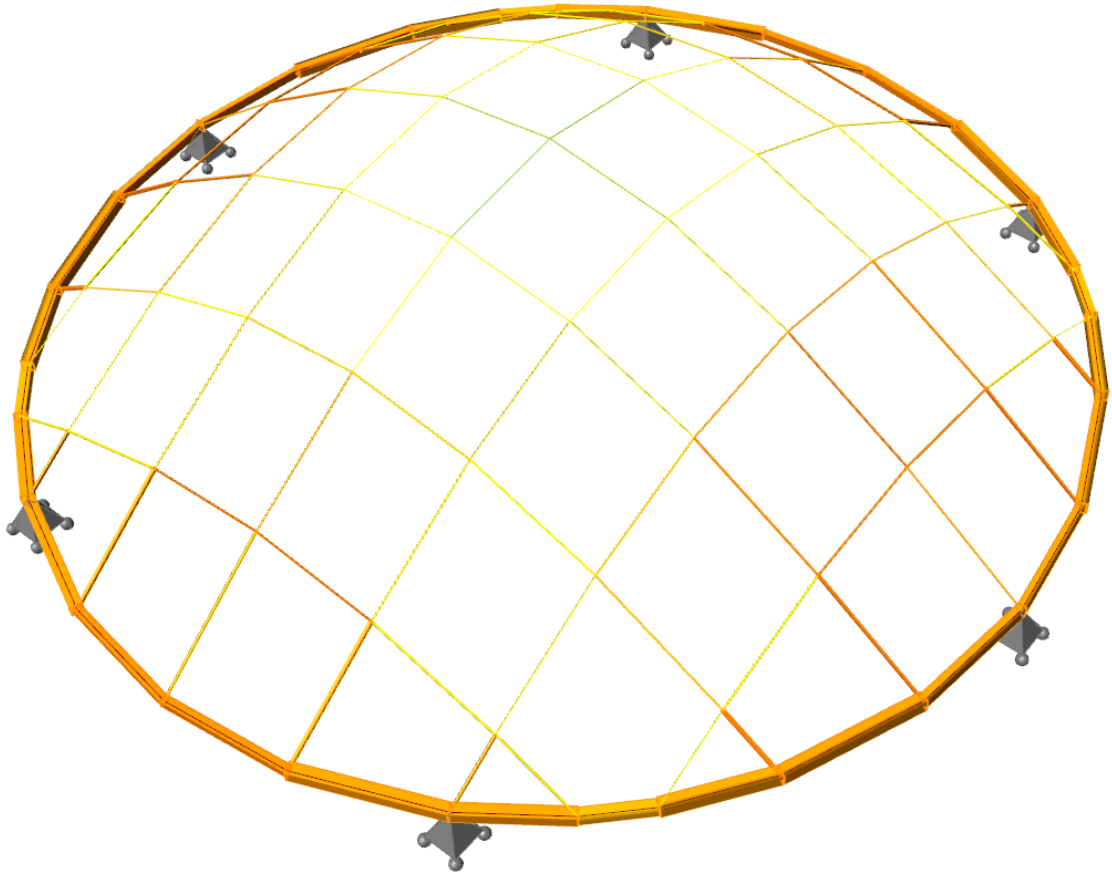
CHALMERS
UNIVERSITY OF TECHNOLOGY



Structural design optimisation by organisation of stiffness

Development of a finite element package for generative design

Master's Thesis in Structural Engineering and Building Technology

CARL HOFF
ISAK NÄSLUND

Department of Applied Mechanics
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2016

Master's thesis 2016:42

Structural design optimisation by organisation of stiffness

Development of a finite element package for generative design

CARL HOFF
ISAK NÄSLUND

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Applied Mechanics
Division of Material and Computational Mechanics
Chalmers University of Technology
Gothenburg, Sweden 2016

Structural design optimisation by organisation of stiffness
Development of a finite element package for generative design
CARL HOFF
ISAK NÄSLUND

Supervisor: Rasti Bartek, Buro Happold Engineering
Supervisor: Mats Ander, Applied Mechanics
Examiner: Mats Ander, Applied Mechanics

Master's Thesis 2016:42
Department of Applied Mechanics
Division of Material and Computational Mechanics
Chalmers University of Technology
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Cover: Results of combining optimisation methods on a dome structure, see section 6.3

Typeset in LaTeX
Printed by Chalmers Reproservice Gothenburg, Sweden 2016

Structural design optimisation by organisation of stiffness
Development of a finite element package for generative design
CARL HOFF & ISAK NÄSLUND
Department of Applied Mechanics
Chalmers University of Technology

Abstract

When a structure is designed, the designer has a choice to make their design in any number of ways. What drives the choices could be structural efficiency (oftentimes measured by weight), economy, architectural qualities, rationality, environmental impact or some other factor. Since these factors often can be hard to measure, especially at an early state of design when the level of uncertainty is high, a need for quick tools for generating acceptable designs is identified.

This thesis work has focused on developing a structural design tool that can be used for this purpose. It is implemented as a finite element package for the Rhinoceros plugin Grasshopper, a parametric design tool which also is well suited for the early stages of design. The package, called *CIGull*, contains a set of customisable optimisation methods for the generation of design, and includes simplified section checks. The methods were all focused on the redistribution of internal forces by organisation of stiffness. The methods included automated iterative section sizing, section alignment rotation, canonical stiffness (statical eigenmode) analysis and connection stiffness (discrete springs) redistribution. The methods were tested on a small set of test structures of varying complexity, as well as on a real project: KAFD Metro Station by Zaha Hadid Architects and Buro Happold Engineering.

The iterative section sizers were found to be quick, but they sometimes produced inefficient load paths. Improvements to these methods were suggested. The section alignment rotation method was shown to be an effective tool to increase structural efficiency, but questions were raised about its level of rationality. The connection stiffness redistribution method was shown to be able to improve structural efficiency, but automation of the method using genetic algorithms was shown to be very time consuming. It was possible to combine statical eigenmode analysis with an iterative section sizer to create new load patterns, which in some cases produced more effective structures. It was concluded that the best results were obtained by combining different methods.

The proposed methods were found to be sensitive to the case-specific input parameters. No one method was shown to produce better results in all cases. It could be concluded that the methods were best used as a means of understanding the structure. From the tested examples we conclude that the developed generative design tool is promising for the early conceptual design stages.

Keywords: *Structural design, Optimisation, Finite element method, FEM, Parametric design, Grasshopper*

Structural design optimisation by organisation of stiffness
Development of a finite element package for generative design
CARL HOFF & ISAK NÄSLUND
Avdelningen för Tillämpad Mekanik
Chalmers Tekniska Högskola

Sammanfattning

Vid utformningen av en struktur har ingenjören stor valmöjlighet i hur designen skall vara. Ett flertal faktorer kan påverka valet, t. ex. strukturell effektivitet (ofta mätt i vikt), ekonomiska faktorer, arkitektoniska kvalitéer, byggbarhet, miljöpåverkan eller någonting annat. Eftersom dessa faktorer kan vara svåra att mäta, särskilt i ett tidigt designskede, kan behovet för ett skissverktyg för denna uppgift identifieras.

I detta examensarbete har fokuset varit att utveckla ett designverktyg för detta syfte. Det har implementerats som en finitelementlösare till 3d-modelleringsprogrammet Rhinoceros plugin Grasshopper, som är ett parametriskt designverktyg väl lämpat för tidiga designskeden. Den framtagna pluginen *CIGull* innehåller även tvärsnittskontroller samt ett antal anpassningsbara optimeringsmetoder för generativ design, som alla syftar till att omfördela inre krafter genom att fördela styvhet. Dessa metoder inkluderar automatisk iterativ tvärsnittsuppdatering, tvärsnittsrotering, kanonisk styvhetsanalys (statisk egenmodsanalys) och omfördelning av inre krafter genom att förändra kopplingsstyvheter. Metoderna har testats på ett antal strukturer av skiftande komplexitet samt ett projekt i byggstadiet: KAFD Metro Station av Zaha Hadid Architects och Buro Happold Engineering.

De automatiska iterativa tvärsnittsuppdateringsalgoritmerna har visats vara snabba men ibland ineffektiva i de lösningar de tog fram. Förslag till förbättringar till dessa algoritmer gavs. Tvärsnittsroteringsmetoden visades vara effektiv för att öka den strukturella effektiviteten, men dess praktiska användbarhet ifrågasattes. En variation av styvheten i kopplingar visades kunna skapa mer effektiva lastvägar, men att automatisera en optimering av dessa styvheter m.h.a. genetiska algoritmer visades vara mycket tidskrävande. De statiska egenmoderna kunde kombineras med de automatiska iterativa tvärsnittsuppdateringsalgoritmerna för att i vissa fall kunna generera mer effektiva lösningar. De bästa resultaten uppnåddes genom att kombinera olika metoder.

Av de föreslagna metoderna visades ingen ge bäst resultat för samtliga situationer. Alla metoder visades vara känsliga för val av initiella parametrar. Slutsatsen kunde dras att metoderna bäst kunde användas för att läsa och förstå en struktur. Från de studerade exemplen dras slutsatsen att det utvecklade verktyget är lovande för användning i tidiga designskeden.

Nyckelord: *Strukturdesign, optimering, Finita elementmetoden, FEM, Parametrisk design, Grasshopper*

Acknowledgements

We would very much like to thank our supervisor and examiner Dr. Mats Ander for his encouragement and and feedback. We would also like to thank our supervisor Rasti Bartek of Buro Happold Engineers in London, for his many ideas and good advice.

We would also like to thank our professor Karl-Gunnar Olsson for always taking the time, and for the many books he has written. Finally we would like to thank our opponents Niclas Karlsson and Marcus Hjelm.

Carl Hoff and Isak Näslund, Gothenburg, June 2016

Nomenclature

The following section presents the symbols used throughout this thesis. Value symbols that have a unit are presented as SI units if nothing else is stated.

General

Greek upper case symbols

$\prod$	Product of a sequence (mathematical operator)	-

Greek lower case symbols

η	Utilisation	
θ	Element rotation	[rad]
λ	Eigenvalue	
λ_1	Lowest eigenvalue	
ρ	Density	[kg/m^3]
σ	Stress	[Pa]
φ	Element rotation around in local x axis	[rad]

Roman upper case symbols

U	Strain energy	[Nm]

Roman italic upper case symbols

E	Young's modulus	[Pa]
G	Shear modulus	[Pa]
I	Second moment of area	[m^4]
L	Element length	[m]
M	Bending moment	[Nm]
N	Normal force	[N]
V	Shear force	[N]

Roman italic lower case symbols

b	Section property width	[mm]
f_y	Yield stress	[Pa]
h	Section property height	[mm]
k	Spring stiffness	[N/m] / [Nm/rad]
m	Mass	[kg]
n	Quantity	[-]
t	Section property thickness	[mm]
u	Element displacement in local x direction	[m]
v	Element displacement in local y direction	[m]
w	Element displacement in local z direction	[m]

Roman bold upper case symbols
For some symbols used for matrix notation (for instance the stiffness matrix **K**) not all elements in the matrix has the same units. In these cases the term [var] is used.

K	Stiffness matrix	[var.]

Roman bold lower case symbols

a	Displacement vector	[m] / [rad]
f	Force vector	[N] / [Nm]

Contents

1 Introduction **1**
- 1.1 Background . . . 1
  - 1.1.1 Stiffness redistribution . . . 3
  - 1.1.2 Finite element solving . . . 7
  - 1.1.3 Canonical stiffnesses . . . 7
- 1.2 Aim . . . 8
- 1.3 Method . . . 8
- 1.4 Limitations . . . 10
  - 1.4.1 General . . . 10
  - 1.4.2 Section properties . . . 10
- 1.5 Outline of the report . . . 11

2 Theory **12**
- 2.1 Finite elements . . . 12
  - 2.1.1 Frame elements . . . 12
    - 2.1.1.1 Bar action . . . 12
    - 2.1.1.2 Beam action . . . 16
    - 2.1.1.3 Torsion . . . 22
    - 2.1.1.4 2D-Frame elements . . . 23
    - 2.1.1.5 3D-Frame elements . . . 24
    - 2.1.1.6 Transformation to global coordinates . . . 25
  - 2.1.2 Springs . . . 27
  - 2.1.3 Connection stiffness . . . 28
- 2.2 Canonical stiffnesses . . . 29
- 2.3 Structural Optimisation . . . 30
  - 2.3.1 Introduction to structural optimisation . . . 30
    - 2.3.1.1 Design space and solution space . . . 31
    - 2.3.1.2 The need for heuristics . . . 33
- 2.4 Optimisation Algorithms . . . 35
  - 2.4.1 Iterative section sizer . . . 35
  - 2.4.2 Section Rotator . . . 36
  - 2.4.3 Mode shape optimisation . . . 39
  - 2.4.4 Genetic algorithms . . . 40

3 Method **42**
- 3.1 Project overview . . . 42

3.1.1 Rhinoceros and Grasshopper . . . 44
3.1.2 Creating plugins for Grasshopper . . . 44
3.1.3 Armadillo . . . 46
3.1.3.1 BLAS and LAPACK . . . 46
3.1.4 C++/CLI . . . 47
3.2 Development of a finite element solver . . . 48
3.2.1 Finite element engine . . . 48
3.2.1.1 Structure . . . 48
3.2.1.1.1 Degrees of freedom . . . 48
3.2.1.1.2 Nodes . . . 48
3.2.1.1.3 Elements . . . 49
3.2.1.2 Forces . . . 49
3.2.1.3 Solvers . . . 49
3.2.1.3.1 Linear solver . . . 49
3.2.1.3.2 Eigen solver . . . 50
3.2.1.4 Post-processing . . . 50
3.2.1.4.1 Section forces and displacements . . . 50
3.2.1.4.2 Utilisation checks . . . 51
3.2.1.5 Optimisers . . . 52
3.2.1.5.1 Section sizer and rotator . . . 52
3.2.1.5.2 Mode shape optimiser . . . 52
3.2.1.5.3 Combined optimiser . . . 52
3.2.2 Wrapper . . . 52
3.2.3 Grasshopper plug-in . . . 53
3.2.3.1 Overview . . . 53
3.2.3.2 Geometry generation . . . 54
3.2.3.3 Elements . . . 54
3.2.3.4 Material . . . 55
3.2.3.5 Restraints . . . 56
3.2.3.6 Forces . . . 56
3.2.3.7 Structure . . . 57
3.2.3.8 Solver . . . 57
3.2.3.9 Results . . . 58
3.2.3.10 Optimisers . . . 59
3.2.4 Verification of results . . . 60
3.3 Optimisation methods . . . 61
3.3.1 Iterative section sizer . . . 61
3.3.1.1 General . . . 61
3.3.1.2 Method 1: Smallest acceptable section . . . 62
3.3.1.3 Method 2: Step-wise incrementation . . . 63
3.3.1.4 Method 3: Step-wise incrementation with down-sizing 64
3.3.1.5 Method 4: First acceptable solution from current section . . . 65
3.3.1.6 Sorting of cross sections . . . 66
3.3.2 Section Rotator . . . 66
3.3.3 Mode shape optimiser . . . 67

3.3.3.1 Procedure . . . 67
3.3.4 Combined section sizer . . . 68
3.3.5 Section optimisation using genetic algorithms . . . 68
3.3.6 Connection stiffness optimisation . . . 69
3.4 Strategy for experiments . . . 70

4 Implementation **71**
4.1 Linear solver . . . 71
4.2 Optimisation methods . . . 72
4.2.1 Iterative section sizer and rotator . . . 72
4.2.1.1 Method 1: First acceptable section . . . 74
4.2.1.2 Method 2: Step-wise incrementation . . . 74
4.2.1.3 Method 3: Step-wise incrementation with down-sizing 75
4.2.1.4 Method 4: First acceptable solution from current section . . . 75
4.2.1.5 Section rotator . . . 76
4.2.2 Mode shape optimiser . . . 77
4.2.3 Combined section sizer . . . 78

5 Case Studies **79**
5.1 Small 2d frame . . . 79
5.1.1 Geometry . . . 79
5.1.2 Boundary conditions . . . 79
5.1.3 Loads . . . 80
5.1.4 Optimisation settings . . . 80
5.2 3d frame . . . 81
5.2.1 Geometry . . . 81
5.2.2 Boundary conditions . . . 81
5.2.3 Loads . . . 81
5.2.4 Optimisation settings . . . 82
5.3 Dome . . . 83
5.3.1 Geometry . . . 83
5.3.2 Boundary conditions . . . 83
5.3.3 Loads . . . 84
5.3.4 Cross sections . . . 84
5.3.5 Optimisation settings . . . 84
5.4 KAFD Metro Station . . . 86
5.4.1 Geometry . . . 87
5.4.2 Boundary conditions . . . 87
5.4.3 Loads . . . 87
5.4.4 Cross sections . . . 88
5.4.5 Optimisation settings . . . 89

6 Results **90**
6.1 Small 2d frame . . . 90
6.1.1 Section sizer . . . 90
6.1.2 Genetic algorithms . . . 91

6.1.3 Canonical stiffnesses and eigenmodes . . . 92
6.1.4 Mode section sizer . . . 93
6.1.5 Combined section sizer . . . 94
6.1.6 Comments . . . 96
6.2 3d frame . . . 97
6.2.1 Section sizer and rotator . . . 97
6.2.2 Genetic algorithms . . . 98
6.2.3 Canonical stiffnesses and eigenmodes . . . 99
6.2.4 Mode section sizer . . . 100
6.2.5 Combined section sizer . . . 101
6.2.6 Comments . . . 102
6.3 Dome . . . 103
6.3.1 Section sizer and rotator . . . 103
6.3.2 Canonical stiffnesses and eigenmodes . . . 105
6.3.3 Mode section sizer . . . 106
6.3.4 Combined section sizer . . . 107
6.3.5 Section sizer with custom settings . . . 109
6.3.5.1 Change of initial condintions . . . 110
6.3.5.2 Combination of methods . . . 111
6.3.6 Comments . . . 112
6.4 KAFD Metro Station . . . 113
6.4.1 Section sizer and rotator . . . 113
6.4.1.1 Minimal initial sections . . . 113
6.4.1.2 Original initial sections . . . 115
6.4.2 Canonical stiffnesses and eigenmodes . . . 117
6.4.3 Mode section sizer . . . 118
6.4.4 Combined section sizer . . . 119
6.4.5 Comments . . . 121

7 Discussion **122**
7.1 CIGull . . . 122
7.2 Iterative Section Sizer . . . 122
7.2.1 Method 1 . . . 122
7.2.2 Method 2 . . . 123
7.2.3 Method 3 . . . 123
7.2.4 Method 4 . . . 123
7.2.5 Combining methods . . . 124
7.2.6 Section rotator . . . 124
7.3 Genetic algorithms . . . 124
7.4 Connection stiffness optimisation . . . 125
7.5 Mode section sizer . . . 127
7.6 Combined section sizer . . . 127
7.7 Package release . . . 127
7.8 Open source code development . . . 128
7.9 Future implementation . . . 128
7.9.1 Structural Mechanics . . . 128

7.10 Future studies 130
7.10.1 Mode shape optimiser 130

Appendix A Comparison of FE calculations **II**

Appendix B Cross sections 2d and 3d Frame **VII**

Appendix C Cross sections Dome **IX**

Appendix D Cross sections KAFD **XII**

Appendix E 3D-frame eigenvalues **XIII**

Division of work

The majority of the thesis work have been made in close collaboration between the writers, and both of us stand behind this report together. For the coding part one person may have written the code the other may have been the one checking and rewriting it. There are though some parts where one or the other writer have done the majority:

Isak Näslund has been the main contributor to writing the code for the optimisers. He has also designed the algorithms and written the majority of the implementation chapter. In the case study and result chapters he has been the main writer of the 'Dome' and 'KAFD Metro Station' sections. He has also written most of the theory part on finite elements and canonical stiffnesses.

Carl Hoff has written much of the code for the finite element solver. He has also written the main parts of the introduction and method chapters, as well as the structural optimisation section in the theory chapter. In the case study and result chapters he has been the main writer of the 2d- and 3d-frame sections.

1
Introduction

1.1 Background

The beginning of a design process is characterised by uncertainty. The typical scenario at an engineering practice is to quickly respond to a design idea generated by the architect, without having neither all the required project specific information nor the time to do a thorough proposal. The aim is then to produce a reasonable draft of a structure to which the other parts of the design team can react. The process of coming up with this initial design of the structure is often driven by the experience of the engineer, and often involves some form of rationalisation. The initial design is also prone to influence the end result in ways that are not always efficient for the project in terms of structural weight (i.e. material use).

There exists a wide range of optimisation methods, wherein the Fully Stressed Design (FSD) is one. This method is based on that the optimal design is obtained when every element is subjected to a stress corresponding to its maximum capacity in at least one loadcase.
This is quantified by Greiner et al. [Greiner, Emperador (2015)] by using equation 1.1:

$$FSD = \sqrt{\sum_{i=1}^{n} \left(\sigma_{i,max} - \sigma_{Ri,max}\right)^2} \tag{1.1}$$

Where $\sigma_{i,max}$ is the maximum stress in one element under any one load case and $\sigma_{Ri,max}$ is the maximum allowed stress in that element. With a set of discrete sections is it possible to minimise this value in order to find an optimum distribution for a given problem. This renders one solution for a statically determinate problem but several distinct solutions for a statically indeterminate one. Given that most structures in practice are statically indeterminate, this observation raises the question which solution should be chosen out of the different FSD:s available? In terms of structural weight this question can be condensed into an optimisation problem, which of the possible solutions is the lightest? It can also be seen as a possibility to evaluate the alternate designs for other criteria than just the weight, for instance structural redundancy, architectural qualities or the pragmatism of the design.
The phenomenon of several distinct FSD:s also poses a problem for an engineering problem if it's not taken into account. Consider a common engineering practice; to analyse a design, find the elements which are overutilised, and increasing their

capacity. This is effectively treating elements in isolation rather than as a system, which leads to structural systems that are optimised locally instead of globally. In a structurally indeterminate system stiffness attracts forces which means that elements cannot be treated solely in isolation, since any change will also affect the whole structure. In the case described here, where some overutilised elements gets strengthened, this means that they may attract even more force, and thus a loop is formed. When a design that fulfils the FSD criterion eventually is obtained, it is far from certain whether this is the 'best' solution. This method is also very sensitive to the initial design of the structure, rendering different solutions for varying initial conditions. [Mueller, Burns (2001)]

As an alternative one could consider all fully stressed designs for a structure, which would render several responses for which a new set of fitness criteria (weight, similar cross section types etc.) could be applied in order to find the best solution. However, this of course means that every possible combination of the problem would have to be considered, rendering a vast number of structures to be analysed. In order to save computation time some so called *heuristics* could be used. A heuristic is a search method that does not necessarily generate the best solution, but rather a 'good enough' solution, but in exchange have a significantly better performance in terms of speed. Some of these methods are:

- Genetic algorithms
- Simulated annealing
- Particle swarm optimisation
- Ant colony optimisation

These algorithms are relatively effective tools for finding 'optimal' solutions, but in order to use them a value to optimise for is needed. If the loads are known the FSD (or some other) value could be used, but what if the loads are unknown? This is, as mentioned earlier, a quite common case in the early design stages. Furthermore, the inherent problem still isn't solved by using these methods. The search methods may be more likely to find a good result, but the non-linear nature of structural engineering tasks makes it very difficult to ensure that the obtained optimal solution is really also the global optimum.

