 k = 5/6 #[-]
189
190 #boundary conditions:
191 bc = ("hinged", "hinged")
192 ## Procedure
193
194 timoshenko = []
195 for ix in range(len(H)):
196     timoshenko.append(Timo(params = {"E": E[ix], "I": I, "L": L, "rho": rho[ix], "A": A, "kG"
        : k*G[ix]},height= H[ix]))
197 pb = TimoPieceWiseBeam1D(beams=timoshenko, bc=bc)
198 fn, phi, pts = pb.modeshapes(max_fn=30000, n_intervals=10000, n_modes= 50)
199 ## Results
200
201 plt.figure(figsize=(6, 6))
202 plt.plot(pts, phi[:,0], color='blue', label='Mode_1')
203 plt.plot(pts, -phi[:,1], '--', color='red', label='Mode_2')
204 plt.plot(pts, phi[:,2], '-.', color='green', label='Mode_3')
205 plt.plot(pts, -phi[:,3], '--', color='magenta', label='Mode_4')
206 plt.plot(pts, -phi[:,4], '-.', color='black', label='Mode_5')
207
208 #plt.ylim(-2, 2)
209 plt.ylabel('Normalized rotation (-)')
210 plt.xlabel('Normalized length (-)')
211 #plt.title('The mode shape of mode 1,2,3,4,and 5')
212 plt.xlim(0,1)
213 plt.yticks([-2,-1.6, -1.2, -0.8, -0.4, 0, 0.4, 0.8, 1.2, 1.6, 2.0])
214 plt.xticks([0,0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0])
215 plt.tick_params(axis='y', which='both', left=True, right=False, direction='out', length=8,
    width=0.5)
216 plt.tick_params(axis='x', which='both', bottom=True, top=False, direction='out', length=8,
    width=0.5)
217 plt.legend(frameon=True, loc='lower_left')
218 plt.grid(True)
219 plt.gca().spines['top'].set_linewidth(0.5)
220 plt.gca().spines['top'].set_color('black')
221 plt.gca().spines['right'].set_linewidth(0.5)
222 plt.gca().spines['right'].set_color('black')
223 plt.gca().spines['bottom'].set_linewidth(0.5)
224 plt.gca().spines['bottom'].set_color('black')
225 plt.gca().spines['left'].set_linewidth(0.5)
226 plt.gca().spines['left'].set_color('black')
227 plt.show()
228

```

```

229 plt.figure(figsize=(6, 6))
230 plt.plot(pts, phi[:,25], color='blue', label='Mode_26')
231 plt.xlim(0, 1)
232 plt.ylim(-1.2,1.2)
233 plt.ylabel('Normalized_rotation(-)')
234 plt.xlabel('Normalized_length(-)')
235 #plt.title('The mode shape of mode 26 (at transition frequency $\omega_c$)')
236 plt.yticks([-1.2,-1.0,-0.8,-0.6,-0.4,-0.2,0.0,0.2,0.4,0.6,0.8,1.0,1.2])
237 plt.xticks([0,0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0])
238 plt.tick_params(axis='y', which='both', left=True, right=False, direction='out', length=8,
239                 width=0.5)
239 plt.tick_params(axis='x', which='both', bottom=True, top=False, direction='out', length=8,
240                 width=0.5)
240 plt.gca().spines['top'].set_linewidth(0.5)
241 plt.gca().spines['top'].set_color('black')
242 plt.gca().spines['right'].set_linewidth(0.5)
243 plt.gca().spines['right'].set_color('black')
244 plt.gca().spines['bottom'].set_linewidth(0.5)
245 plt.gca().spines['bottom'].set_color('black')
246 plt.gca().spines['left'].set_linewidth(0.5)
247 plt.gca().spines['left'].set_color('black')
248 plt.legend(frameon=True, loc='lower_left')
249 plt.grid(True)
250 plt.show()
251
252 plt.figure(figsize=(6, 6))
253 plt.plot(pts, phi[:,26], color='blue', label='Mode_27')
254 plt.plot(pts, phi[:,27], '--', color='red', label='Mode_28')
255 plt.plot(pts, phi[:,29], '--', color='magenta', label='Mode_30')
256 plt.plot(pts, phi[:,30], '-.', color='black', label='Mode_31')
257
258 plt.ylim(-2, 2)
259
260 plt.ylabel('Normalized_rotation(-)')
261 plt.xlabel('Normalized_length(-)')
262 #plt.title('The mode shape of mode 27, 28, 30 and 31')
263 plt.xlim(0,1)
264 plt.yticks([-2,-1.6, -1.2, -0.8, -0.4, 0, 0.4, 0.8, 1.2, 1.6, 2.0])
265 plt.xticks([0,0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0])
266 plt.tick_params(axis='y', which='both', left=True, right=False, direction='out', length=8,
267                 width=0.5)
267 plt.tick_params(axis='x', which='both', bottom=True, top=False, direction='out', length=8,
268                 width=0.5)
268 plt.gca().spines['top'].set_linewidth(0.5)
269 plt.gca().spines['top'].set_color('black')
270 plt.gca().spines['right'].set_linewidth(0.5)
271 plt.gca().spines['right'].set_color('black')
272 plt.gca().spines['bottom'].set_linewidth(0.5)
273 plt.gca().spines['bottom'].set_color('black')
274 plt.gca().spines['left'].set_linewidth(0.5)
275 plt.gca().spines['left'].set_color('black')
276 plt.legend(frameon=True, loc='lower_left')
277 plt.grid(True)
278 plt.show()
279
280 plt.figure(figsize=(6, 6))
281 plt.plot(pts, -phi[:,28], color='blue', label='Mode_29')
282 plt.plot(pts, phi[:,31], '--', color='red', label='Mode_32')
283
284 plt.ylim(-2, 2)
285 plt.xlim(0,1)
286 plt.yticks([-2,-1.6, -1.2, -0.8, -0.4, 0, 0.4, 0.8, 1.2, 1.6, 2.0])
287 plt.xticks([0,0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0])
288 plt.tick_params(axis='y', which='both', left=True, right=False, direction='out', length=8,
289                 width=0.5)
289 plt.tick_params(axis='x', which='both', bottom=True, top=False, direction='out', length=8,
290                 width=0.5)
290 plt.gca().spines['top'].set_linewidth(0.5)
291 plt.gca().spines['top'].set_color('black')
292 plt.gca().spines['right'].set_linewidth(0.5)
293 plt.gca().spines['right'].set_color('black')

```

```
294 plt.gca().spines['bottom'].set_linewidth(0.5)
295 plt.gca().spines['bottom'].set_color('black')
296 plt.gca().spines['left'].set_linewidth(0.5)
297 plt.gca().spines['left'].set_color('black')
298 plt.ylabel('Normalized rotation (-)')
299 plt.xlabel('Normalized length (-)')
300 #plt.title('The mode shapes of modes 29 and 32')
301 plt.legend(frameon=True, loc='lower left')
302 plt.grid(True)
303 plt.show()
```