This thesis aims to create tools that helps the user explore the solution space of structural engineering. By developing generative tools that are easy to use and so streamlining the design process, the intention is to make it easier

1.1.1 Stiffness redistribution

Although the concept of statically indeterminate structures may seem troublesome at first glance (for calculation purposes, since the loads can take be carried in several different ways), it can also be regarded as an opportunity for good structural design. Consider a simple frame with rigid connections and a horisontal load:

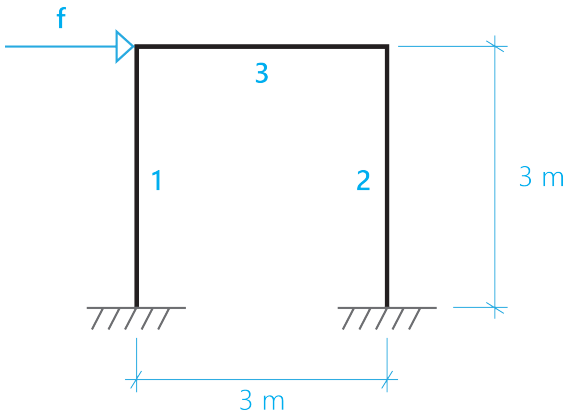


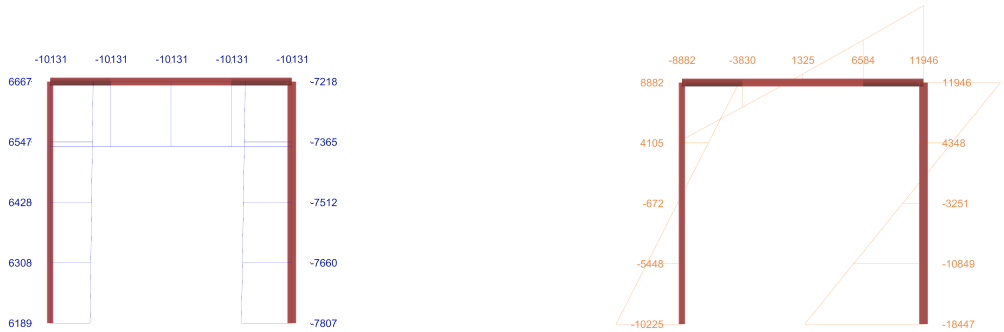
Figure 1.1: A simple frame consisting of three beams. All connections are rigid.

The vertical elements are fixed to the ground, and all connections are rigid. It is loaded with a horisontal force of $\mathbf{f = 15.9}$ [kN] at the connection between element 1 and 3 (see fig. **??**). The elements have rectangular cross sections of varying size, see table 1.1. The material is steel, with $E = 210$ [GPa] and $f_y = 275$ [MPa]

Element	Section property
1	RHS 70 x 50 x 8
2	RHS 110 x 50 x 8
3	RHS 120 x 50 x 8

Table 1.1: Section properties for the three elements
*RHS = Rectangular hollow section [HH] x [WW] x [t]

When analysing this structure the following axial forces and moments are obtained:



(a) Axial forces

(b) Moments

By calculating a combined stress and comparing to the yield stress, utilisation factors can be obtained (see fig. 1.3).

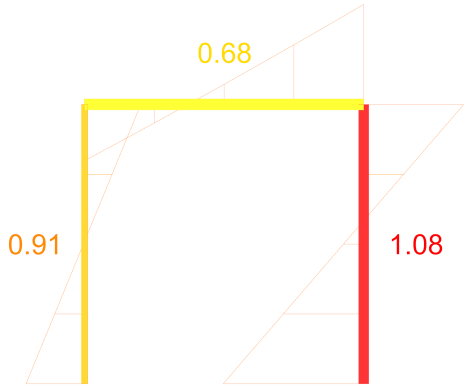
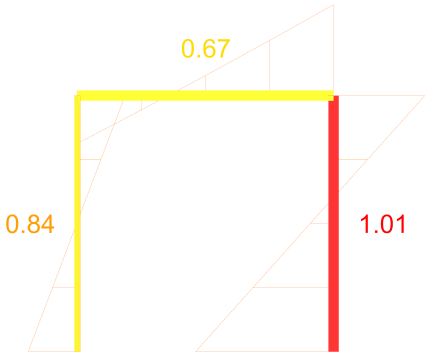
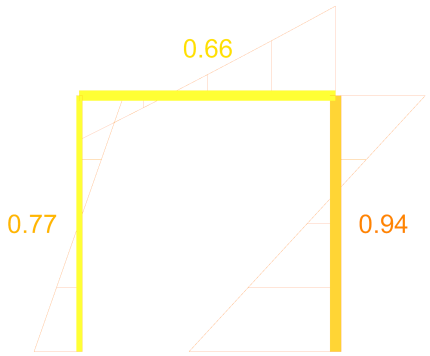


Figure 1.3: Utilisation

When analysing these results, one can conclude that the utilisation is high but acceptable in the left column and the top beam, and too high in the right column. In order to produce an acceptable response, the utilisation needs to be reduced in the right column. The first idea that comes to mind is, naturally, to increase the height of the cross section in that column. This has been done in fig. 1.4. In image 1.4a it can be noted that the utilisations have been reduced in all elements, but the utilisation in the right column is still over 1. The size needs to be increased once more, rendering the result in fig 1.4b, which is acceptable.



(a) Cross section of right column increased to: RHS 120 x 50 x 8



(b) Cross section of right column increased to: RHS 130 x 50 x 8

Figure 1.4: Utilisations for the frame as the cross section of the right column is increased.

While there is nothing that is obviously wrong with this process (it may very well be the best solution with regards to the particular circumstances), it is always interesting to review the alternative solutions. What happens for instance, if the size of the left column is increased instead? Analysing the results (fig. 1.5) one can see that the results are acceptable even after just one incrementation. Since the

structure is statically indeterminate, the load path is determined by the stiffness of the elements. While increasing the section size of the right column does increase its capacity it also increases its stiffness, and therefore 'attracts' more load. Increasing the section size of the left column can therefore redistribute the load path into a more efficient structural behaviour.

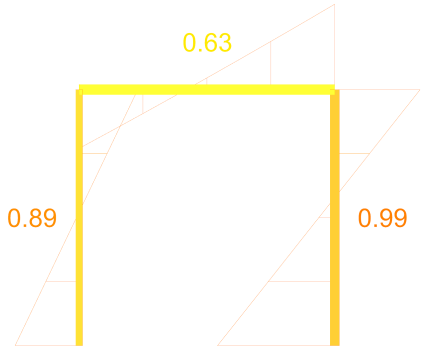
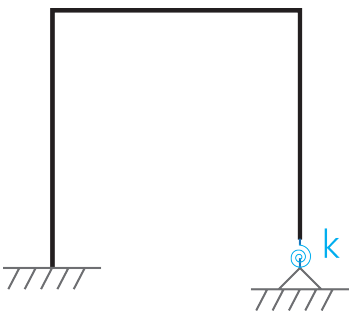
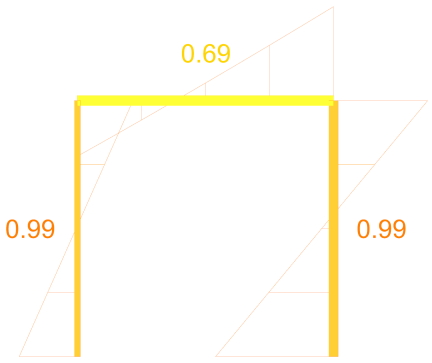


Figure 1.5: Utilisations for the frame as the cross section of the left column is increased to: RHS 90 x 50 x 8.

Another way of redistributing the force flow in a structure could be to deal with the problem more directly. Structural connections are often thought of as sort of binary, either fixed or free. This is a simplification that is often reasonable to make, but in reality almost all connections are not fully fixed or fully restrained, but something in between. In this example all connections are modelled as stiff, but what if that is changed? The highest moment in the right column is found closest to the support. If the capacity for that connection was to be lowered, the load is forced to the left column instead. Using a rotational spring (see fig. 1.6a) with a stiffness of $k = 6 \cdot 10^6$ $[\frac{Nm}{rad}]$ results in an acceptable utilisation rate using the original sections.



(a) New boundary conditions, introducing a rotational spring at the right support.



(b) Utilisations using a spring stiffness of $k = 6 \cdot 10^6$ $[\frac{Nm}{rad}]$

Figure 1.6: The original frame new boundary conditions

The results of this short example are presented in table 1.2.

Model desc.	Element	Section property	Util. [%]	Weight [kg]
Original	1	RHS 70 x 50 x 8	91	168.5
	2	RHS 110 x 50 x 8	108	
	3	RHS 120 x 50 x 8	68	
Increase right column size	1	RHS 70 x 50 x 8	77	176.0
	2	RHS 130 x 50 x 8	94	
	3	RHS 120 x 50 x 8	66	
Increase left column size	1	RHS 80 x 50 x 8	89	172.2
	2	RHS 110 x 50 x 8	99	
	3	RHS 120 x 50 x 8	63	
Connection stiffness	1	RHS 70 x 50 x 8	99	168.5
	2	RHS 110 x 50 x 8	99	
	3	RHS 120 x 50 x 8	69	

Table 1.2: Results from the different methods presented above, including a weight of the structure (using $\rho = 7800\frac{kg}{m^3}$)

This is, clearly, a quite specific example designed to prove a point, and far from all structures exhibit these properties. Irregardless, what the example does show is that the most intuitive solution to a structural design problem is not necessarily the best one. From table 1.2 it can be seen that editing the connection stiffness produced the lowest structural weight and thus the lowest material usage. But is that then the best solution? One can't say for sure. There is no such thing as a free lunch, and all solutions have their benefits and setbacks. Lowering the connection stiffness does provide an acceptable solution, but at the expense of increased deflection. Increasing the left column size also uses less material than increasing the right column, but pushes the utilisation rate to 99% for one of the elements, which could be dangerous if there is a high degree of uncertainty in the loads.

This leads to the conclusion that in order to find a good structural design, the designer must consider many different solutions. All design strategies have their flaws, and when dealing with complicated structures the enormous number of possible solutions makes it very difficult to find them all. The criteria for which design is 'best' may not even be measurable but rather a matter of the architectural qualities of the structure.

This thesis aims to create ways of quickly and rather effortlessly generate acceptable (but not necessarily perfect) solutions for the designer to consider, in order to more easily explore the solution space (see 2.3.1.1) of structural design.

1.1.2 Finite element solving

A prerequisite for the finite element method to be effective is the presence of automated calculations. Two of the more common methods of doing this is either using MATLAB or some other matrix solver or by using a commercial FE solver. Both tools have advantages and disadvantages, which can be related to how accessible the data is. Using a matrix solver puts the user in total control, but has drawbacks in that the user interface is very time consuming to work with. Everything needs to be done manually, which creates a problem as soon as the structure is not very small and simple. As the user must keep track of which degree of freedom is which, the work becomes very tedious when working with 3d structures, leading to errors occurring frequently and an overly complicated procedure when making changes to the design.

The alternative to use a commercial software is in these regards a better option, since they are usually designed to alleviate these issues. The main setback using a commercial software is that it puts the user 'at the mercy' of the developer. Whatever the developer has implemented is that which the user can access. If the user wishes to access some data there is sometimes an application programming interface (API) from which more data can be accessed. However, this also only includes what the developer has chosen to expose, so the inherent problem is still unsolved.

In order to resolve this issue one need to use a working method which puts the user in control, but makes it easier to work with. The ideal option is to implement the structural engineering methods that a matrix solver is using in a user-friendly framework. A possible solution is to use available finite element packages, which exist for some different environments (both for code and so called 'visual programming' environments). Everything comes down to how much control the user requires, where the maximum control means implementing it themselves.

1.1.3 Canonical stiffnesses

The theory of canonical stiffnesses was presented by Karl-Gunnar Olsson and Carl Thelin in their 2003 report [Olsson, Thelin (2003)]. The theory is presented in more detail in section 2.2, but can be summarised as a method of finding the 'weakest' deformation modes of a structure, i.e. the deformations that require the lowest amount of energy to occur. Since this method can be used to measure the stability of a structure, it is of interest to investigate how it can be used as an alternative way to generate and possibly optimise structural design.

1.2 Aim

The aim of the thesis is to investigate ways of increasing structural capacity through the redistribution of forces. Methods of structural engineering design will be developed and tested as a tool for structural design sketching. This tool should produce acceptable designs for a given case. The aim of the thesis is also to investigate whether the use of stiffness redistribution can be implemented as a useful tool for engineers in order to understand how a structure works. Furthermore, is it possible to use this knowledge in an automated process, in order to in a similar fashion generate a set of designs? One method that is intended to be studied for this purpose is the theory of canonical stiffnesses (see section 2.2), which can be used for obtaining information on the weaknesses of the structure.

The obtained solutions are intended to be used as a *Design Support System* (DSS), as opposed to the choice of producing one "optimal" solution in a so called *black box solver*. The resulting designs should be possible for the user to adapt to the specific design case. Although the process should be as general as possible, it also needs to be very easy model a problem and to interpret the results, since this allows a (somewhat) seamless process of interpreting and reacting to the results.

Another aim of this thesis is to develop a computationally sound framework, which could easily be expanded to implement new functions. It is also of interest to explore how using programming as a tool can enable structural engineers to extend their control of their work. By designing a robust framework of code, and by using custom-built code, the user is only limited to their own imagination.

1.3 Method

In the light of the aim of the thesis, the method can be summarised as follows:

1. Obtain a finite element solver that
   - is accessible from Grasshopper (see 3.1.1)
   - has a quick calculation response
2. Create a plug-in for Grasshopper that can access the FE solver
3. Develop a set of different design optimisation methods:
   - *Section sizer* (see 3.3.1)
   - Genetic algorithms (see 2.4.4)
   - Canonical stiffness (see 2.2 (theory) and 2.4.3 (implementation))
4. Test these methods on some basic structures to analyse their behaviour
5. Make a case study on a real life project

Choice of FEM solver

In order to optimise a structure a means of analysing it must be provided. There are several means of doing this, some of the identified alternatives are:

1. The CALFEM package for MATLAB.
2. Using a commercial FEM software, for instance GSA / Robot / SAP2000, etc.
3. Programming to the API of one of the commercial FEM softwares.
4. Karamba, a commercial FEM package for Grasshopper.
5. Making a basic FEM solver for Grasshopper.

In the context of this thesis, there are two constraints to the choice of solver:

- The API needs to be exposed.
- The response must be very quick for the iterative analysis to be a viable option.

Since one of the end goals of the thesis work is to create a tool for Grasshopper, **Options 1 and 2** are not possible. **Option 3** is a better version of option 2, and by controlling the FEM software from grasshopper is a good way to make sure that the basic structural mechanic calculations are made in a good way. However, this requires the entire API to be exposed and it also adds an unnecessary inefficiency in the link between Grasshopper and the software, an inefficiency that may be very detrimental to the overall performance given that it is expected that the number of iterations will be quite high. An alternative then is **Option 4**, using a native Grasshopper FEM solver like Karamba. However, this shares the problems of Option 3, although not to the same extent. The main reason why Options 3 and 4 are not viable is that in order to solve the eigenvalue problem not only the API needs to be exposed, but rather the entire stiffness matrix.

Therefore **Option 5** should be used, developing a (simplified) native FEM solver for Grasshopper.

Summarised specification of task

The main questions that should be investigated are:

- What are the implications of a design process which begins in some design and adjusts the section properties until a satisfactory result is obtained? Can this process be automated?
- What alternatives to "regular" automated section sizing exists, and what are their advantages and drawbacks?
- Can canonical stiffnesses be used as such an alternative and
  - as a method of finding efficient structures?
  - as a way of understanding a structure when there is much uncertainty in the design process?
- Can the use of connection stiffnesses redistribute the flow of forces in a structure, in order to reduce weight?
  - Is it possible to automate this process?
  - How can this be realized in practice?

1.4 Limitations

1.4.1 General

Since the scope of the problem is in the discrete range, a natural limitation of the thesis is that continuous solutions will be disregarded. This means that the study will be performed using elements with constant cross-sections and discrete dimensions. Another limitation of the thesis is that any topology optimisation will be omitted. This limitation facilitates comparing eigenvalues and eigenmodes (see the section on canonical stiffnesses, section 2.2) of the stiffness matrix to one another, since any change of the degrees of freedom will increase the complexity of comparing the eigenvalues and modes.

The aim of the thesis is to investigate the method of design generation. Rather than developing one specific method, the intention is to develop several, which should produce different designs for the engineer to compare and choose from. Since the aim is focused in this way, the individual methods will not be investigated to the extent that they would have been, should only one method have been developed.

When developing 'acceptable' designs, some form of code checking must be done. Earlier in the introduction (see fig. 1.3) this was done by checking a utilisation rate based on combined axial and bending (see 3.2.1.4.2). These checks could be done very rigorously, and should for the final verification of a design. In this thesis though, the process is be simplified to only include some basic checks. It should however be build in such a manner that it can be easily expanded with more checks at a later time.

1.4.2 Section properties

To be able to perform a strength check on the elements of a structure, it must be possible to check various stresses. Since the FEM solver will be implemented for three dimensional calculations, this includes torsional checks. Torsion theory has two parts, St Venant's theory and Vlassov's theory, where the former is dominating for closed sections and the latter dominating for open sections (see 2.1.1.3). Given the complex nature of Vlassov's theory it is reasonable as a first step to only implement closed sections (which also will simplify the checks for other stresses). In summary, this limits the sections to be used to:

- Rectangular hollow sections (RHS)
- Rectangular solid sections
- Circular hollow sections (CHS)
- Circular solid sections

1.5 Outline of the report

In **chapter 2** an introduction to the finite element theory is presented, which has been implemented in the thesis work. Thereafter follows an introduction to the theory of canonical stiffness, an introduction to the optimisation problem and a description of some of the optimisation algorithms and methods that currently exist.

Chapter 3 and **4** describes the way that the theory have been implemented in two steps, **chapter 3** more in general, and **chapter 4** more in detail.

Chapter 5 presents several case studies where the various methods have been tested on some test structures, and on a real project, the *King Abdullah Financial District (KAFD) Metro Station* in Ar Riyadh, Saudi Arabia, by Zaha HAdid Architects and Buro Happold Engineering.

The results are presented in **chapter 6**, and discussed in **chapter 7**.

2
Theory

2.1 Finite elements

To solve structural problems different methods exists. One of the most commonly used is finite element analysis in which each member is discretized. In this thesis frame structures will be the focus point and both 2D and 3D situations will be covered, with a focus on 3D structures. The following section will give a brief overview of finite elements used in this thesis. For further reading refer to [Dahlblom, Olsson (2010)].

2.1.1 Frame elements

A frame element consists of multiple differential equations governing its behaviour for its different deformation modes and degrees of freedom. For a 2D case each end of the element has three degrees of freedom, two translational and one rotational, whereas in 3D the element will have six degrees of freedom on each node, 3 translational and 3 rotational. Translation in the direction of the centreline of the beam will be governed by bar action, rotation around and translation along the axes perpendicular to the centre line by beam action and, for the 3D-case, rotation around the centreline by torsion. All theory used is under the assumption that the members have a constant cross section and a constant module of elasticity.

2.1.1.1 Bar action

Figure 2.1: Degrees of freedom and distributed load for a bar

The equilibrium equation for a bar is:

$$\frac{dN}{dx} + q_x(x) = 0; \quad 0 \leq x \leq L \tag{2.1}$$

with the constitutive equation:

$$N = EA\frac{du}{dx} \tag{2.2}$$

These equations can be used to construct the differential equation for bar action, or axial stiffness, also called the strong form[Ottosen, Pettersson (1992)]:

$$\frac{d}{dx}\left(EA\frac{du}{dx}\right) + q_x(x) = 0; \quad 0 \leq x \leq L \tag{2.3}$$

This equation can, using the previously mentioned assumptions of constant cross section area and modulus of elasticity, be simplified to:

$$EA\frac{d^2u}{dx^2} + q_x(x) = 0; \quad 0 \leq x \leq L \tag{2.4}$$

where:

$E =$ Modulus of elasticity$[N/m^2]$
$A =$ Cross section area$[m^2]$
$u =$ Displacement along local x-axis$[m]$
$q_x(x) =$ Axial load distributed along the member$[N/m]$
$L =$ Length of the bar$[m]$
$N =$ Normal force$[N]$

This differential equation has four possible boundary conditions. These boundary conditions can be either geometrical or natural. The geometric boundary conditions are defined as:

$$\begin{aligned} u(0) &= u_0 \\ u(L) &= u_L \end{aligned} \tag{2.5}$$

These boundary conditions can be used when a displacement is prescribed at one or both ends of the bar.
The natural boundary conditions are defined as:

$$\begin{aligned} N(0) &= EA\frac{du}{dx}\bigg|_{x=0} = N_0 \\ N(L) &= EA\frac{du}{dx}\bigg|_{x=L} = N_L \end{aligned} \tag{2.6}$$

These boundary condition can be used when a normal force is prescribed at one or both ends of the bar. As the differential equation (eq 2.4) is a second degree differential equation it is necessary to have two boundary condition to solve it, one for each end of the bar. Used boundary conditions can be two geometrical, two natural or a combinations of the two.
To obtain the weak form of eq 2.4, used to construct the FE-formulation, the equation is multiplied by an arbitrary weight function $\nu(x)$ and integrated over the length of the element [Ottosen, Pettersson (1992)]:

$$\int_0^L \nu \left[EA\frac{d^2u}{dx^2} + q_x(x) \right] dx = 0 \tag{2.7}$$

Integration by parts of the first term and insertion of the constitutive condition in eq 2.2 leads to the weak form:

$$\int_0^L \frac{d\nu}{dx} EA\frac{du}{dx} dx = -\left[\nu N\right]_0^L + \int_0^L \nu q_x(x) dx \tag{2.8}$$

The finite element approximation is introduced for the displacement as:

$$u = \mathbf{N}^e\mathbf{a}^e = \begin{bmatrix} N_1^e & N_2^e \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \end{bmatrix} \tag{2.9}$$

where $N_i = N_i(x)$ are shape functions, not to be confused with the normal force N, and u_i are the displacements at the endpoints of the element. As $\mathbf{a}$ is independent of x the following can be concluded:

$$\frac{du}{dx} = \mathbf{B}^e\mathbf{a}^e = \frac{d\mathbf{N}^e}{dx}\mathbf{a}^e = \begin{bmatrix} \frac{dN_1^e}{dx} & \frac{dN_2^e}{dx} \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \end{bmatrix} \tag{2.10}$$

Inserting eq 2.10 in eq 2.8 gives:

$$\left(\int_0^L \frac{d\nu}{dx} EA\mathbf{B}^e dx \right) \mathbf{a}^e = -\left[\nu N\right]_0^L + \int_0^L \nu q_x(x) dx \tag{2.11}$$

Using the Galerkin method, the weight functions are chosen to be equal to the shape function which yields[Ottosen, Pettersson (1992)]:

$$\nu = \mathbf{N^e c} \tag{2.12}$$

As ν is arbitrary and $\mathbf{N}$ are known, $\mathbf{c}$ must be arbitrary. The fact that ν gives that $\nu = \nu^T$, which makes it possible to write eq 2.12 as:

$$\nu = \mathbf{c}^T\mathbf{N}^{eT} \tag{2.13}$$

which in turn gives that:

$$\frac{d\nu}{dx} = \mathbf{c}^T\mathbf{B}^{eT} = \mathbf{c}^T\frac{d\mathbf{N}^{eT}}{dx} \tag{2.14}$$

As $\mathbf{c}^T$ is idependent of x, inserting 2.13 and 2.14 in 2.11 gives:

$$\mathbf{c}^T \left[\left(\int_0^L \mathbf{B}^{eT} EA\mathbf{B}^e dx \right) \mathbf{a}^e + \left[\mathbf{N}^{eT} N\right]_0^L - \int_0^L \mathbf{N}^{eT} q_x(x) dx \right] = 0$$

Since this expression must hold for any arbitrary $\mathbf{c}^T$, the following must hold:

$$\left(\int_0^L \mathbf{B}^{eT} EA\mathbf{B}^e dx \right) \mathbf{a}^e = -\left[\mathbf{N}^{eT} N\right]_0^L + \int_0^L \mathbf{N}^{eT} q_x(x) dx \tag{2.15}$$

which can be rewritten to:

$$\mathbf{K}^e\mathbf{a}^e = \mathbf{f}_b^e + \mathbf{f}_l^e \tag{2.16}$$

where:

$$\mathbf{K}^e = \int_0^L \mathbf{B}^{eT} EA \mathbf{B}^e dx \tag{2.17}$$

$$\mathbf{f}_b^e = -\left[\mathbf{N}^{eT} N\right]_0^L \tag{2.18}$$

$$\mathbf{f}_l^e = \int_0^L \mathbf{N}^{eT} q_x(x) dx \tag{2.19}$$

The shape functions used must fulfill completeness and compatibility requirements. The completeness conditions for bar action are:

- The approximation for the deflection u must be able to represent an arbitrary rigid-body motion.
- The approximation for the deflection u must be able to represent a constant strain state.

and the compatibility condition is [Ottosen, Pettersson (1992), p.324]:

- The approximation for the displacement u must vary continuously within the element and over it's boundaries boundaries.

The simplest shape functions that fulfil these for a bar element are:

$$\begin{aligned} N_1^e &= -\frac{1}{L}(x - L) \\ N_2^e &= \frac{1}{L}x \end{aligned} \tag{2.20}$$

which in turn gives:

$$\begin{aligned} B_1^e &= \frac{N_1^e}{dx} = -\frac{1}{L} \\ B_2^e &= \frac{N_2^e}{dx} = \frac{1}{L} \end{aligned} \tag{2.21}$$

Inserting eq 2.21 in eq 2.17 gives:

$$\begin{aligned} \mathbf{K}^e &= \int_0^L \left(\begin{bmatrix} -\frac{1}{L} \\ \frac{1}{L} \end{bmatrix} EA \begin{bmatrix} -\frac{1}{L} & \frac{1}{L} \end{bmatrix} \right) dx = \\ &= EA \begin{bmatrix} \frac{1}{L^2} & -\frac{1}{L^2} \\ -\frac{1}{L^2} & \frac{1}{L^2} \end{bmatrix} \int_0^L 1 dx = \frac{EA}{L} \begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix} \end{aligned} \tag{2.22}$$

Inserting eq 2.20 in eq 2.18 gives:

$$\mathbf{f}_b^e = -\left[\begin{bmatrix} -\frac{1}{L}(x - L) \\ \frac{1}{L}x \end{bmatrix} N \right]_0^L = -\begin{bmatrix} 0 \\ 1 \end{bmatrix} N(L) + \begin{bmatrix} 1 \\ 0 \end{bmatrix} N(0) = \begin{bmatrix} N(0) \\ -N(L) \end{bmatrix} \tag{2.23}$$

Inserting eq 2.20 in eq 2.19 gives:

$$\mathbf{f}_l^e = \int_0^L \begin{bmatrix} -\frac{1}{L}(x-L) \\ \frac{1}{L}x \end{bmatrix} q_x(x)dx$$

assuming constant load $q_x(x) = q_x$ this gives:

$$\mathbf{f}_l^e = q_x \int_0^L \begin{bmatrix} -\frac{1}{L}(x-L) \\ \frac{1}{L}x \end{bmatrix} dx = \frac{q_x}{L} \left[\begin{bmatrix} -\frac{x^2}{2} + Lx \\ \frac{x^2}{2} \end{bmatrix} \right]_0^L = \frac{q_x L}{2} \begin{bmatrix} 1 \\ 1 \end{bmatrix} \tag{2.24}$$

To summarise:

$$\mathbf{K}^e = \frac{EA}{L} \begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix} \tag{2.25}$$

$$\mathbf{f}_b^e = \begin{bmatrix} N(0) \\ -N(L) \end{bmatrix} \tag{2.26}$$

$$\mathbf{f}_l^e = \frac{q_x L}{2} \begin{bmatrix} 1 \\ 1 \end{bmatrix} \tag{2.27}$$

2.1.1.2 Beam action

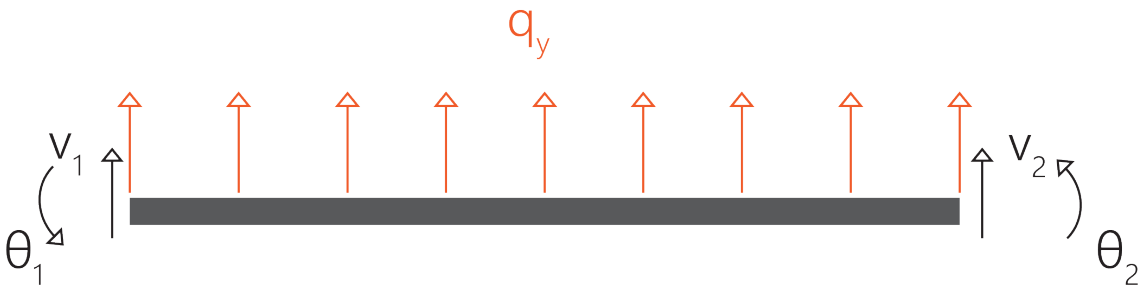


Figure 2.2: Degrees of freedom and distributed load for a beam

There exist multiple different theories for 1D structures carrying load in bending, i.e beams. One of the most commonly used and most straight forward theories is the Euler-Bernoulli beam theory. This theory implies several assumptions, namely:

- Small displacements and rotations
- Small strains
- Plane sections remain plane and orthogonal to the centre line

For the scope of this thesis these assumptions are deemed reasonable why this beam theory is used. The equilibrium equations for a beam are:

$$\frac{dV}{dx} = -q_y(x); \quad 0 \leq x \leq L \tag{2.28}$$

and

$$\frac{dM}{dx} = V; \quad 0 \leq x \leq L \tag{2.29}$$

with the constitutive relation:

$$M = -EI\frac{d^2v}{dx^2} \tag{2.30}$$

As the centre line is assumed to be orthogonal to the section planes, it means that the shear strain will be zero. This means that the shear force will need to be evaluated based on the deflection. Inserting eq 2.28 in eq 2.29 gives:

$$\frac{d^2M}{dx^2} + q_z(x) = 0; \quad 0 \leq x \leq L \tag{2.31}$$

Inserting the constitutive relation eq 2.30 gives the differential equation for an Euler-Bernoulli beam, also called the strong form [Ottosen, Pettersson (1992)]:

$$\frac{d^2}{dx^2}\left(EI\frac{d^2v}{dx^2}\right) - q_y(x) = 0 \quad 0 \leq x \leq L$$

which can be simplified, assuming constant cross section and modulus of elasticity, to:

$$EI\frac{d^4v}{dx^4} - q_y(x) = 0 \quad 0 \leq x \leq L \tag{2.32}$$

where:

$E =$ Modulus of elasticity$[N/m^2]$
$I =$ Second moment of inertia$[m^4]$
$v =$ Transversal deflection along local x-axis$[m]$
$q_y(x) =$ Transversal load distributed along the member$[N/m]$
$L =$ Length of the beam$[m]$
$M =$ Bending moment$[Nm]$
$V =$ Shear force$[N]$

This differential equation has eight possible boundary conditions. These boundary conditions can be either geometrical or natural. The geometric boundary conditions are defined as:

$$\begin{aligned} v(0) &= u_0 \\ v(L) &= u_L \end{aligned} \tag{2.33}$$

and

$$\begin{aligned} \left.\frac{dv}{dx}\right|_{x=0} &= \theta_0 \\ \left.\frac{dv}{dx}\right|_{x=L} &= \theta_L \end{aligned} \tag{2.34}$$

The boundary conditions in eq 2.33 can be used when a bending moment is prescribed at one or both ends of the beam and the ones in eq 2.34 when the shear force is prescribed at one or both ends of the beam.
The natural boundary conditions are defined as:

$$\begin{aligned} EI\frac{d^2u}{dx^2}\Big|_{x=0} &= M_0 \\ EI\frac{d^2u}{dx^2}\Big|_{x=L} &= M_L \end{aligned} \tag{2.35}$$

and

$$\begin{aligned} EI\frac{d^3u}{dx^3}\Big|_{x=0} &= V_0 \\ EI\frac{d^3u}{dx^3}\Big|_{x=L} &= V_L \end{aligned} \tag{2.36}$$

The boundary conditions in eq 2.35 can be used when a displacement is prescribed at one or both ends of the beam and the ones in eq 2.36 when the rotation is prescribed at one or both ends of the beam.
As the differential equation (eq 2.32) is a fourth degree differential equation it is necessary to have four boundary condition to solve it, two for each end of the beam. As rotation is linked to bending moment and shear force to translation, the two boundary conditions for each end need to be one of the BC:s in eq 2.33 or eq 2.36 and one of the BC:s in eq 2.34 or eq 2.36.
Similar to the bar, a weak form is sought. To get this the strong form in eq 2.32 is multiplied by an arbitrary weight function ν and integrated between its boundaries. As for the bar, integration by parts gives the weak form:

$$\int_0^L \frac{d^2\nu}{dx^2}M dx = \left[\frac{d\nu}{dx}M\right]_0^L - [\nu V]_0^L - \int_0^L \nu q_z(x)dx \tag{2.37}$$

The finite element approximation is introduced as:

$$v = \mathbf{N}^e\mathbf{a}^e = \begin{bmatrix} N_1^e & N_2^e & N_3^e & N_4^e \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \\ u_3 \\ u_4 \end{bmatrix} \tag{2.38}$$

whereNi=Ni(x)are shape functions,u_i are nodal values at the endpoints of the element. As $\mathbf{a}$ is independent of x the following can be concluded:

$$\frac{d^2v}{x^2} = \mathbf{B}^e\mathbf{a}^e = \frac{d^2\mathbf{N}^e}{x^2}\mathbf{a}^e = \begin{bmatrix} \frac{d^2N_1^e}{x^2} & \frac{d^2N_2^e}{x^2} & \frac{d^2N_3^e}{x^2} & \frac{d^2N_4^e}{x^2} \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \\ u_3 \\ u_4 \end{bmatrix} \tag{2.39}$$

As for the bar the Galerkin method is used, which also here is transposable giving:

$$\nu = \mathbf{N}^e \mathbf{c} = \mathbf{c}^T \mathbf{N}^{eT} \tag{2.40}$$

which gives:

$$\begin{aligned} \frac{dv}{dx} &= \mathbf{c}^T \frac{d\mathbf{N}^{eT}}{dx} \\ \frac{d^2v}{x^2} &= \mathbf{c}^T \mathbf{B}^{eT} \end{aligned} \tag{2.41}$$

Inserting eq 2.30, eq 2.40 and eq 2.41 into the weak form in eq 2.37 gives, as $\mathbf{c}^T$ is independent on x:

$$\mathbf{c}^T \left(\left(-\int_0^L \mathbf{B}^{eT} EI \mathbf{B}^e dx \right) \mathbf{a}^e - \left[\frac{d\mathbf{N}^{eT}}{dx} M \right]_0^L + \left[\mathbf{N}^{eT} V \right]_0^L + \int_0^L \mathbf{N}^{eT} q_z(x) dx \right) = 0$$

As $\mathbf{c}^T$ is arbitrary the following must hold:

$$\left(\int_0^L \mathbf{B}^{eT} EI \mathbf{B}^e dx \right) \mathbf{a}^e = \left[\mathbf{N}^{eT} V \right]_0^L - \left[\frac{d\mathbf{N}^{eT}}{dx} M \right]_0^L + \int_0^L \mathbf{N}^{eT} q_z(x) dx \tag{2.42}$$

which can be rewritten to:

$$\mathbf{K}^e \mathbf{a}^e = \mathbf{f}_l^e + \mathbf{f}_b^e \tag{2.43}$$

where:

$$\mathbf{K}^e = \int_0^L \mathbf{B}^{eT} EI \mathbf{B}^e dx \tag{2.44}$$

$$\mathbf{f}_b^e = \left[\mathbf{N}^{eT} V \right]_0^L - \left[\frac{d\mathbf{N}^{eT}}{dx} M \right]_0^L \tag{2.45}$$

$$\mathbf{f}_l^e = \int_0^L \mathbf{N}^{eT} q_z(x) dx \tag{2.46}$$

The shape functions used must fulfill completeness and compatibility requirements. For beam action the completeness conditions are [Ottosen, Pettersson (1992), p.323]:

- The approximation for the deflection v must be able to represent an arbitrary rigid-body motion.
- The approximation for the deflection v must be able to represent an arbitrary constant curvature.

and the compatibility condition is [Ottosen, Pettersson (1992), p.324]:

- The approximation for the deflection v must vary continuously and with continuous slopes over the element boundaries.

The simplest shape functions that fulfil these conditions for a beam element are:

$$\begin{aligned} N_1^e =& 1 - 3\frac{x^2}{L^2} + 2\frac{x^3}{L^3} \\ N_2^e =& x\left(1 - 2\frac{x}{L} + \frac{x^2}{L^2}\right) \\ N_3^e =& \frac{x^2}{L^2}\left(3 - 2\frac{x}{L}\right) \\ N_4^e =& \frac{x^2}{L}\left(\frac{x}{L} - 1\right) \end{aligned} \tag{2.47}$$

which in turn gives:

$$\begin{aligned} \frac{dN_1^e}{dx} =& 6\frac{x^2}{L^3} - 6\frac{x}{L^2} \\ \frac{dN_2^e}{dx} =& 1 - 4\frac{x}{L} + 3\frac{x^2}{L^2} \\ \frac{dN_3^e}{dx} =& 6\frac{x}{L^2} - 6\frac{x^2}{L^3} \\ \frac{dN_4^e}{dx} =& 3\frac{x^2}{L^2} - 2\frac{x}{L} \end{aligned} \tag{2.48}$$

and:

$$\begin{aligned} B_1^e =& \frac{d^2N_1^e}{dx^2} = 12\frac{x}{L^3} - \frac{6}{L^2} \\ B_2^e =& \frac{d^2N_2^e}{dx^2} = 6\frac{x}{L^2} - \frac{4}{L} \\ B_3^e =& \frac{d^2N_3^e}{dx^2} = \frac{6}{L^2} - 12\frac{x}{L^3} \\ B_4^e =& \frac{d^2N_4^e}{dx^2} = 6\frac{x}{L^2} - \frac{2}{L} \end{aligned} \tag{2.49}$$

Inserting eq 2.49 into eq 2.44, assuming constant E and I:

$$\begin{aligned} \mathbf{K}^e = EI \int_0^L & \begin{bmatrix} B_1^eB_1^e & B_1^eB_2^e & B_1^eB_3^e & B_1^eB_4^e \\ B_2^eB_1^e & B_2^eB_2^e & B_2^eB_3^e & B_2^eB_4^e \\ B_3^eB_1^e & B_3^eB_2^e & B_3^eB_3^e & B_3^eB_4^e \\ B_4^eB_1^e & B_4^eB_2^e & B_4^eB_3^e & B_4^eB_4^e \end{bmatrix} \\ = \ldots = & \frac{EI}{L^3}\begin{bmatrix} 12 & 6L & -12 & 6L \\ 6L & 4L^2 & -6L & 2L^2 \\ -12 & -6L & 12 & -6L \\ 6L & 2L^2 & -6L & 4L^2 \end{bmatrix} \end{aligned} \tag{2.50}$$

Looking at eq 2.47 and eq 2.48 at the beam ends, i.e. at $x = 0$ and $x = L$ gives:

$$
\begin{aligned}
N_1^e(0) = 1; \quad & N_1^e(L) = 0 \\
N_2^e(0) = 0; \quad & N_2^e(L) = 0 \\
N_3^e(0) = 0; \quad & N_3^e(L) = 1 \\
N_4^e(0) = 0; \quad & N_4^e(L) = 0 \\
\left.\frac{dN_1^e}{dx}\right|_{x=0} = 0; \quad & \left.\frac{dN_1^e}{dx}\right|_{x=L} = 0 \\
\left.\frac{dN_2^e}{dx}\right|_{x=0} = 1; \quad & \left.\frac{dN_2^e}{dx}\right|_{x=L} = 0 \\
\left.\frac{dN_3^e}{dx}\right|_{x=0} = 0; \quad & \left.\frac{dN_3^e}{dx}\right|_{x=L} = 0 \\
\left.\frac{dN_4^e}{dx}\right|_{x=0} = 0; \quad & \left.\frac{dN_4^e}{dx}\right|_{x=L} = 1
\end{aligned}
\tag{2.51}
$$

Using these properties in eq 2.45 gives:

$$
\mathbf{f}_b^e = \begin{bmatrix} -V(0) \\ M(0) \\ V(0) \\ -M(0) \end{bmatrix} \tag{2.52}
$$

Inserting eq 2.47 into eq 2.46, assuming uniform load gives:

$$
\mathbf{f}_l^e = \frac{q_y L}{2} q_z \int_0^L \begin{bmatrix} N_1^e \\ N_2^e \\ N_3^e \\ N_4^e \end{bmatrix} dx = ... = \frac{q_y L}{2} \begin{bmatrix} 1 \\ \frac{L}{6} \\ 1 \\ -\frac{L}{6} \end{bmatrix} \tag{2.53}
$$

To summarise:

$$
\mathbf{K}^e = \frac{EI}{L^3} \begin{bmatrix} 12 & 6L & -12 & 6L \\ 6L & 4L^2 & -6L & 2L^2 \\ -12 & -6L & 12 & -6L \\ 6L & 2L^2 & -6L & 4L^2 \end{bmatrix} \tag{2.54}
$$

$$
\mathbf{f}_b^e = \begin{bmatrix} -V(0) \\ M(0) \\ v(0) \\ -M(0) \end{bmatrix} \tag{2.55}
$$

$$
\mathbf{f}_l^e = \frac{q_y L}{2} \begin{bmatrix} 1 \\ \frac{L}{6} \\ 1 \\ -\frac{L}{6} \end{bmatrix} \tag{2.56}
$$

Figure 2.3: Degrees of freedom and distributed load for torsional stiffness

2.1.1.3 Torsion

Torsion is only present for elements in 3d space. Two main theories for torsion exists, St Venants and Vlasovs. St Venants theory assumes, like the beam theory, that plane sections remain plane. This only holds true for circular sections. For all other sections there will exist some warping. This warping is is especially present in open, thin walled sections, whereas for closed section St Venants torsion will dominate.
To ignore the warping, even for massive sections, that are not circular would lead to a stiffer system. To compensate for this one can multiply the torsional stiffness with an appropriate factor that reduces the stiffness.
With this reduction one can approximate the torsional stiffness, using only St Venents theory, which leads to a system that is simpler to solve.
The equilibrium equation for torsion is:

$$\frac{dT}{dx} + q_\omega(x) = 0 \quad 0 \leq x \leq L \tag{2.57}$$

with the constitutive relation, assuming constant material and cross section:

$$T = GK_v \frac{d\varphi}{dx} \tag{2.58}$$

which together gives the differential equation for torsion:

$$GK_v \frac{d^2\varphi}{dx^2} - q_\omega(x) = 0 \quad 0 \leq x \leq L \tag{2.59}$$

where:

$G =$ Shear modulus$[N/m^2]$
$K_v =$ St Venants torsional constant$[m^4]$
$\varphi =$ Rotation angle$[rad]$
$q_\omega(x) =$ Torsional load distributed along the member$[Nm/m]$
$L =$ Length of the bar$[m]$
$T =$ Torsional moment$[Nm]$

This differential equation has four possible boundary conditions. These boundary conditions can be either geometrical or natural. The geometric boundary conditions are defined as:

$$\begin{aligned} \varphi(0) &= \varphi_0 \\ \varphi(L) &= \varphi_L \end{aligned} \tag{2.60}$$

These boundary conditions can be used when a rotation angle is prescribed at one or both ends of the bar.
The natural boundary conditions are defined as:

$$\begin{aligned} GK_v \frac{d\varphi}{dx}\bigg|_{x=0} &= T_0 \\ GK_v \frac{d\varphi}{dx}\bigg|_{x=L} &= T_L \end{aligned} \tag{2.61}$$

These boundary condition can be used when a torsional moment is prescribed at one or both ends of the bar. As the differential equation (eq 2.59) is a second degree differential equation it is necessary to have two boundary condition to solve it, one for each end of the bar. Used boundary conditions can be two geometrical, two natural or a combinations of the two.
As the differential equation for torsion is of degree 2, as the bar, the same principle can be used for torsion to derive the FE-formulation. For a more detailed expiation how this is done refere to section 2.1.1.1, or [Dahlblom, Olsson (2010)] or [Ottosen, Pettersson (1992)]. Using this principle gives the following element stiffness matrix and element force vectors:

$$\mathbf{K}^e = \frac{GK_v}{L}\begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix} \tag{2.62}$$

$$\mathbf{f}_b^e = \begin{bmatrix} T(0) \\ -T(L) \end{bmatrix} \tag{2.63}$$

$$\mathbf{f}_l^e = \frac{q_\omega L}{2}\begin{bmatrix} 1 \\ 1 \end{bmatrix} \tag{2.64}$$

2.1.1.4 2D-Frame elements

As stated before, frame elements in 2D-space have three degrees of freedom in each node, giving six degrees of freedom in total. This includes four translational, two along the centre axis, two perpendicular and two rotational degrees of freedom. This in turn means that the element stiffness matrix for a 2D-frame element can be taken as a combination of one bar action stiffness matrix (equation 2.25) and one beam action element stiffness matrix (equation 2.54) yielding the following:

$$\mathbf{K}^e = \frac{E}{L}\begin{bmatrix} A & 0 & 0 & -A & 0 & 0 \\ 0 & \frac{12I}{L^2} & \frac{6I}{L} & 0 & -\frac{12I}{L^2} & \frac{6I}{L} \\ 0 & \frac{6I}{L} & 4I & 0 & -\frac{6I}{L} & 2I \\ -A & 0 & 0 & A & 0 & 0 \\ 0 & -\frac{12I}{L^2} & -\frac{6I}{L} & 0 & \frac{12I}{L^2} & -\frac{6I}{L} \\ 0 & \frac{6I}{L} & 2I & 0 & -\frac{6I}{L} & 4I \end{bmatrix} \tag{2.65}$$

In the same manner, the element load vector for distributed loads can be taken as a combination of equation 2.27 and equation 2.56 giving:

$$\mathbf{f}_l^e = \frac{L}{2}\begin{bmatrix} q_x \\ q_y \\ \frac{q_y L}{6} \\ q_x \\ q_y \\ -\frac{q_y L}{6} \end{bmatrix} \tag{2.66}$$

2.1.1.5 3D-Frame elements

As for the 2D case, a frame element in 3D-space can be taken as a combination of the different stiffness modes. In this case nodes, though, each nodes has six degrees of freedom, yielding a total of twelve. To get the stiffness matrix for this kind of element one can, as for the 2D-case, combine the stiffness matrices from bar action (equation 2.25), beam action two times (equation 2.54) and torsion (equation 2.62) giving the following:

$$\mathbf{K}^e = \begin{bmatrix} \frac{EA}{L} & 0 & 0 & 0 & 0 & 0 & -\frac{EA}{L} & 0 & 0 & 0 & 0 & 0 \\ 0 & \frac{12EI_z}{L^3} & 0 & 0 & 0 & \frac{6EI_z}{L^2} & 0 & -\frac{12I}{L^3} & 0 & 0 & 0 & \frac{6EI_z}{L^2} \\ 0 & 0 & \frac{12EI_y}{L^3} & 0 & -\frac{6EI_y}{L^2} & 0 & 0 & 0 & -\frac{12EI_y}{L^3} & 0 & -\frac{6EI_y}{L^2} & 0 \\ 0 & 0 & 0 & \frac{GK_v}{L} & 0 & 0 & 0 & 0 & 0 & -\frac{GK_v}{L} & 0 & 0 \\ 0 & 0 & -\frac{6EI_y}{L^2} & 0 & \frac{4EI_y}{L} & 0 & 0 & 0 & \frac{6EI_y}{L^2} & 0 & \frac{2EI_y}{L} & 0 \\ 0 & \frac{6EI_z}{L^2} & 0 & 0 & 0 & \frac{4EI_z}{L} & 0 & -\frac{6EI_z}{L^2} & 0 & 0 & 0 & \frac{2EI_z}{L} \\ -\frac{EA}{L} & 0 & 0 & 0 & 0 & 0 & \frac{EA}{L} & 0 & 0 & 0 & 0 & 0 \\ 0 & -\frac{12EI_z}{L^3} & 0 & 0 & 0 & -\frac{6EI_z}{L^2} & 0 & \frac{12I}{L^3} & 0 & 0 & 0 & -\frac{6EI_z}{L^2} \\ 0 & 0 & -\frac{12EI_y}{L^3} & 0 & \frac{6EI_y}{L^2} & 0 & 0 & 0 & \frac{12EI_y}{L^3} & 0 & \frac{6EI_y}{L^2} & 0 \\ 0 & 0 & 0 & -\frac{GK_v}{L} & 0 & 0 & 0 & 0 & 0 & \frac{GK_v}{L} & 0 & 0 \\ 0 & 0 & -\frac{6EI_y}{L^2} & 0 & \frac{2EI_y}{L} & 0 & 0 & 0 & \frac{6EI_y}{L^2} & 0 & \frac{4EI_y}{L} & 0 \\ 0 & \frac{6EI_z}{L^2} & 0 & 0 & 0 & \frac{4EI_z}{L} & 0 & -\frac{6EI_z}{L^2} & 0 & 0 & 0 & \frac{4EI_z}{L} \end{bmatrix} \tag{2.67}$$

As for the case, the load vector can be taken as a combination of the different partial load vectors. A combination of equation 2.27, equation 2.56 two times and equation 2.64 gives:

$$\mathbf{f}_l^e = \frac{L}{2}\begin{bmatrix} q_x \\ q_y \\ q_z \\ q_\omega \\ -\frac{q_z L}{6} \\ \frac{q_y L}{6} \\ q_x \\ q_y \\ q_z \\ q_\omega \\ \frac{q_z L}{6} \\ -\frac{q_y L}{6} \end{bmatrix} \tag{2.68}$$

2.1.1.6 Transformation to global coordinates

The elements described in previous sections are all described in their local coordinates. That is, the x direction for the elements runs along the centre line of the element with local the y- and z-axes perpendicular to the centre line. For a system with multiple elements, the local coordinate system for each element don't need to coincide. To be able to solve a system like that, one or more of the local element coordinate systems, with stiffness matrices and force vectors, needs to be transformed. This is done by transforming all matrices and vectors into a chosen global coordinate system, often the global xyz.
For a 2D-Frame element the transformation matrix is defined as [Dahlblom, Olsson (2010)]:

$$\mathbf{G} = \begin{bmatrix} n_{x\overline{x}} & n_{y\overline{x}} & 0 & 0 & 0 & 0 \\ n_{x\overline{y}} & n_{y\overline{y}} & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & n_{x\overline{x}} & n_{y\overline{x}} & 0 \\ 0 & 0 & 0 & n_{x\overline{y}} & n_{y\overline{y}} & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \tag{2.69}$$

and for a 3d-Frame element:

$$\mathbf{G} = \begin{bmatrix} n_{x\overline{x}} & n_{y\overline{x}} & n_{z\overline{x}} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ n_{x\overline{y}} & n_{y\overline{y}} & n_{z\overline{y}} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ n_{x\overline{z}} & n_{y\overline{z}} & n_{z\overline{z}} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & n_{x\overline{x}} & n_{y\overline{x}} & n_{z\overline{x}} & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & n_{x\overline{y}} & n_{y\overline{y}} & n_{z\overline{y}} & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & n_{x\overline{z}} & n_{y\overline{z}} & n_{z\overline{z}} & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & n_{x\overline{x}} & n_{y\overline{x}} & n_{z\overline{x}} & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & n_{x\overline{y}} & n_{y\overline{y}} & n_{z\overline{y}} & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & n_{x\overline{z}} & n_{y\overline{z}} & n_{z\overline{z}} & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & n_{x\overline{x}} & n_{y\overline{x}} & n_{z\overline{x}} \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & n_{x\overline{y}} & n_{y\overline{y}} & n_{z\overline{y}} \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & n_{x\overline{z}} & n_{y\overline{z}} & n_{z\overline{z}} \end{bmatrix} \tag{2.70}$$

where:

$$\begin{aligned} n_{x\overline{x}} &= \mathbf{n}_{\overline{x}} \bullet \mathbf{n}_x; \quad n_{y\overline{x}} = \mathbf{n}_{\overline{x}} \bullet \mathbf{n}_y; \quad n_{z\overline{x}} = \mathbf{n}_{\overline{x}} \bullet \mathbf{n}_z; \\ n_{x\overline{y}} &= \mathbf{n}_{\overline{y}} \bullet \mathbf{n}_x; \quad n_{y\overline{y}} = \mathbf{n}_{\overline{y}} \bullet \mathbf{n}_y; \quad n_{z\overline{y}} = \mathbf{n}_{\overline{y}} \bullet \mathbf{n}_z; \\ n_{x\overline{z}} &= \mathbf{n}_{\overline{z}} \bullet \mathbf{n}_x; \quad n_{y\overline{z}} = \mathbf{n}_{\overline{z}} \bullet \mathbf{n}_y; \quad n_{z\overline{z}} = \mathbf{n}_{\overline{z}} \bullet \mathbf{n}_z; \end{aligned} \tag{2.71}$$

where:

$$\begin{aligned}
\mathbf{n}_{\overline{x}} &= \text{unit vector in the direction of local x-axis} \\
\mathbf{n}_{\overline{y}} &= \text{unit vector in the direction of local y-axis} \\
\mathbf{n}_{\overline{z}} &= \text{unit vector in the direction of local z-axis} \\
\mathbf{n}_{x} &= \text{unit vector in the direction of global x-axis} \\
\mathbf{n}_{y} &= \text{unit vector in the direction of global y-axis} \\
\mathbf{n}_{z} &= \text{unit vector in the direction of global z-axis}
\end{aligned} \tag{2.72}$$

To ease reading in the previous sections the overline notation has been omitted. In the following equations overline notation indicates local coordinates. The relation between local and global coordinates for the degrees of freedom is:

$$\overline{\mathbf{a}}^e = \mathbf{G}\mathbf{a}^e \tag{2.73}$$

with a similar relation for the force vectors:

$$\mathbf{f}_l^e = \mathbf{G}^T\overline{\mathbf{f}}_l^e \tag{2.74}$$

and:

$$\mathbf{f}_b^e = \mathbf{G}^T\overline{\mathbf{f}}_b^e \tag{2.75}$$

The equation systems found in eq 2.16 and eq 2.43 are as previously mentioned in local coordinates and are now denoted:

$$\overline{\mathbf{K}}^e\overline{\mathbf{a}}^e = \overline{\mathbf{f}}_b^e + \overline{\mathbf{f}}_l^e \tag{2.76}$$

Inserting eq 2.73, eq 2.74 and eq 2.75 in eq 2.76 gives the system in global coordinates:

$$\mathbf{K}^e\mathbf{a}^e = \mathbf{f}_l^e + \mathbf{f}_b^e \tag{2.77}$$

where:

$$\mathbf{K}^e = \mathbf{G}^T\overline{\mathbf{K}}^e\mathbf{G} \tag{2.78}$$

With this, the degrees of freedom, together with stiffness matrices and force vectors, has been transformed into global coordinates.

2.1.2 Springs

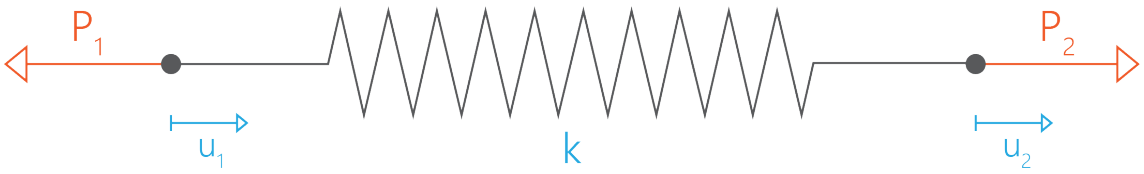


Figure 2.4: A one dimensional translational spring connecting two degrees of freedom.

Springs are the simplest possible structural elements. They are a purely discrete representation of a one dimensional stiffness between two degrees of freedom. The basic equation of a spring is:

$$N = k\delta \tag{2.79}$$

where:

$$\begin{aligned} k = &\quad \text{Spring stiffness}[N/m] \\ N = &\quad \text{Force in the spring}[N] \\ \delta = &\quad \text{Deformation of the spring}[m] \end{aligned}$$

By recognising that:

$$\delta = u_2 - u_1$$

and

$$\begin{aligned} P_1 = &\quad N \\ P_2 = &\quad -N \end{aligned}$$

the equation can be written on matrix form:

$$\begin{bmatrix} k & -k \\ -k & k \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \end{bmatrix} = \begin{bmatrix} P_1 \\ P_2 \end{bmatrix} \tag{2.80}$$

Or

$$\mathbf{K}^e \mathbf{a}^e = \mathbf{f}^e \tag{2.81}$$

And so, the spring element stiffness matrix is written:

$$\mathbf{K}^e = k \begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix} \tag{2.82}$$

The process for a rotational spring can be solved analogically, but with the difference that a rotational spring has the stiffness:

$$k = \quad \text{Spring stiffness}[\text{Nm/rad}]$$

2.1.3 Connection stiffness

To make a finite element model work all elements must interact with each other. For frame elements this interaction takes place in their nodes, when a node is shared with one or several other elements. The typical type of connection is either fixed or pinned. For a fixed connection all degrees of freedom is shared by the elements connected to the node. For a pinned connection all translational degrees of freedom are shared while the rotational degrees of freedom are separate for each element.

(a) Fixed rotations **(b)** Free rotations

Figure 2.5: An element connection in 2d, with varying connection properties.

This way of connecting elements is 'binary' in this regard as it forces all degrees to be either completely connected (the same degree of freedom) or completely free (see fig. 2.5). To overcome this it is possible to model the connection between the degrees of freedom as springs. This can be done for each degrees of freedom separately, modelling all or just some of them as spring restrained connections. Doing so lets one control the nodal stiffness for each connection, which in turn can be used to for example compare different connection details of a structure to see how they affect the overall behaviour.

2.2 Canonical stiffnesses

The term canonical stiffness was first coined by Niels Ottosen, but is also described by Olsson and Thelin [Olsson, Thelin (2003)][1]. In their paper they showed that by assuming a linear load-displacement relationship [Olsson, Thelin (2003)]:

$$\mathbf{f} = \lambda \mathbf{a} \tag{2.83}$$

the response of the structure can be rewritten as:

$$\mathbf{K}\mathbf{a} = \mathbf{f} = \lambda \mathbf{a} \tag{2.84}$$

and therefore:

$$(\mathbf{K} - \lambda \mathbf{I})\mathbf{a} = \mathbf{0} \tag{2.85}$$

from which the eigenvalues λ_i and corresponding eigenvectors $\mathbf{a}_i$ can be determined. If the eigenvectors are normalized, it means that:

$$\mathbf{a}_i^T \mathbf{a}_i = 1 \tag{2.86}$$

Premultipication of eq 2.84 $\mathbf{a}_i^T$ gives:

$$\mathbf{a}_i^T \mathbf{K} \mathbf{a}_i = \lambda_i \tag{2.87}$$

where:

$$\mathbf{a}_i^T \mathbf{K} \mathbf{a}_i = 2\mathrm{U}_i \tag{2.88}$$

where U_i is the strain energy for mode i. This mean that the eigenvectors correspond to the strain energy, $\mathrm{U}_i = \lambda_i/2$, and by using the principle of virtual work the conclusion can be drawn that the mode which produces the least amount of strain energy is the mode most sensitive to external loading (which is in turn is found using the original assumption, equation 2.83).

By sorting the eigenmodes by their corresponding eigenvalues the most sensitive load patterns for the structure could be obtained. By maximising the lowest eigenvalue the overall response of the structure could be said to be increasingly stiff. One thing that may be noted here is that the lowest eigenmode may be irrelevant to the expected behaviour of the structure (in terms of loading).

[1] The term *canonical stiffness* isn't used in this paper, but is used in [Dahlblom, Olsson (2010), p. 152-153]. In [Olsson, Thelin (2003)] the concept is called as *static eigenmodes*. The concept is also described in [Olsson (2006), p. 58-59]

2.3 Structural Optimisation

2.3.1 Introduction to structural optimisation

Structural optimisation is a field within structural engineering which aims to find the 'best' solutions to design problems. What the 'best' solutions are may vary from case to case and could be for example minimizing weight, maximising stiffness, minimising cost or environmental impact.

In their 2009 book *An Introduction To Structural Optimisation*, Christensen and Klarbring name three types of structural optimisation, which are defined by what variables they use [Christensen, Klarbring (2009)]. They are:

- **Size optimisation**
  Size optimisation uses structural dimensions as input variables, for example cross section area.
- **Shape optimisation**
  Shape optimisation uses the shape of structural elements as its input variables, for example by tapering elements.
- **Topology optimisation**
  Topology optimisation changes the connectivity between finite elements to form load paths. In a discrete system of structural mechanics (for instance a truss) this can be achieved by letting the section property of some elements be zero, eliminating them from the calculation and possibly also changing the stiffness matrix by removing some degrees of freedom.

They also state a general mathematical formulation

$$\begin{cases} \text{minimise} & f(x,y) \quad \text{w.r.t. } x, y \\ \text{subject to} & \begin{cases} \text{design constraints on } x \\ \text{behavioural constraints on } y \\ \text{equilibrium constraint} \end{cases} \end{cases} \tag{2.89}$$

Where

- f *Objective function.* A function which returns a value that sets a grade on how good a design is. It is often chosen so that a low value is preferable to a high value.
- x *Design variable.* A variable which sets the state of the problem, and which is used to change the design of the structure, for example element height, width or material.
- y *State variable.* A variable which describes a response of a structure and thus cannot be set manually, $y = g(x)$. Examples are stresses, strains, reactions and displacements.

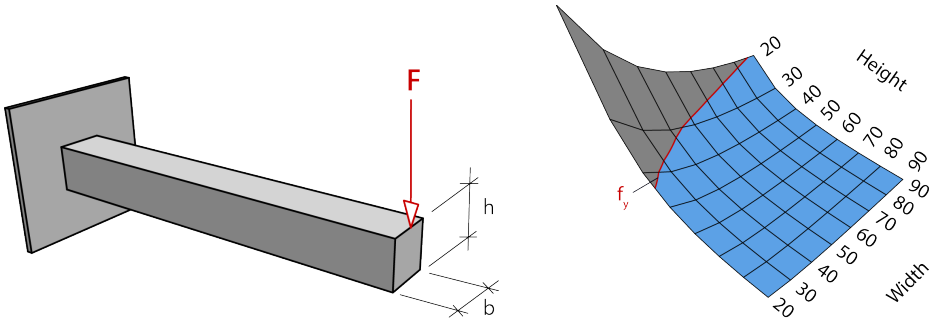
2.3.1.1 Design space and solution space

Two terms that often are used when talking about structural optimisation are *design space* and *solution space*, where the design space is all possible solutions and the solution space the subset of this space containing all acceptable solutions. As an example, consider the beam in figure 2.6a. In the example the only design variables are the height h and width b. These variables each have a set of discrete sizes that they may bet set to, ranging from 20 mm to 90 mm (with 10 mm increments). This means that each dimension has 8 different options, rendering a total of 64 solutions:

		Height							
		20	30	40	50	60	70	80	90
Width	20	1500	667	375	240	167	122	94	74
	30	1000	444	250	160	111	82	62	49
	40	750	333	187	120	83	61	47	37
	50	600	267	150	96	67	49	37	30
	60	500	222	125	80	56	41	31	25
	70	429	190	107	68	48	35	27	21
	80	375	167	94	60	42	31	23	19
	90	333	148	83	53	37	27	21	16

Table 2.1: The maximum resulting bending stress in the element σ [MPa] for varying width and height assuming elastic behaviour and $E = 210$ [GPa].

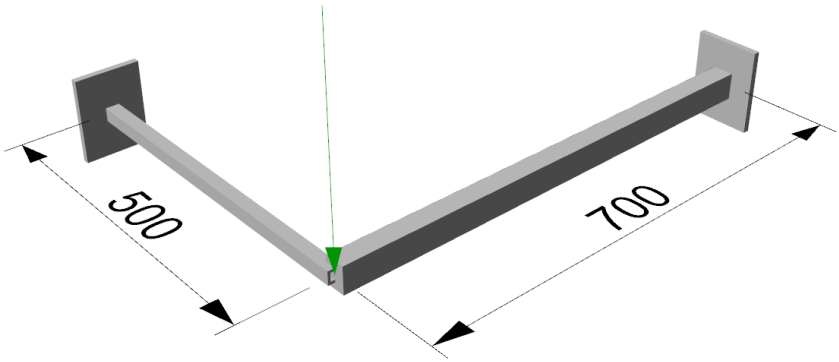
This range of results is what is sometimes called the 'design space' (although it also has been called 'search space' and 'phase space'). In the design space the variables can be thought of as dimensions [Bedney (2016)]. The 'solution space' is the subspace of the design space which consist of 'acceptable' solutions. What is deemed acceptable is chosen from case to case. In fig. 2.6b, the entire surface represents the design space. If the yield stress is taken as $f_y = 335$ [MPa], the surface can be divided into acceptable and unacceptable solutions. Here, the blue area is the acceptable solutions, i.e. the solution space.



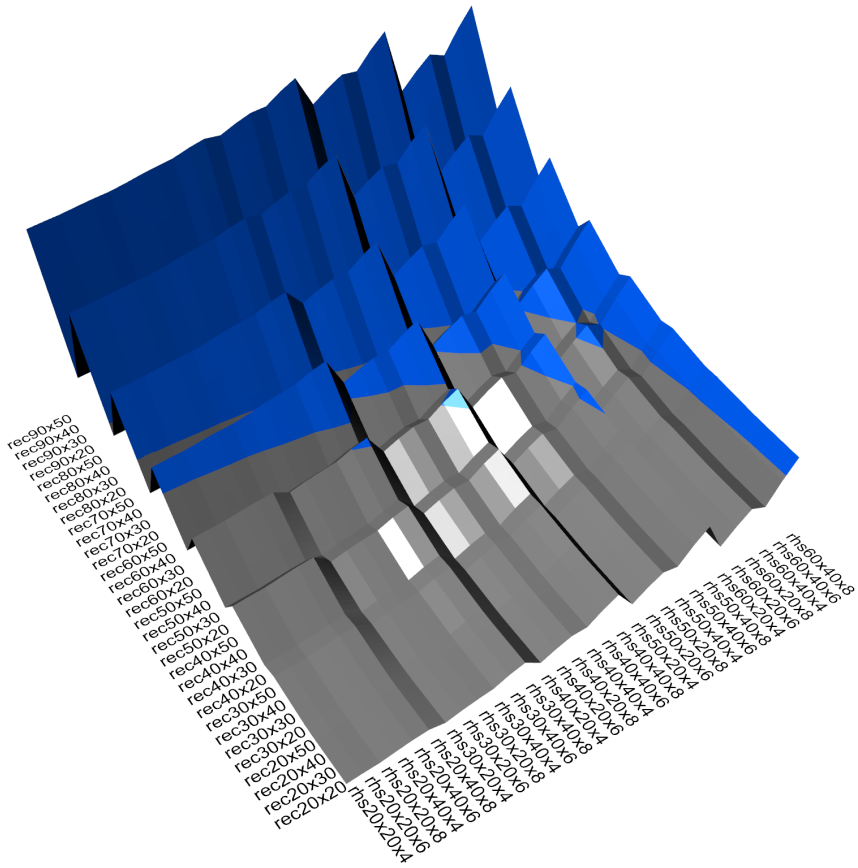
(a) A simple problem of a rectangular solid beam with fixed - free boundary conditions and a transversal point load of $F = 2000N$.

(b) A three-dimensional illustration of table 2.1, showing maximum stress. The red line represents the yield stress ($f_y = 275$ [MPa]) of the material and therefore the blue area shows the solutions that will not yield.

Figure 2.6: A simple example to illustrate the design space and solution space.



(a) Two beams fixed at the ends and rigidly connected in a free node. The RHS element is 500 mm long and the solid rectangular element is 700 mm long. A point load of 1000 N is applied in the downward direction on the free node. Gravity is also considered.



(b) The reciprocal of the maximum utilisation in the system ($z = \eta_{max}^{-1}$) plotted against varying section properties. The blue coloured area represents $1/\eta_{max} > 1$, ($\Rightarrow \eta_{max} < 1$), which is the solution space in this case.

Figure 2.7: An illustration of a structural engineering optimisation problem for two elements of discrete section properties.

In fig. 2.6 an example for two variables have been illustrated where the z coordinate in fig 2.6b represents the maximum stress in the element. This is a quite basic example, but if one would consider adding another element (see fig. 2.7a), the number of variables rapidly increases. In the case of fig 2.7a the variables are 5 (width and height of both sections, plus thickness of the RHS), more than can be visualised in 3d space. The solution space can instead be plotted for a set of discrete sections, which implicates that the behaviour will be more non-linear. In fig. 2.7b the maximum utilisation of the system has been (reciprocally) plotted for a set of section combinations, with the height coordinate z being calculated to ($z = 1/\eta_{max}$).

Having obtained the solution space, the task is now to find the optimum solution. In this case it would be easy to find, since the solution space is sufficiently small. In terms of minimal weight, the optimum solution consists of a rectangular section of 20x20 mm and an RHS of 60x40x8 mm. This results in a structural weight of 8,42 kg.

2.3.1.2 The need for heuristics

Heuristics are methods of problem solving that aim to find a 'good enough' solution rather than finding the global optimum. The two main reasons to do this are that it is more efficient in terms of computation time and that the heuristic method is easier to use. It can easily be shown why it is necessary in the field of structural design to use a heuristic method. Bear in mind that the normal design procedure of choosing some sections and evaluating them also is a heuristic, since the engineer doesn't know that the chosen sections are the 'optimal' ones. Consider the example from fig. 2.7a and fig. 2.7b, it can be seen that for two elements with 30 and 32 section properties respectively, the design space is made up of 960 solutions. For any structure the size of the design space is calculated as:

$$\prod_{i=1}^{n_e} n_{s,i} \tag{2.90}$$

Or simplified, if all elements have the same set of sections to choose from:

$$n_s^{n_e} \tag{2.91}$$

Where

n_e = is the number of elements in the structure
n_s = is the number of possible sections

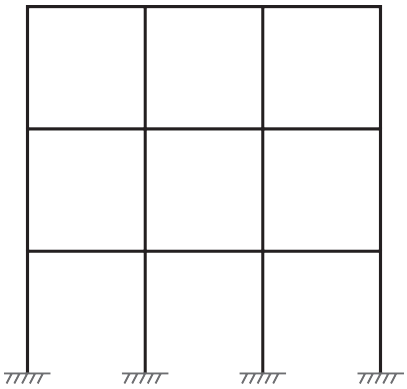


Figure 2.8: Simple frame structure consisting of 3 bays and 3 storeys. The total number of elements is 21, but if each element has 30 possible section properties, the number of solutions is a staggering $10.46 \cdot 10^{27}$!

If we consider a simple 2d structure of 3 bays and 3 storeys (see fig. 2.8), with a set of 30 sections to choose from, the size of the design space is $n_s^{n_e} \Rightarrow 30^{21} = 10.46 \cdot 10^{27}$ solutions large! Even if it would be possible to process a billion results per second it would still take well over 300 trillion years to process them all.

Even though that in a real design scenario not all solutions would have to be evaluated (if a solution where all the sections in the structure are the second stiffest produces an acceptable result, it isn't really necessary to check for the stiffest sections), it is still difficult to reduce the design space significantly. This of course states the need for more efficient search algorithms.

2.4 Optimisation Algorithms

This section will cover the theory of the various optimisations algorithms that are studied in this thesis. They are as following:

- *Iterative section sizer*
  Analysing the structure and increasing the section sizes if they fail.
- *Section rotator*
  Aligning the normal of elements to their principal bending direction.
- *Mode shape optimisation*
  Using canonical stiffnesses to find sensitive deformation patterns.
- *Genetic algorithms*
  Using the 'survival of the fittest' to generate optimum solutions.

2.4.1 Iterative section sizer

In the introduction to this thesis a common design procedure was described, where the engineer would identify a overutilised member of a structure and increasing its size for it to pass a section check. Likewise can underutilised members have their section size reduced until all elements have a utilisation of about 100%. This process can of course be automated with varying level of sophistication and may seem to be a good optimisation process due to its simplicity. It does produce a fully stressed design (FSD), but as shown by Mueller and Burns [Mueller, Burns (2001)], an FSD isn't necessarily also the optimal solution in terms of minimum weight, nevertheless in other optimisation aspects. On the contrary, they showed that one structure may have several FSDs, some of which are not even *possible* to find using an iterative section sizer. They call these designs '*repelling* fully stressed designs'.

Because of the many possible resulting designs, the conclusion can be drawn that the initial design is of great importance when dealing with structurally indeterminate structures. At the s

2.4.2 Section Rotator

The direction of the normal, or local z-axis, can have a very large impact on the behaviour of a three dimensional beam element. A simple example is to take a cantilever beam with a massive rectangular section as an example. It can be noted that this statically determinate "system" will have a constant inner inner force pattern, even though the stresses will change. A statically indeterminate system might change inner force patterns if the cross sections of elements are rotated.

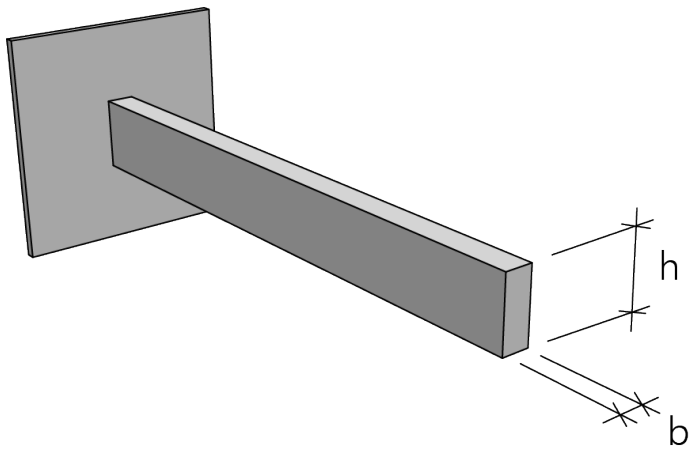


Figure 2.9: A cantilever beam with a rectangular cross section.

The cross section have the following properties:

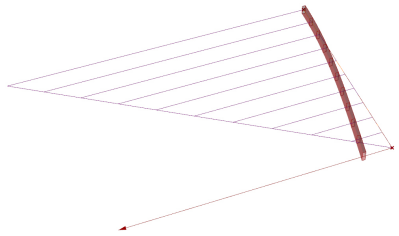
$$\begin{aligned} h &= 150\ mm \\ b &= 50\ mm \\ I_{\overline{y}} &= 14\,062\,500\ mm^4 \\ I_{\overline{z}} &= 1\,562\,500\ mm^4 \end{aligned} \tag{2.92}$$

It can be seen that the section has significantly higher stiffness for rotations around the local y-axis compared to the local z-axis, where the moment of inertia around the y-axis is roughly ten times bigger than around the z-axis. This makes a huge difference for it's capacity for moments and forces from different directions, and hence the orientation of the member can be critical depending on the load situation.

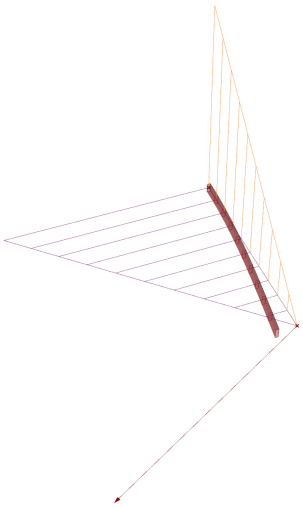
In this example the beam has the following properties:

- Length $L = 3\,m$
- Young's modulus $E = 210\,GPa$
- Yield stress $f_y = 275\,MPa$

The cantilever is loaded with a point load of $P = 14\,kN$ applied at its free end. The direction of the normal (local z-axis) of the beam is set to the global z-axis. The following results are acquired depending on the direction of the force:



(a) Point load in the direction of the local y-axis $\{0, -P, 0\}$. Graph shows moment around the local z-axis (minor axis bending). Moment around the local y-axis (major axis bending) is zero for this load case. Utilization of 2.62 at the support.

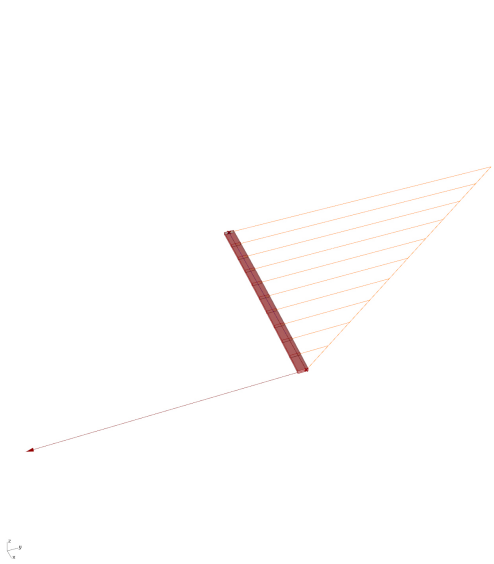


(b) Point load in the direction of a combination of the local y-axis and z-axis $\{0, -P/\sqrt{(2)}, -P/\sqrt{(2)}\}$. Graph shows moment around the local z-axis (minor axis bending) in purple and moment around the local y-axis (major axis bending) in orange. Utilization of 2.47 at the support.

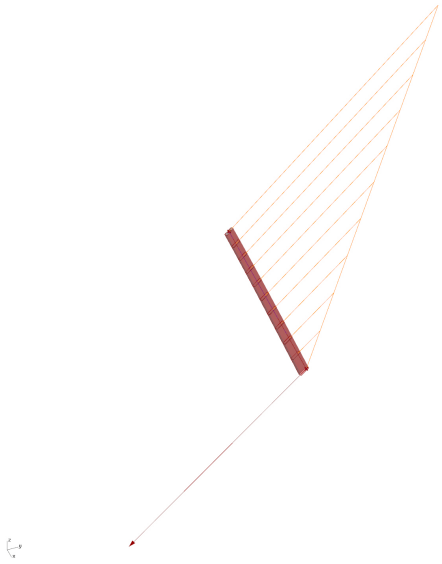
(c) Point load in the direction of the local z-axis $\{0, 0, -P\}$.Graph shows moment around the local y-axis (major axis bending). Moment around the local z-axis (minor axis bending) is zero for this load case. Utilization of 0.87 at the support.

Figure 2.10

As can be seen in fig. 2.10c, the section is well equipped to handle the applied force, with a utilization well below 1, while the other cases in fig. 2.10a and 2.10b gives utilizations well above 2, and thereby failures. This clearly illustrates that the orientation of the member in relation to the orientation to the load can be extremely important for certain situations.
In a real life situation one can seldom control the direction of the load. It is, though, for most cases possible to control the orientation of the member if no detailing restrain or similar are present. For the examples in fig. 2.10a and fig. 2.10b the section could be rotated 90° and 45° respectively to obtain the following:



(a) Point load in the direction of the local y-axis $\{0, -P, 0\}$.The direction of the element normal is in this case, compared to fig. 2.10a flipped 90° and aligned with the force. Graph shows moment around the local y-axis (major axis bending). Moment around the local z-axis (minor axis bending) is zero for this load case. Utilization of 0.87 at the support.



(b) Point load in the direction of the local y-axis $\{0, -P/\sqrt{(2)}, -P/\sqrt{(2)}\}$.The direction of the element normal is in this case, compared to fig. 2.10b flipped 45° and aligned with the force. Graph shows moment around the local y-axis (major axis bending). Moment around the local z-axis (minor axis bending) is zero for this load case. Utilization of 0.87 at the support.

Figure 2.11

Naturally the alignment of the strong direction of the section to the force for this case gives better results in the form of lower utilization as can be seen in fig. 2.11a and fig. 2.11b and give the same results as in fig. 2.10c.
For a more general case, aligning the sections with loads applied directly on them do not have to give the best results, as all members are affected by the rest of the structure and some members do not even have to have forces acting directly on them. One approach to make this more general is to align the normal of the section to the "principle moment" of the worst stresses section. The principle moment is calculated as:

$$M_{principle} = \sqrt{M_{\overline{y}}^2 + M_{\overline{z}}^2} \tag{2.93}$$

The direction of this new normal is found by:

$$\overline{\mathbf{z}}_{new} = \overline{\mathbf{z}} M_{\overline{y}} - \overline{\mathbf{y}} M_{\overline{z}} \tag{2.94}$$

Where $M_{\overline{y}}$ and $M_{\overline{z}}$ are taken from the section with the highest principle moment $M_{principle}$.

2.4.3 Mode shape optimisation

In the 2003 paper *Use of Static Eigenmodes in Mechanical Design* by Karl-Gunnar Olsson and Carl Thelin [Olsson, Thelin (2003)] are statical eigenmodes (see section 2.2) used as a design tool. An example using an armchair is used (see fig. 2.12).

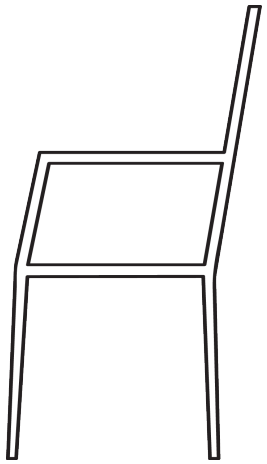


Figure 2.12: An armchair used as a case study. The chair is statically indeterminate in 2d, with the load having the possibility to transfer through either the armrest or the seat.
Adopted from [Olsson, Thelin (2003)]

The method of canonical stiffnesses is used through finding the eigenmodes and sorting them based on their corresponding eigenvalues. Thereafter are the eigenmodes of the lowest corresponding strain energy analysed. The eigenmodes were not only considered with regard to their eigenvalue, but also to an imagined load case that the eigenmode corresponds to. For instance, the "tilting mode" (see image 2.13) is considered usually dangerous, most likely because it corresponds to the second most common load case: leaning back on the chair (the most common being simply

sitting on the chair, loading it only vertically). These dangerous modes are more important to avoid.

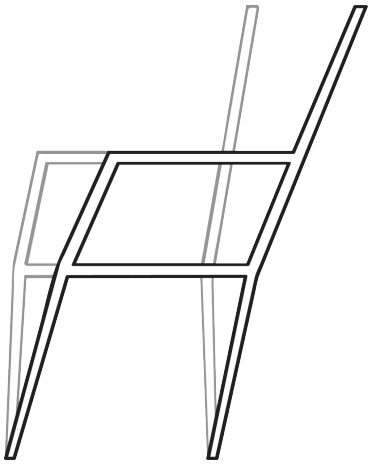


Figure 2.13: An especially dangerous deformation for the armchair.
Adopted from [Olsson, Thelin (2003)]

In conclusion, the method is intended to give the designer information about the naturally weak modes of deformation in a structure. By then tending to the most dangerous modes of deformation an effective material usage can be obtained. It is concluded that the method is especially useful when the loads on the structure are unknown or hard to envisage.

2.4.4 Genetic algorithms

Genetic algorithms is a heuristic based on the theory of evolution. The allegory is 'survival of the fittest' in the sense that solutions are selected based on a 'fitness' value, and allowed to reproduce. There are several descriptions of the method, one of which is presented by Melanie Mitchell in her 1999 book [Mitchell (1999)]. She presents the following algorithm:

1. Generate a randomised *population* of i genomes
2. Calculate the fitness of each genome
3. Generate i number of new genomes ('offspring')
    - Select two of the existing genomes to be parents, based on fitness (higher fitness means more likely to get selected).
    - With a certain probability, cross over the genomes of the parents at a randomly selected point.
    - Form two offspring.
    - Mutate the offsprings with a certain probability.
4. Let the children form the new generation
5. Go to step 2

The algorithm is run until a number of stagnated fitness of predetermined length is reached.

The strength of the genetic algorithms are that they require very little analysis of complex problems and can still come up with reasonably good results. They simply analyse the problem for a number of randomised initial settings and then tries to improves the results. This means that, given the right settings, GAs are well suited to handle non-linear problems. The setbacks are that a large number of calculations are required for finding a reasonable result, which means a high calculation time.

There are several other methods available in the family of 'generic' optimisation techniques. A good summary is presented by Peter Debney in an article in The Structural Engineer [Bedney (2016)].

3
Method

To recapitulate, the aim of this thesis have been to investigate different optimisation methods in a parametric environment. In order to achieve this, the following major steps have been taken (each described in more detail in the following sections):

1. Developing a finite element plugin for Grasshopper that contains three separate parts:
   - A finite element package called *CIFem* in C++
   - A wrapper project *CIFem_Wrapper* in C++ / CLI that enables *CIGull* to access CIFem.
   - The actual plugin for Grasshopper, called *CIGull*, that uses CIFem to perform structural engineering calculations and serving as a platform for creating structural engineering optimisation algorithms.
2. Developing optimisation methods and implementing them in *CIFem*.
3. Designing a set of test structures in order to test the optimisation methods

The individual steps will be described in the following chapter.

3.1 Project overview

A well-known principle of object oriented design is to keep code modular. This allows a particular piece of code to do what it does best, without too many case-specific additions, and promotes expandability. This goes for individual classes within a library, but is also true for the libraries themselves (known as the Adapter Pattern [Freeman, Freeman (2004)]). In the case of this thesis work three interconnected libraries (*CIFem*, *CIFem_wrapper*, *CIGull*) have been created, which work with two available groups of libraries. These external libraries actually consist of several parts, but can be collated into two groups: Rhinoceros and Armadillo. An overview of the the different parts can be seen in fig. 3.1

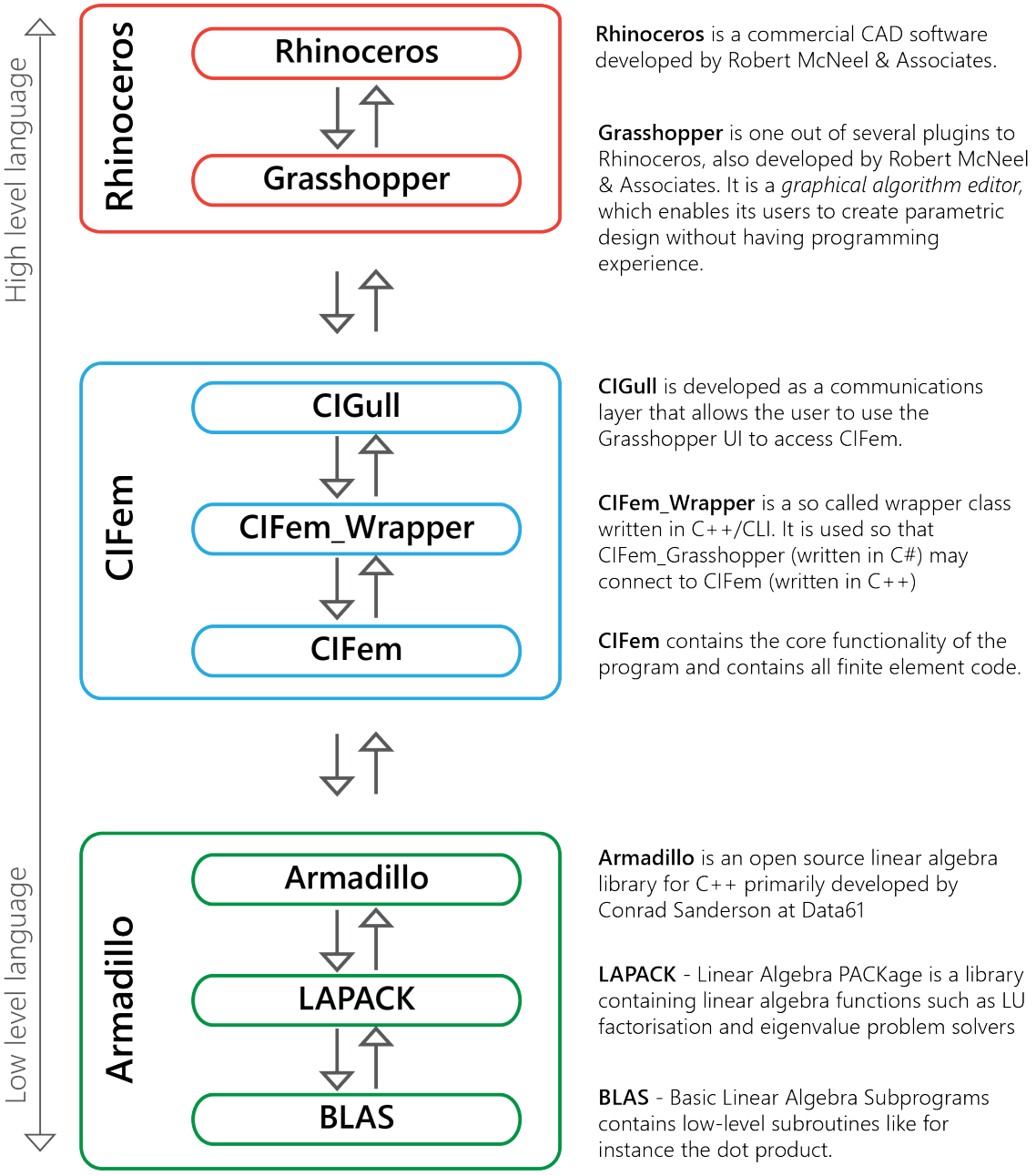


Figure 3.1: A high level flow chart of information between the different code libraries. Rhino is used as the user interface, taking commands and displaying results. CIFem is the product of this thesis, adding a finite element package as a plugin to grasshopper. Armadillo is a linear algebra package which is required for CIFem to perform its calculations. For further reading about Rhino, consult [Robert McNeel & Associates (2016)]. For further reading on Armadillo, consult [Sanderson (2010)]

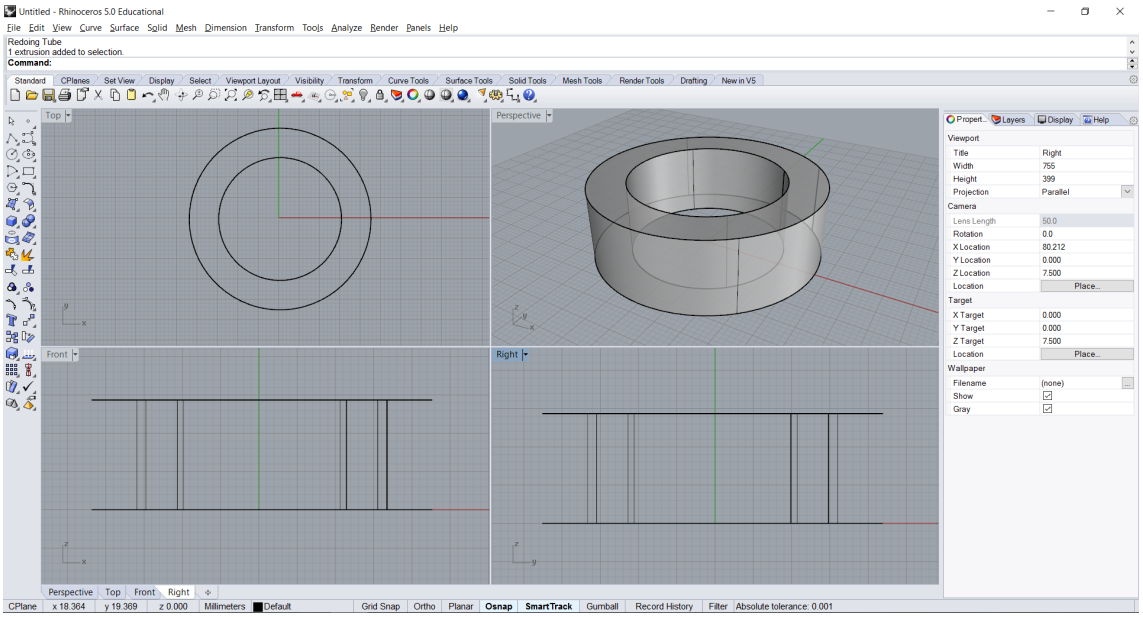


Figure 3.2: The workspace in Rhinoceros 5, including the four viewports (middle), modelling tools (left), properties (right) and menu and command line (top).

3.1.1 Rhinoceros and Grasshopper

The Rhinoceros part of the program is used as the front end, where the user interacts with the model. Rhinoceros is a commercial CAD software developed by Robert McNeel & Associates. It has all the capabilities that one expects of a CAD software, but also a capability to work with NURBS (Non-Uniform Rational B-Splines) curves and surfaces [Robert McNeel & Associates (2016)]. Fig. 3.2 shows the work space in Rhino.

Rhino has a public Application Programming Interface (API) which has allowed the community to create numerous plugins for various different purposes. One of the most recognised ones is Grasshopper, which is a graphical algorithm editor [Davidson (2016)] developed by David Rutten for Robert McNeel & Associates [Tedeschi (2011)]. It is a tool that allows its users to interact with Rhinoceros through a visual programming language. Grasshopper can be used together with Rhino, or just by using Rhino as a viewport. Generally are all actions that are possible in Rhino also possible in Grasshopper. Fig. 3.3 shows the work space in Grasshopper. A good introduction to Grasshopper is available from Delft University of Technology, see [TU Delft (2015)]. Grasshopper has, like Rhino, a public API, which has generated as many or even more plugins.

3.1.2 Creating plugins for Grasshopper

A possible reason for the popularity of Grasshopper is its relatively gentle learning curve combined with its almost endless possibilities of 3d modelling. In the Grasshopper environment, the wiring is done linearly, meaning that loops are not possible. The easiest method of resolving that problem is to use the built-in code

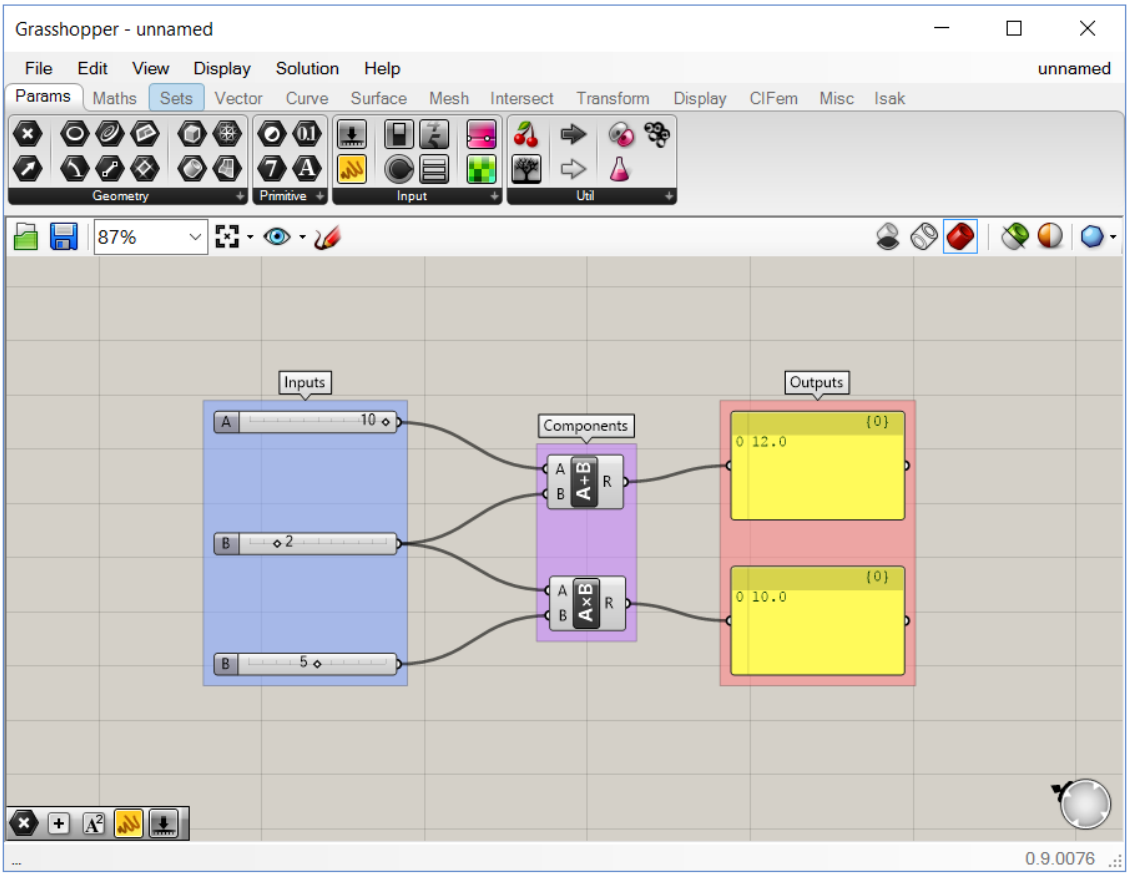


Figure 3.3: The workspace of Grasshopper. Inputs (blue) are connected to components (purple) using wires. The outputs (red) are then wired in the same way.

components in grasshopper. There are two available, one for Visual Basic (VB) and one for C#. These components offer great flexibility and may be saved as custom components to be reused.

If one intends to create something that requires more of a structure it is possible to create a plugin instead. The plugin consist of a Grasshopper Assembly file (.gha) which really is a .dll, but has had its file extension changed so that Grasshopper can find and load it. This .dll may be made in any which way, but for this thesis work it has been written in Microsoft Visual Studio Community 2015, an integrated development environment (IDE). Using an IDE rather than the built-in editor offers several advantages, wherein the most important ones are:

- Possibility to reference other libraries. In the case of this thesis accessing Armadillo has been crucial, and not possible to do from within Grasshopper.
- Object oriented design. It isn't possible within Grasshopper to create classes.
- Code completion and code refactoring abilities.
- Speed. Compiling code at compile time rather than at run-time means that it only happens once and not while running, which improves speed.

In order to access some written code, one must implement the Grasshopper API. There are some quite strict formatting requirements that need to be fulfilled in order for the code to be possible to be used and its information passed around. Fortunately, there is a very good software development kit (SDK) available for download from Grasshopper or from:
http://s3.amazonaws.com/mcneel/grasshopper/1.0/sdk/en/GrasshopperSDK.chm.

3.1.3 Armadillo

A prerequisite doing finite calculations is to be able to do matrix operations, such as solving the system of equations $\mathbf{K} \cdot \mathbf{a} = \mathbf{f}$. This is not a trivial matter, and since the calculations in this case need to be performed several times, speed is an important factor to consider. There are several open source linear algebra packages available for different coding languages. A C++ package called Armadillo [Sanderson (2010)] was selected for this thesis work, the main reasons being:

- **Syntax**. Armadillo provides a syntax similar to the one used in MATLAB, which is useful for migrating functions from CALFEM.
- **Expandability**. By using C++ rather than C# (which also would be a good option, since Grasshopper plugins are usually written in .NET) a possibility to expand the functionality to other platforms is added.
- **Speed**. Performance tests suggest that it is both faster than MATLAB and other C++ libraries (for instance IT++ and Newmat) [Sanderson (2010)]

3.1.3.1 BLAS and LAPACK

A library of Fortran-callable functions called BLAS (Basic Linear Algebra Subprograms) was released in 1979 [Lawson et al (1979)] as a library of low level subroutines that solves basic linear algebraic operations, such as the dot product. It has since developed into a widely used building block when developing larger linear algebra solvers. These solvers often adopts the interface of BLAS, allowing it to be exchanged for more efficient (faster) hardware-specific libraries. For instance has both Intel and AMD developed their own BLAS libraries, which are optimised for their respective processors. In Armadillo the default implementation of BLAS is called ATLAS (Automatically Tuned Linear Algebra Software), an open source implementation of the BLAS API, however it may be replaced by a machine-optimised package, thus further increasing speed.

LAPACK is a numerical linear algebraic library, written in Fortran. LAPACK offers more high-level functions than BLAS, for instance matrix factorisation and eigenvalue problem solving [Univ. of Tennessee et al (2016)]. It is commonly used, it is for instance the library behind MATLAB [Moler (2000)]. As for BLAS, LAPACK can be replaced by some other high-performance, machine-specific library.

Both BLAS and LAPACK are written so that they may be exchanged for more efficient libraries, and Armadillo also incorporates this. This means that some speed gain may be possible by compiling the program with machine-specific libraries. In

this thesis project the compiled code will be implemented with the basic libraries, but since the code is open source, any user who wishes to increase their efficiency have the possibility to recompile with optimised libraries.

3.1.4 C++/CLI

With Armadillo written in C++, and Grasshopper plugins written in .NET, some form of interpreter is needed. This task can be solved using C++/CLI. CLI stands for *Common Language Infrastructure*, and is a "[...] specification for executable code and the execution environment (the Virtual Execution System) in which it runs" [ECMA International (2012), p. 9]. Simplified, this allows high-level languages to be (somewhat) platform-independent by compiling them into code conforming to a common standard. This is done by standard for .NET languages (such as C# and VB.NET), which are (usually) compiled into a lower level language called *Common Intermediate Language* (CIL). This is not done for C++, which usually is compiled directly to machine code.

To to enable communication between the two languages one can use C++/CLI, which is an extension of C++ [ECMA International (2005), p. xii] that does conform to the CLI standard. When both languages (C# and C++) can get compiled into intermediate languages that both conform to the same standard, communication between them is possible. Of course, there are several major issues about this process that could be raised here, but they are beyond the scope of this thesis.

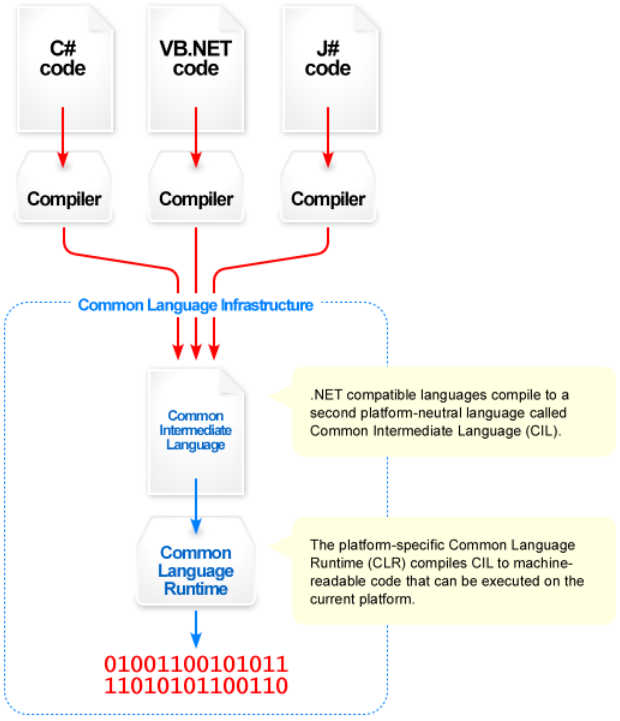


Figure 3.4: An overview of the common language infrastructure (CLI) [Image from Wikimedia Commons]

3.2 Development of a finite element solver

As mentioned in section 3.1, the finite element solver consists of three parts. A engine written in C++ to take advantage of the matrix libraries avilable in the language, a Grasshopper plug-in written i C# and a link between the two. The following sections will go through each part in more detail.

3.2.1 Finite element engine

The finite element engine, called CIFem, is the heart of the package. It is here that the elements are assembled to a structure, and the structure is calculated and analyzed. Here follows a description of the most important classes in CIFem.

3.2.1.1 Structure

The structure is the hub in which all geometrical data is stored. It keeps track of all elements and nodes in the system. When an element is added, the structure is responsible for connecting it in a proper way to neighbouring elements and nodes. This connection is done through the degrees of freedom, which is the only object shared between different elements and nodes, and thereby acts as the link between them.
The structure is also responsible for the assembling of the global stiffness matrix.

3.2.1.1.1 Degrees of freedom The degrees of freedom, or DOF:s, can be either translational or rotational. They store information about possible prescribed displacements, as boundary conditions where a fixed degree of freedom has a prescribed displacement of 0.
After a system as solved using one of the solvers, the resulting displacements or reaction forces are stored in the degrees of freedom. This information is later used when the elements are post-processed, see section 3.2.1.4 .
The degrees of freedom all have a unique index which ranges from 0 to the number of DOF:s in the system. This index is corresponding to a position in the global displacement and force vector and is used when the global stiffness matrix is assembled and when resulting forces and displacements are stored in the DOF.

3.2.1.1.2 Nodes The nodes contain information about their location in the appropriate space, either a 3D-point (XYZ) or 2D-point (XY) depending on the system. They also contain a set of degrees of freedom, three for 2D-nodes, two translational and one rotational, and six for 3D-nodes, three translational and three rotational. A node is responsible for providing its DOF:s with the appropriate boundary conditions, if any are present, in the form of preset displacements.
When the structure is assembled, the nodes help distribute the DOF:s to the elements connected to it. How this is done depends on the element releases, see section 2.1.3 and the upcoming section.

3.2.1.1.3 Elements Two element types are implemented in the engine, 3D-frame elements described in section 2.1.1.5 and springs described in section 2.1.2. The elements contain a set of degrees of freedom, that can be shared with other elements and nodes depending on coupling conditions (called releases for frame elements).
The elements are responsible for generating their own element stiffness matrix (see eq. 2.67), which are later assembled by the structure to a global stiffness matrix. The frame elements are also responsible for calculating their self weight load vector (using eq. 2.68), given the gravity field which affects them.
Some of the post-processing, including the calculation of section forces and displacements along the element, is done by the elements them self, using data stored in the degrees of freedom.

3.2.1.2 Forces

There are two types of forces implemented in the solver, gravity load and point-forces and moments. These loads are grouped in load combinations and used by the linear solver when solving the system.
The gravity can be controlled in terms of amplitude and direction, using a gravity field. This then generates forces and moments (calculated by the elements).
The point forces and moments consist of a direction, amplitude and point of application.

3.2.1.3 Solvers

There are currently two solvers implemented in the engine. The first is a linear solver that solves the displacements and reaction forces, given a set of applied forces and boundary conditions. The second is an eigenmode solver, that solves the static eigenmodes, or canonical stiffnesses for a structure (see section 2.2).

3.2.1.3.1 Linear solver The linear solver solves the equation system $\mathbf{Ka} = \mathbf{f}$. This is done by separating the system to free and prescribed degrees of freedom:

$$\begin{bmatrix} \mathbf{K}_{ff} & \mathbf{K}_{fp} \\ \mathbf{K}_{pf} & \mathbf{K}_{pp} \end{bmatrix} \begin{bmatrix} \mathbf{a}_f \\ \mathbf{a}_p \end{bmatrix} = \begin{bmatrix} \mathbf{f}_f \\ \mathbf{f}_p \end{bmatrix} \tag{3.1}$$

where f denotes free and p denotes prescribed DOF:s. The system is then solved for the free DOF:s where all information in known except for the displacements and/or rotations of the DOF:s:

$$\mathbf{K}_{ff}\mathbf{a}_f = (\mathbf{f}_f - \mathbf{K}_{fp}\mathbf{a}_p) \tag{3.2}$$

Finally the reaction forces and/or moments are calculated:

$$\mathbf{f}_p = \mathbf{K}_{pf}\mathbf{a}_f + \mathbf{K}_{pp}\mathbf{a}_p \tag{3.3}$$

The displacements, rotations, reaction forces and reaction moments are then stored in the DOF:s and later used for post-processing of the elements.

3.2.1.3.2 Eigen solver Canonical stiffnesses are described in section 2.2. The eigensolver calculates these canonical stiffnesses, or static eigenmodes, by solving the eigenvalue problem for the free DOF:s of the stiffness matrix, using the decomposition shown in eq 3.1:

$$eig(\mathbf{K}_{ff}) \tag{3.4}$$

This gives a resulting vector of the eigenvalues and a matrix of the eigenmodes:

$$\begin{aligned} \lambda &= \begin{bmatrix} \lambda_1 & \lambda_2 & \dots & \lambda_n \end{bmatrix} \\ \mathbf{V} &= \begin{bmatrix} \mathbf{v}_1 & \mathbf{v}_2 & \dots & \mathbf{v}_n \end{bmatrix} \end{aligned} \tag{3.5}$$

where n is the number of free DOF:s. To get reaction forces for a certain mode, the free DOF:s are assigned to this modes eigenvector:

$$\mathbf{a}_f = \mathbf{v}_i \tag{3.6}$$

after which eq 3.3 can be used. The displacements, rotations, reaction forces and reaction moments are then stored in the DOF:s for the modes that are to be post-processed (see below).

3.2.1.4 Post-processing

The post-processing can be divided into two steps. The calculation of section forces and displacements, and utilisation checks of the elements.

3.2.1.4.1 Section forces and displacements The section forces are calculated by the elements, using the information stored in their degrees of freedom. For the 3D-frame elements a chosen number of evaluation points along the local x-axis are set up in which the following displacements and section forces are calculated:

- u - Displacement along the local x-axis
- v - Displacement along the local y-axis
- w - Displacement along the local z-axis
- ϕ - Rotation around the local x-axis
- N - Normal force
- V_z - Shear force in local z-direction
- V_y - Shear force in local y-direction
- M_y - Bending moment around the local y-axis (major axis bending)
- M_z - Bending moment around the local z-axis (minor axis bending)
- T - Torsional moment around the local x-axis

The forces are calculated load-combination by load-combination for the linear solver and mode by mode for the eigen solver, and stored in the elements.

3.2.1.4.2 Utilisation checks For the frame elements, a set of utilisation checks are implemented. Given the scope of the thesis, these checks are chosen to be simplified, to give an overall assessment of the performance of the structure rather than actually verifying it. In these simplified checks a generic isotropic material is considered and it is assumed that it has an ultimate capacity whereafter it fails. In other words, for all checks the utilisation (η) should be less than 1: $\eta \leq 1$

Axial check

$$\eta = \frac{\sigma_{||}}{f_u} \tag{3.7}$$

where:

$$\sigma_{||} = \frac{N}{A} \tag{3.8}$$

Combined axial and bending check

$$\eta = \frac{\sigma_{||}}{f_u} \tag{3.9}$$

Where

$$\sigma_{||} = \frac{N}{A} + \frac{M_y}{I_y} z_{max} + \frac{M_z}{I_z} y_{max} \tag{3.10}$$

For a rectangular cross section $z_{max} = \pm h/2$ and $y_{max} = \pm b/2$, why four checks are needed, where $\sigma_{||}$ is taken as the maximum absolute value of the four.

For circular sections this check can be simplified, since $I_y \equiv I_z$ and $z_{max} = y_{max} = \pm r$. Therefore the check can be simplified to:

$$\sigma_{||} = \frac{N}{A} + \frac{M_{principle}}{I} r \tag{3.11}$$

where:

$$M_{principle} = \sqrt{M_y^2 + M_z^2} \tag{3.12}$$

Here only two checks are needed, and $\sigma_{||}$ is taken as the maximum absolute value of the two.

Shear check

Shear checks are only implemented for rectangular sections where the check is performed in the local y and z directions of the element, separately.

$$\eta = \frac{\tau}{f_u/\sqrt{3}} \tag{3.13}$$

where:

$$\tau = \frac{VS}{It} \tag{3.14}$$

All checks are done on the evaluation points used to calculate the section forces. The check that gives the highest utilisation for each point is then stored in the elements.

3.2.1.5 Optimisers

Three section optimisers are implemented in the engine.

3.2.1.5.1 Section sizer and rotator The section sizer and rotator uses the linear solver described in section 3.2.1.3.1 in an iterative procedure. This procedure is further described in section 3.3.1 and section 3.3.2.

3.2.1.5.2 Mode shape optimiser The mode shape optimser uses the eigensolver described in section 3.2.1.3.2. It is further described in section 3.3.3

3.2.1.5.3 Combined optimiser The combined optimiser works by a combination of the section sizer and rotator, and the mode shape optimiser and is further described in section 3.3.4

3.2.2 Wrapper

The wrapper is by far the smallest project in the solution. Its sole purpose is to link the FE-engine written is C++ to the Grasshopper plug-in written in C#. This link is done using C++/CLI which is a language able to handle both managed code (C#) and unmanaged code (C++). The base principle is that for every class in the FE-engine that needs to be exposed to the Grasshopper plug-in, there is a wrapper class holding a pointer to an unmanaged object. The Grasshopper plug-in can then make function calls to the wrapper objects, which in turn send these calls forward to the FE-engine.

3.2.3 Grasshopper plug-in

This section describes the grasshopper plugin project, called *CIGull*. An image of the grasshopper set-up of a structure will be presented and each part will be described.

3.2.3.1 Overview

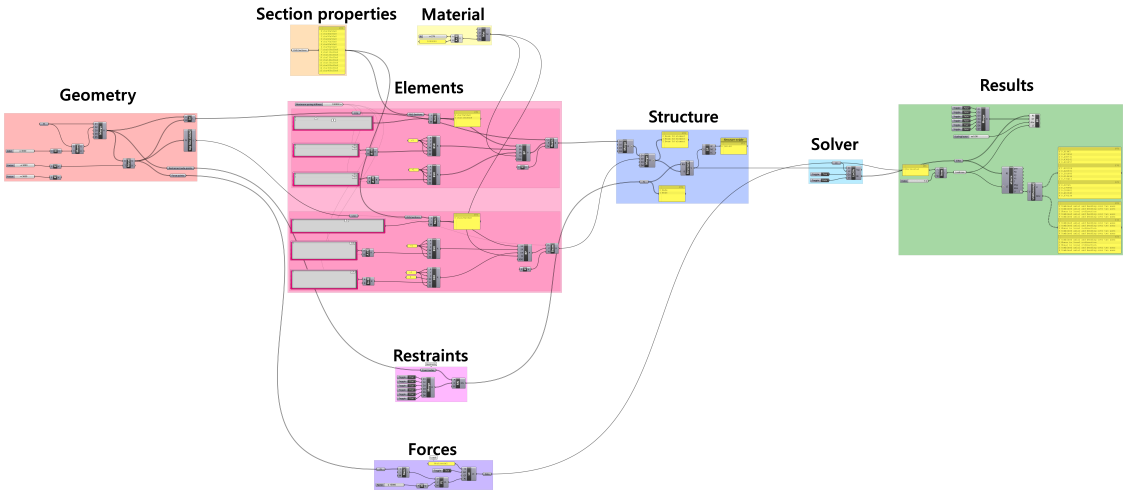


Figure 3.5: An overview of the set-up of a simple structure in Grasshopper

The set-up of the Grasshopper file can be divided into a number of subgroups:

- **Geometry generation**
- **Section properties**
- **Material**
- **Elements**
- **Restraints**
- **Forces**
  - Point loads and moments
  - Gravity loads
- **Structure**
- **Solver**
  - Linear solver
  - Eigensolver
- **Optimisers**
  - Iterative section sizer
  - Mode shape optimiser
  - Combined optimiser
- **Results**

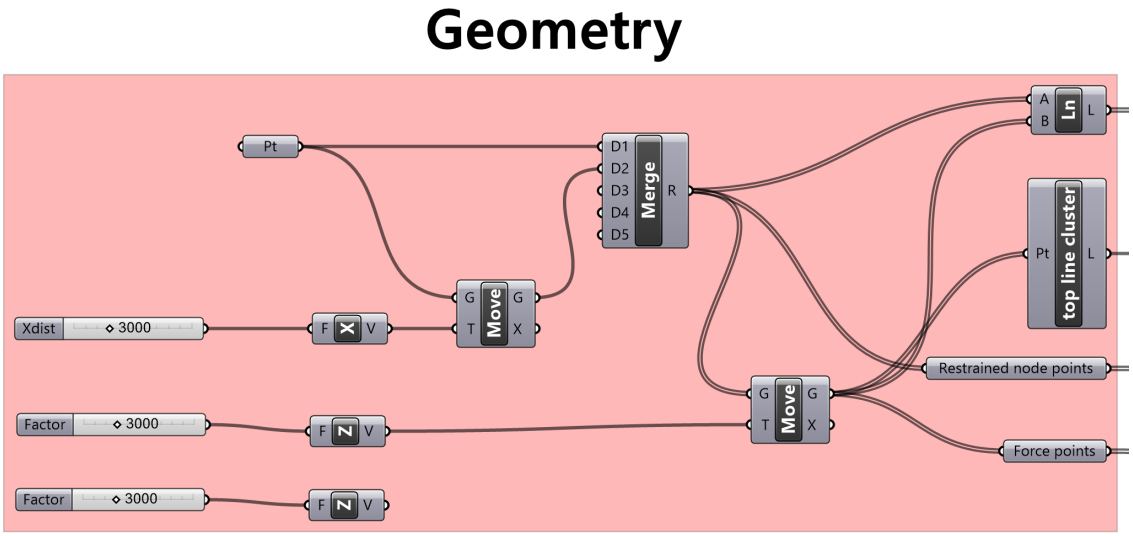


Figure 3.6: The geometry generation consist of only two types of elements, points and lines. In this section only already built-in Grasshopper components and types are used.

3.2.3.2 Geometry generation

In Rhinoceros and Grasshopper there are a multitude of geometric elements available, including curves, breps, surfaces and meshes. The only types that are needed for generation of geometry for *CIGull* are the two simplest ones, points (used for restraints and applied loads) and lines (center lines for the elements).

3.2.3.3 Elements

The actual elements, which at the time of writing are only implemented in the form of beams, are created from a:

- Center line (see above)
- Beam property, which contains
  - Cross section
  - Start and end releases
  - Material
  - Cross section group (optional, used for optimisation processes)
- Element normal vector (local z axis)

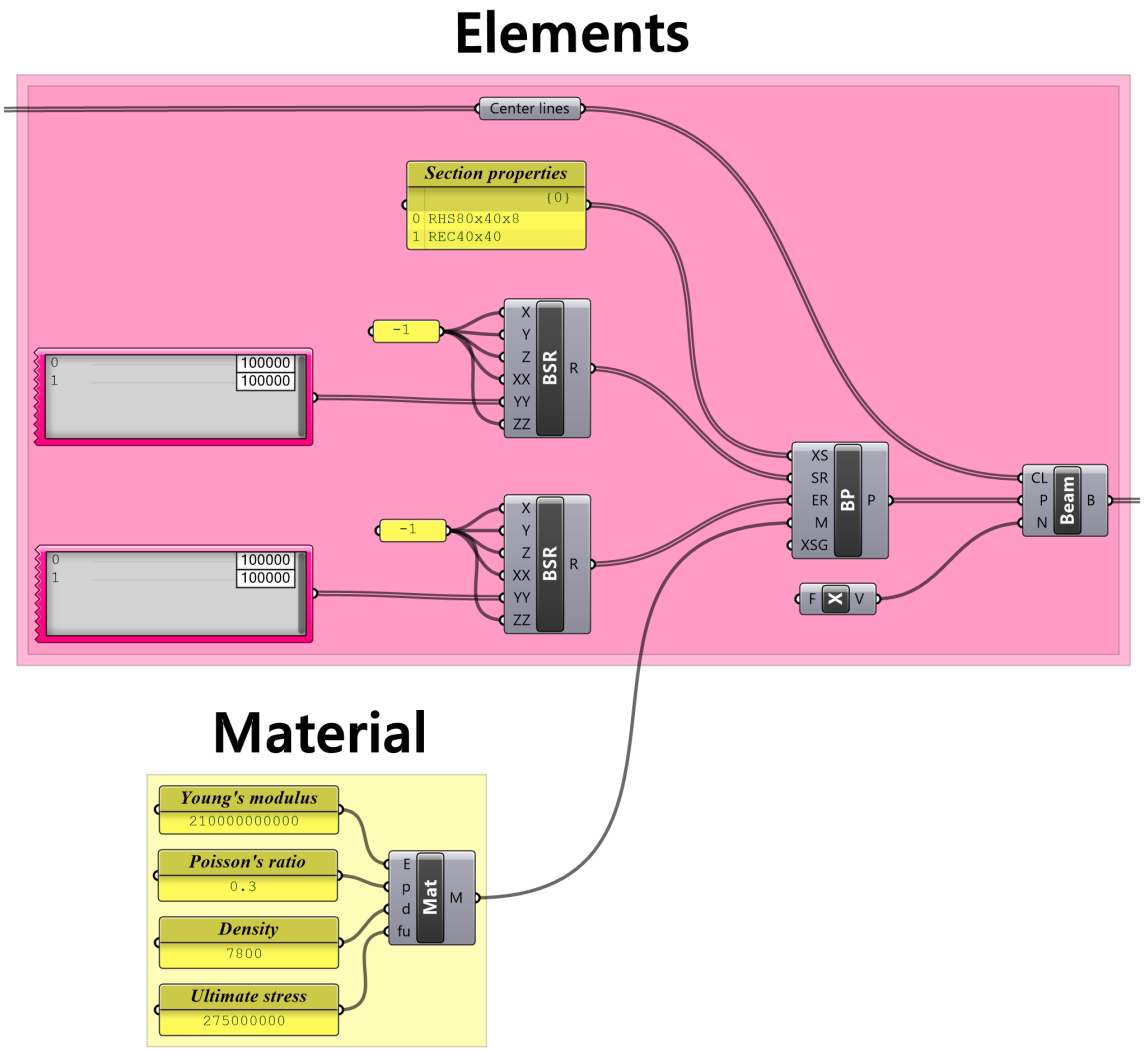


Figure 3.7: Element generation, including material generation and the generation of beam properties. Here two beams are generated (from two separate centerlines and two sets of beam properties. The elements have fixed releases for all translations, and rotation in the XX- and ZZ directions. The YY-rotations are restrained by a rotational spring of $k = 100$[kNm/rad] for both elements in both the start and end release.)

3.2.3.4 Material

The material is implemented as linear isotropic and generated from four parameters:

- Young's modulus E
- Poisson's ratio ν
- Density ρ
- Ultimate stress f_u.

The ultimate stress is used in utilisation checks (see section 3.2.1.4.2). In the example in fig. 3.12 a steel material is used, where the ultimate stress is taken as the yield stress as a simplification.

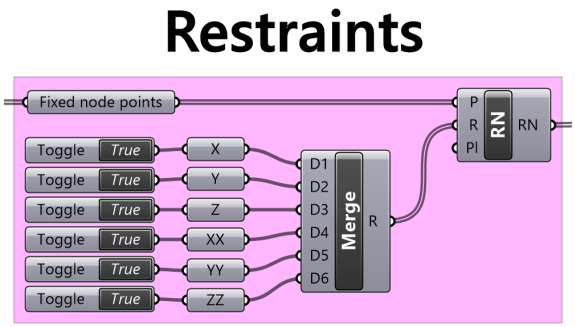


Figure 3.8: Restraints

3.2.3.5 Restraints

The restraint nodes are generated from three parameters:

- The point to restrain
- The directions to restrain
- The plane in which the restraint directions work (optional)

In the current version of the code, the restraint is implemented only for global coordinates (the third parameter defaults to the XY plane). See 'future implementations' 7.9.1.

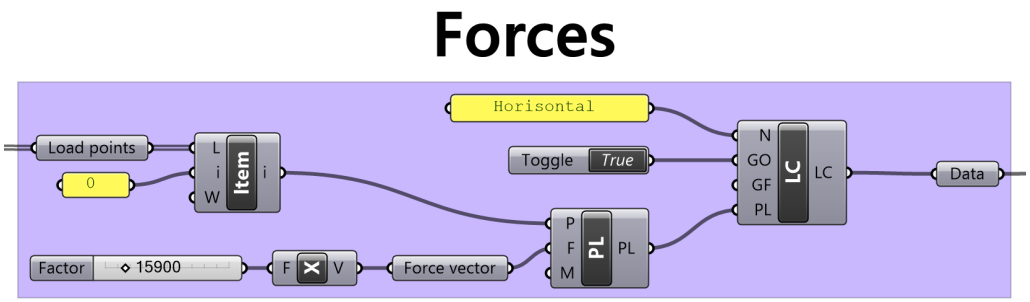


Figure 3.9

3.2.3.6 Forces

Forces and moments are applied to the structure in load combination. A load combination consist of a name, a list of point loads and moments, and gravity options. The point loads are created from a position and a force or moment vector. Distributed loads are currently not implemented, see 'future implementations' 7.9.1.

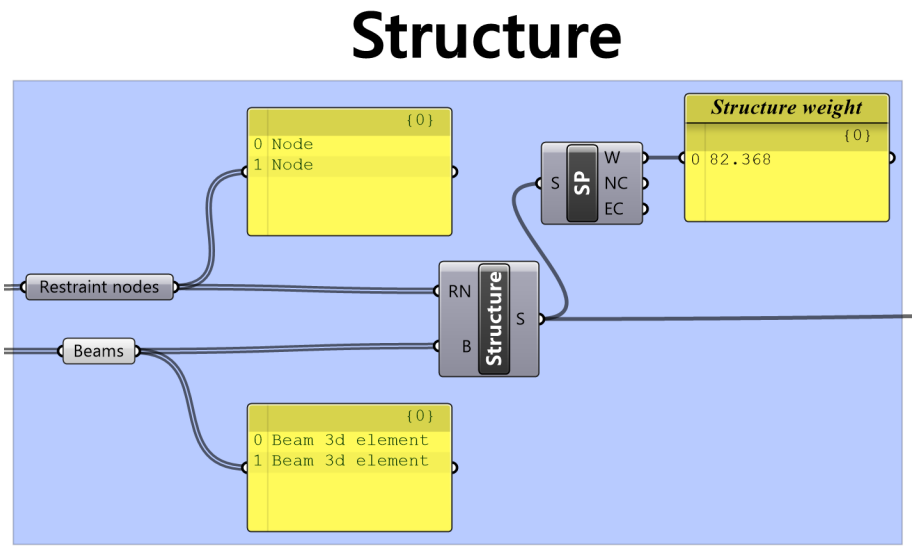


Figure 3.10: A structure component and a structure properties component

3.2.3.7 Structure

The structure is mainly a container for restraint nodes and beams. However, as described in 3.2.1.1, the structure is also responsible for connecting the degrees of freedom in the nodes and elements.

3.2.3.8 Solver

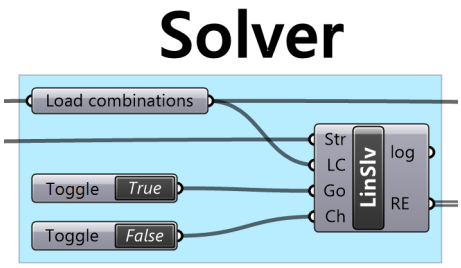


Figure 3.11: There are two solvers currently implemented, shown here is the linear solver.

CIGull currently has two available solvers, a linear solver and an eigensolver. The linear solver (see fig. 3.11) combines a structure and a set of load combinations and solves the displacements and reactions. It also has a check of the structure implemented, which if turned on will control if the structure is sufficiently restrained. It is however recommended to check this once and then turn it off, to improve speed. The eigensolver has three parameters:

- Structure
- List of modes to post-process
- Boolean 'go' toggle

Where the second parameter is an input to control how many eigenmodes are included in the results.

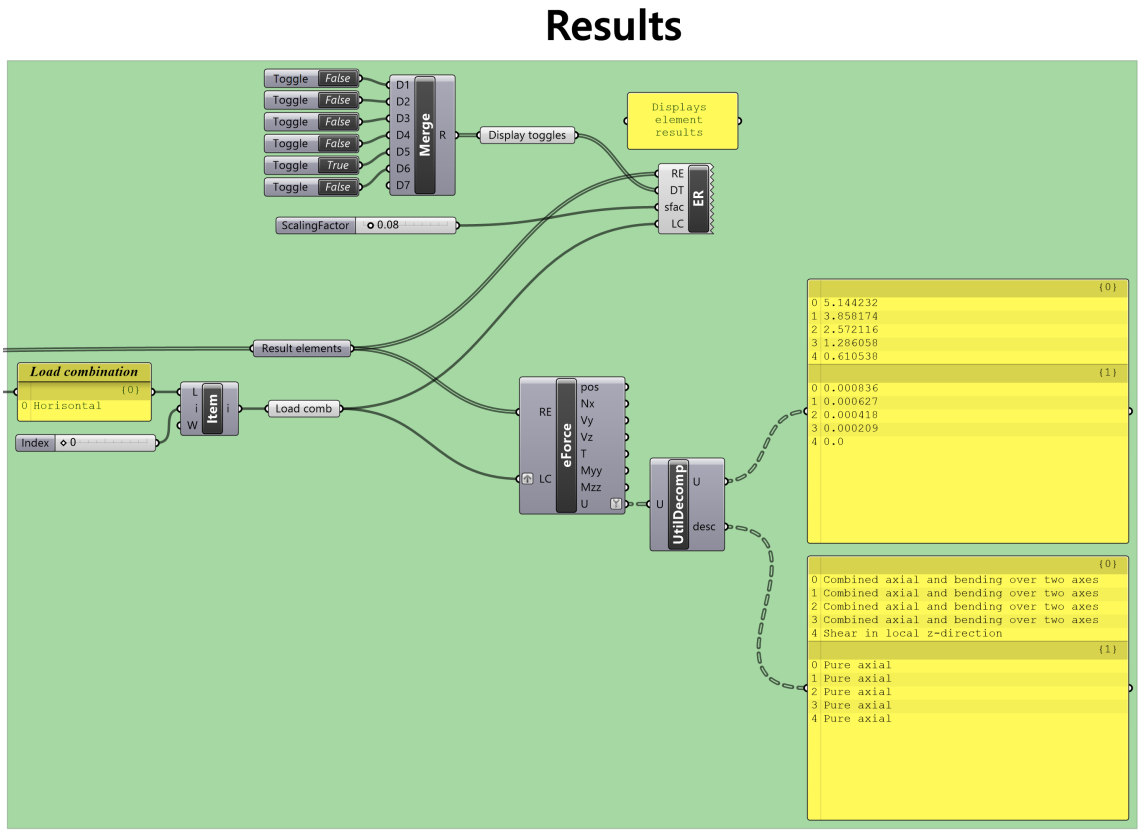


Figure 3.12: Outputs from a solver can be outputted to Grasshopper (in the form of text and numbers), or plotted to the Rhino window.

3.2.3.9 Results

Once the solver has run, all results (incl. utilisations) are calculated, so what is left is basically to access them. The results can either be interpreted numerically in Grasshopper, or visualised in the Rhino viewport. Forces and moments can be plotted as diagrams, and element displacements can be drawn.

It can be noted that the element results can be interpreted not only from a linear solver, but from an eigensolver or from an optimising component as well.

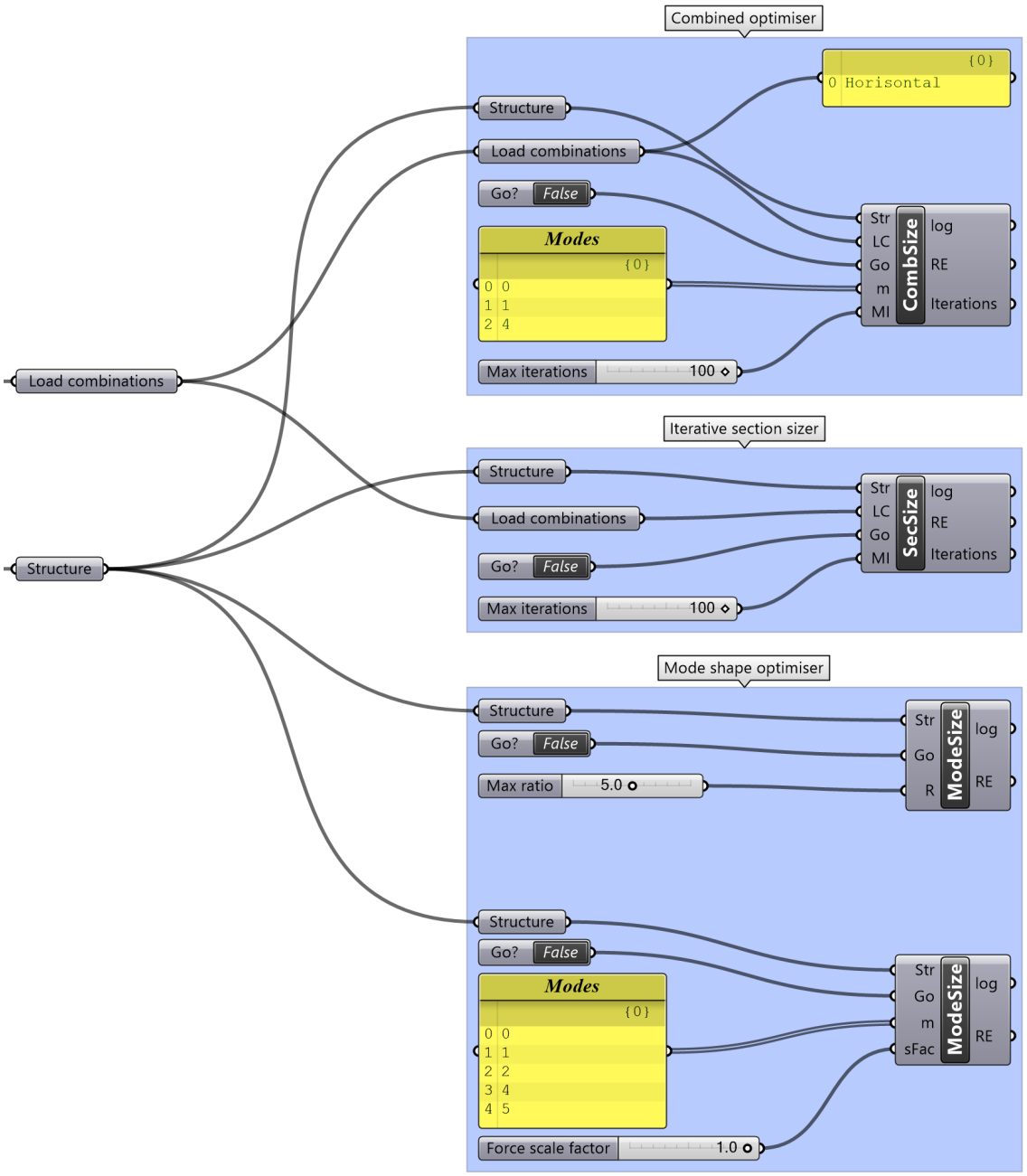


Figure 3.13: The four different optimisers implemented.

3.2.3.10 Optimisers

There are four optimising components implemented, however there are really only three optimising methods. This is because the mode shape optimisers is implemented in two forms (see fig. 3.13, the lowest component group). The difference is how the eigenmodes to include are chosen. In the upper version all eigenvectors with an eigenvalue that has a ratio which is lower than the value given as input are included, and in the lower version the specific modes can be chosen.

3.2.4 Verification of results

To verify that the finite element solver is performing its calculations correctly, a comparative case study has been made. In the study results from *CIGull* were compared to results calculated using the CALFEM package for MATLAB. The study is presented in Appendix A.

The results are expected to be similar, since the FE calculations in *CIGull* are based on the CALFEM formulations. In order to only address the core of the finite element calculations, calculations of variables such as the area or the second moment of area are omitted by using the *CIGull* values in CALFEM.

The results were compared for element displacements, and found to be corresponding within a margin of error of less than 0.1‰for the studied case.

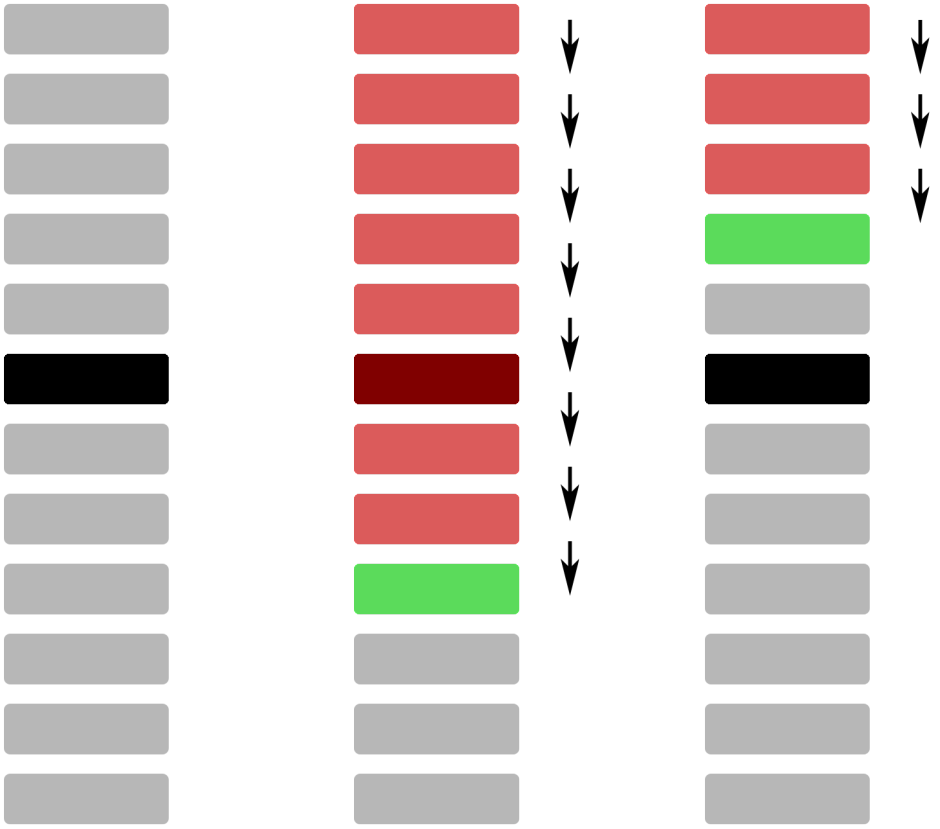
3.3 Optimisation methods

3.3.1 Iterative section sizer

As stated in section 3.2.1.5 three element cross section optimisers are implemented in the FE-engine. The first is the section sizer that utilises the results from the linear solver (see section 3.2.1.3.1).

3.3.1.1 General

The iterative section sizer works by iteratively solving the system in its current state and updating the element cross sections each iteration.The sections of the elements are changed based on their respective utilisation, see 3.2.1.4.2. The section property will be changed if the maximum utilisation for the element exceeds some set limits. If the maximum stress is above the set maximum limit (overutilised) the the section property will increase in size, if the stress is below the set minimum limit (underutilised) it will decrease in size. The iteration process will stop once a solution is found where every element in the system has a utilisation within the set limit, or when no new solution is found compared to the previous iteration.
The main goal for the section sizer is to reduce the weight of the elements. Therefore the list of sections used is stored based on cross section area which, for elements with constant cross section and material, will correspond to weight and thereby material usage. This makes choosing an earlier position in the list favourable for reducing the weight of the element. See Appendix C for an example of such a list.
The force and stress pattern in a structurally indeterminate structure are highly dependent on the stiffness distribution. This means that a change of sections likely will influence the way the structure behaves and how the forces are carried. In other words, how the sections are updated may influence the end result significantly. To get a better control of this, four different approaches on how to update the sections are implemented.

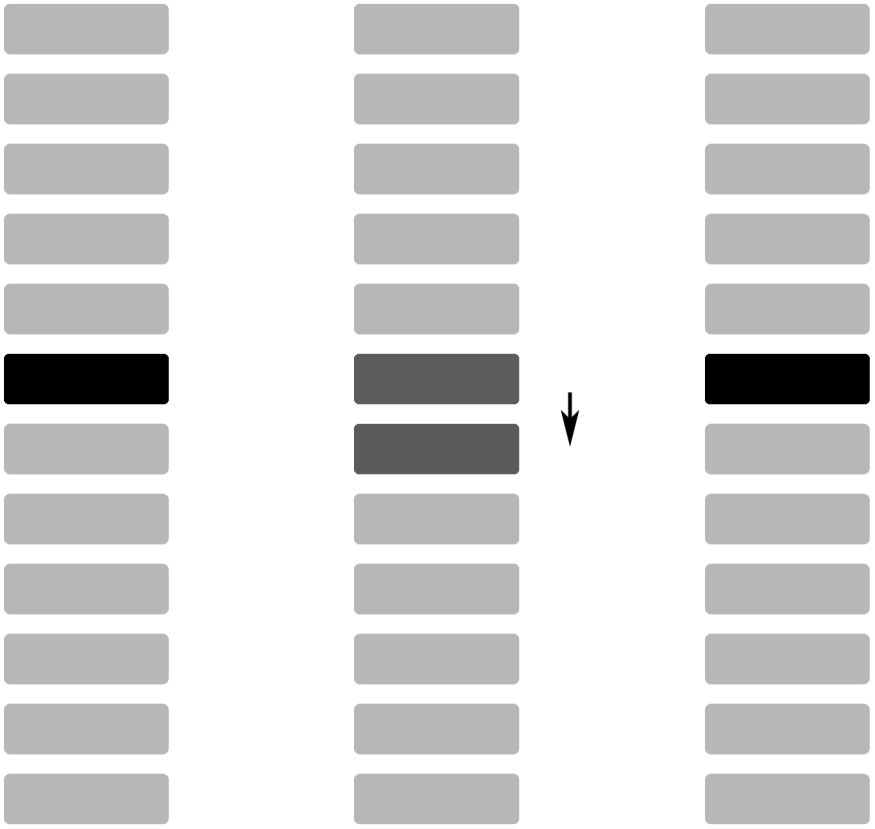


(a) Current section **(b)** Overutilised **(c)** Underutilised

Figure 3.14: Section sizer method 1. Set of possible sections sorted by area. The black denotes the section from previous iteration. If the member is overutilised, all sections are checked, from smallest to largest, until one is found that can withstand the forces. The same method is used if the member is underutilised. Red sections are checked and failed. Green is checked and ok.

3.3.1.2 Method 1: Smallest acceptable section

The first method searches the set of possible sections from smallest to largest area until a cross section is found that can withstand the forces from the current iteration. This makes it possible for each element to get a quite drastic change in cross section each iteration, which can lead to convergence in fewer iterations. This faster convergence comes with a cost of having to do multiple checks for each element each iteration, which makes each iteration slower. The possibility to make big jumps in change of cross section also means that the force pattern of the structure can change quite a lot for each iteration.

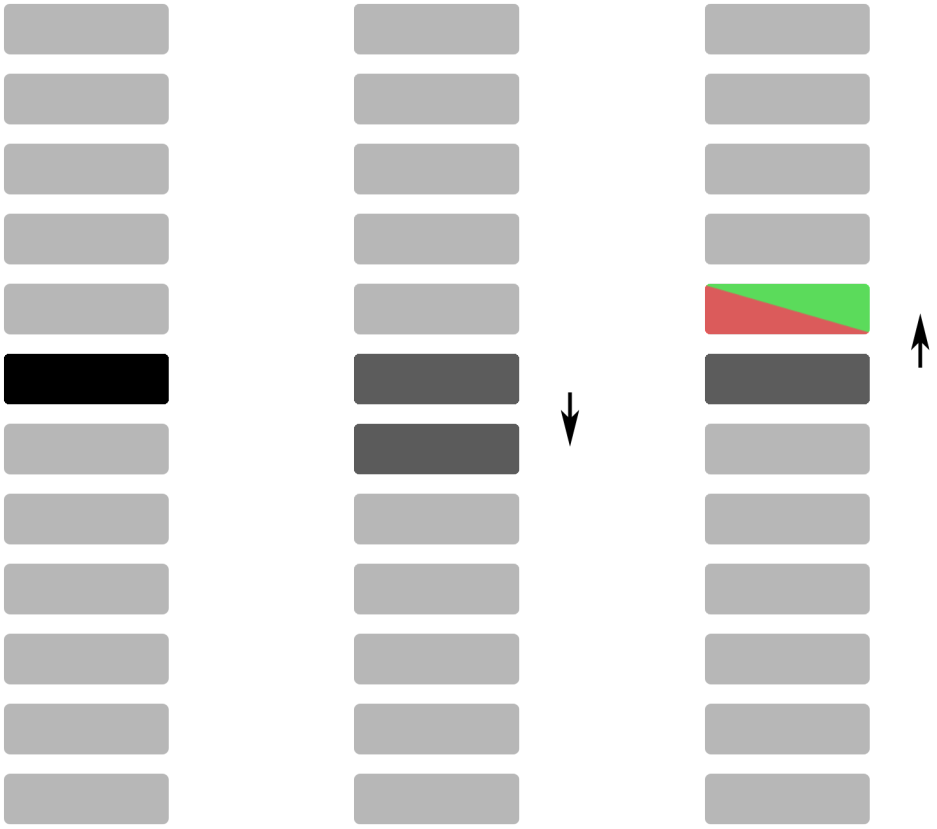


(a) Current section **(b)** Overutilised **(c)** Underutilised

Figure 3.15: Section sizer method 2. Set of possible sections sorted by area. The black denotes the section from previous iteration. If the member is overutilised, the next section in the list is picked. If the member is underutilised nothing is done. No checks are done for this method.

3.3.1.3 Method 2: Step-wise incrementation

The second method only allows an increase in size of the element. This is done by changing the cross section to the next one on the list, i.e. one with larger cross section area, for elements that have a maximum utilisation that exceeds the set limit. This method will be quick for each iteration (since there is only one initial check being done), but it may need more iterations to find convergence as the cross sections are incremented one size at a time. This method should only give small changes in the force pattern between each iteration, as only small changes to the cross sections are allowed. The method might, though, produce elements with quite low utilisation as the stresses in some elements may decrease as the stiffness of others increase.

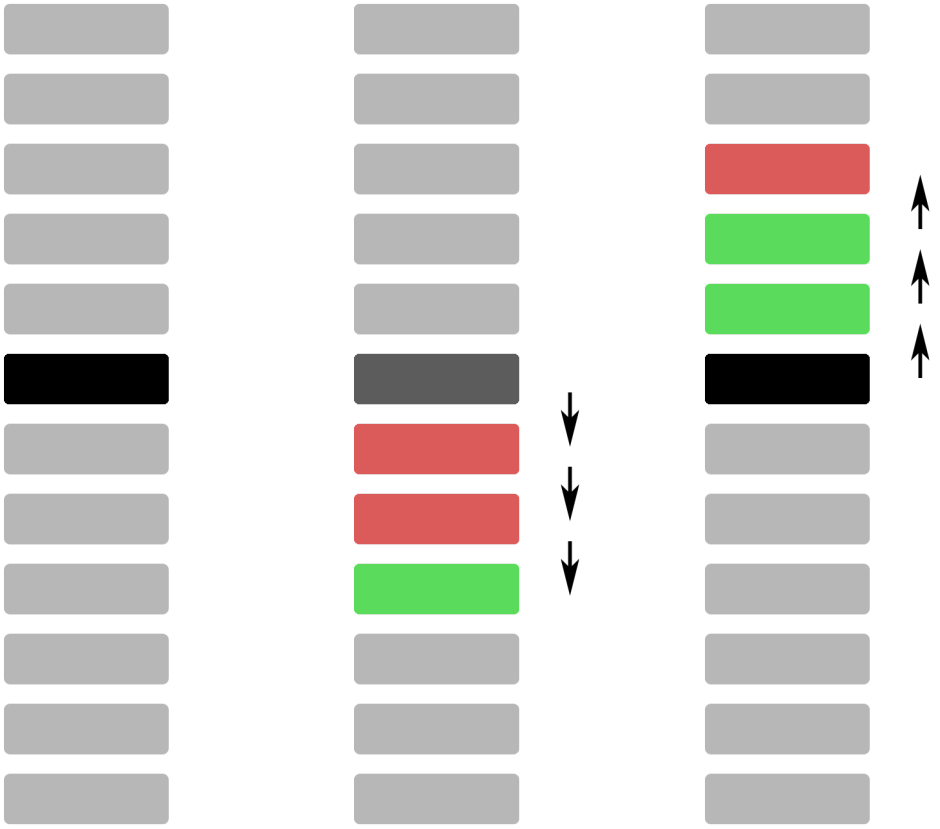


(a) Current section **(b)** Overutilised **(c)** Underutilised

Figure 3.16: Section sizer method 3. Set of possible sections sorted by area. The black denotes the section from previous iteration. If the member is overutilised, the next section in the list is picked. If the element is underutilised, it is checked if the cross section above in the list is able to handle the load. If this is, that section is picked, otherwise the cross section remains unchanged.

3.3.1.4 Method 3: Step-wise incrementation with down-sizing

The third method checks the sections in close proximity of the cross section currently applied, in terms of size of area. If the element is overutilised the cross section will be incremented to the next on the list, i.e. one with greater area. If it is underutilised it will check if it's possible to decrease the cross section size to the previous on the list, if not it will keep the current one. Problems that can arise with this method is that the cross section on the next position on the list does not necessarily handle the section force better than the previous, as larger cross section area do not implicitly give higher resistance to specific loads.
As this method only allow small changes, it should give quite small changes in force pattern between each iteration. It also usually require quite a few global iterations before convergence is found.



(a) Current section **(b)** Overutilised **(c)** Underutilised

Figure 3.17: Section sizer method 4. Set of possible sections sorted by area. The black denotes the section from previous iteration. If the member is overutilised, all sections larger than the current are checked in order from smallest to largest, until one is found that can withstand the forces. If the member is underutilised, all sections smaller than the current are checked, in order from the largest to the smallest, until one is found that can **not** withstand the forces. The section one step larger is then taken. Red sections are checked and failed. Green is checked and ok

3.3.1.5 Method 4: First acceptable solution from current section

The fourth method works similarly to the first method described in 3.3.1.2. The difference is that the search for an appropriate cross section for an element starts on the position in the list that was used in the previous iteration and then checks downwards on the list of available cross sections if the element is overutilised or upwards on the list if the element is underutilised. This is intended to make the update procedure quicker compared to method 1 (3.3.1.2), as the search starts at an position that in most cases should be closer to the sought one, if the force pattern has not changed too significantly.
The possibility to do big changes in sections size could lead to big changes in force patterns. As more iterations are run though, the changes magnitude of the changes should decrease, why this method could have the possibility for faster convergence compared to method 1.

3.3.1.6 Sorting of cross sections

The set of possible cross sections used in the methods is, as mentioned before, sorted by area. Sorting in this manner might, though, give situations where cross sections further down on the list are worse for carrying a specific load compared to earlier entries. As an example one can look at two simple rectangular cross sections:

$$\begin{aligned} h_1 &= 150\ mm \\ w_1 &= 50\ mm \\ A_1 &= 7\ 500\ mm^2 \\ I_{\bar{y}1} &= 14\ 062\ 500\ mm^4 \\ I_{\bar{z}1} &= 1\ 562\ 500\ mm^4 \end{aligned} \tag{3.15}$$

$$\begin{aligned} h_2 &= 100\ mm \\ w_2 &= 100\ mm \\ A_2 &= 10\ 000\ mm^2 \\ I_{\bar{y}1} &= 8\ 333\ 333\ mm^4 \\ I_{\bar{z}1} &= 8\ 333\ 333\ mm^4 \end{aligned} \tag{3.16}$$

Comparing the two cross sections one can note that the area in the cross section in eq 3.16 is larger than the cross section in eq 3.15; This means that the first cross section will be placed higher in the list. If the worst load situation for an element should lead to major axis bending being the critical utilisation factor, though, the first cross section would be better at handling this, even though it has a later position in the set.
This is something that can be problematic, especially for method 3 and 4 during downsizing. It can be exemplified by a situation where an element is flagged as underutilised, with a current cross section slightly larger in terms of area than the one described in eq 3.16. Both method 3 and 4 then checks the cross sections upwards in the list, i.e. eq 3.16, to see if it is able to handle the load. If the worst load situation is major axis bending, too high for eq 3.16 it means that the cross section will remain unchanged, even if the cross section described in eq 3.15 would be able to handle this load. This happens because the downsizing procedure for both method 3 and 4 is done by checking the cross sections from largest to smallest.

3.3.2 Section Rotator

As described in section 2.4.2, the orientation of the cross section for an element can be of high importance. As described, it is possible to find a "principle moment" and its direction by equation 2.93 and 2.94. A rotation of the cross section around its own axis can lead to a stiffness redistribution and thereby a force redistribution. This means that a re-computation of the system needs to be done once a section has been rotated, which might lead to a new rotation of the section. This leads to a iterative procedure, where one first solves the system then rotates the sections each iteration until convergence, i.e. when no more rotations of the sections are needed.

As this is a iterative procedure, as the section sizer described in 3.3.1, the two can be used in tandem. This is done by solving the system, rotating, solving system again and then updating sections. This is done iteratively until convergence, i.e. when no sections needs to be updated or rotated.

3.3.3 Mode shape optimiser

When the mode shape has been used previously (see the theory section, 2.4.3), the process has been manual. The eigenmodes have been obtained and sorted by eigenvalue, thereafter the designer has been given the choice to identify relevant deformations and take actions to reduce the structures susceptibility to them. This thesis work aims to implement this process, and if possible also improve it by implement it in an automated process.
The mode shape optimiser is the second element cross section optimiser implemented in the FE-engine, utilising the results from the eigensolver described in section 3.2.1.3.2

3.3.3.1 Procedure

The optimisation method is based on the same foundation as the manual method previously described. The intention is however to automate the process by letting the eigenmodes influence the stiffness redistribution. The hypothesis is that this will allow the structure to be more efficiently utilised, since the stiffness increase gets focused on the most heavily strained elements. Since the eigenvectors obtained are normalised, they do not need to correspond to any real load case. Therefore the these eigenvectors must be properly scaled by a chosen scale factor s so that they can be used to generate useful results. Described schematically this is done by:

1. Solve the eigenvalue problem
2. Select the relevant eigenmodes.
   This can be done either by choosing the specific modes or choosing an eigenvalue ratio limit (for instance one could include all eigenmodes corresponding to eigenvalues up to x times the lowest eigenvalue).
3. For each eigenmode:
   (a) Calculate a ratio of the eigenvalue $R = \lambda_i/\lambda_1$, where λ_1 is the lowest eigenvalue.
   (b) Using the displacement vector to find the corresponding stresses in the elements (in the same way as for a structure with prescribed displacements in all of its degrees of freedom)
   (c) Analysing the elements to find their utilisation rate
   (d) Find the element in the structure with the lowest utilisation rate η_{min}
   (e) Calculate the applied forces on the DOFs (according to eq. 2.83)
   (f) Scale all forces with a scale factor $C = \eta_{min}^{-1} \cdot R \cdot s$.
   This scales the applied forces so that the least utilised element gets a utilisation of 100%, and all other elements are overutilised, for the first eigenmode. All other (selected) eigenmodes gets a reduced scale factor

(R) to correspond to the strain they induce compared to the first eigenmode.
   (g) Increase the section properties so that the elements can carry the load. This is done in the same manner as method 1 used by the section sizer, see section 3.3.1.2
   (h) Record the new section property
4. When all eigenmodes have been analysed, the section property of each element is taken as the largest one obtained for any eigenmode.

One thing that can be noted is that this process only considers modal loads, and no actual load cases. It is therefore useful for finding "stable" structures but not necessarily useful for an actual load case.

3.3.4 Combined section sizer

Since the aim of the mode shape optimiser, described in section 3.3.3, is to find inherently stable structures, an hypothesis is that these structures seem like a good starting points for the section sizers. The section sizers are highly prone to be influenced by the initial design, and by starting from an effective structure the behaviour is steered toward a potentially more efficient behaviour.
The combined section sizer is the third element cross section optimiser implemented in the FE-engine and is easily explained, since it is a combination the other two methods:

1. Perform the mode shape optimisation (see section 3.3.3)
2. Starting from the acquired design, perform a section sizing procedure (see section 3.3.1)

3.3.5 Section optimisation using genetic algorithms

In Grasshopper, there is a single-objective genetic algorithm implementation included. It's called *Galapagos* and contains a genetic algorithm solver and a simulated annealing solver. It is well implemented in the Grasshopper interface, making it easy to assign variables and output.

Being a single-objective solver one must create a combined value which includes all desired parameters. In this case for instance it would not be enough to simply minimise the weight, because the structure also needs to have a utilisation less than 100% in all sections to be acceptable. One way to do this is to use a *hard constraint*, which basically only accepts solutions with acceptable utilisations, for instance by:

$$m = \begin{cases} m & \text{if} \quad \eta_i < 1 \quad \forall \quad i \\ \infty & \text{else} \end{cases} \tag{3.17}$$

This method treats all unacceptable solutions the same, regardless of whether the maximum utilisation is 101% or 1000%. A scaled weight function could be used

instead:

$$m = \begin{cases} m & \text{if} \quad \eta_i < 1 \quad \forall \quad i \\ m \cdot \eta_{max} & \text{else} \end{cases} \tag{3.18}$$

As an alternative to Galapagos, there is a plugin called Octopus available [Vierling, R (2014)]. It has two features that are useful for this optimisation task:

- Start from presets option
- Multi-objective optimisation

Starting from presets is an interesting option because genetic optimisers often start from very bad fitness and work themselves toward a good solution. Since each solution is quite time consuming to evaluate even a relatively small improvement to the initial conditions could prove to result in a large amount of saved time. This could be done either by manually setting the starting geometry to something which is reasonable, or by using another generative algorithm.

Multi-objective optimisation is good because of the reasons stated earlier. Now the genetic algorithm could be solved by for instance optimising for both weight and maximum utilisation. By optimising for both of these (or more) goals a number of results can be obtained, rather than just one. The advantage of this is that the cut-off between the two optimisation criteria is performed manually, putting the user in control without having to predict how a weighing factor will affect a single optimisation criterion.

3.3.6 Connection stiffness optimisation

The connection stiffness optimisation is implemented using the theory in section 2.1.3. To evaluate this method a structure is created with joints that are rigid for translations and torsion, and have rotational springs for bending in the major and minor axes. An existing genetic algorithm is then used to find the best suitable stiffness for these springs.

3.4 Strategy for experiments

In order to analyse the proposed tools, a strategy must be developed. This strategy is presented below:

1. **Which questions need to be answered?**
   - When are the different methods good/bad?
   - What are the limits of the methods?
   - How and why do the results differ?
   - What customisation is possible?
   - How practically applicable are the methods?
   - How can the different methods be combined?
2. **Which models should be analysed to answer these questions?**
   - 2d frame
     - *This frame should be simple enough to study the basics of the different methods. See fig. 5.1*
   - 3d frame
     - *This frame should be more complicated, which should show the capabilities of the different methods, in terms of both efficiency and solving capability. See fig. 5.3*
   - Dome-like structure
     - *Using a structure with a high degree of statical indeterminacy should further elucidate the strengths and weaknesses of the methods. See fig. 5.5*
   - Case study: KAFD Metro Station
     - *The complexity of this free-form structure should serve as a thorough examination that challenges the methods on a real project. See fig. 5.9*
     - *This model is intended to especially study how the methods can be combined, and how applicable the results are in a real project.*

4

Implementation

This chapter explains the implementation of the developed package in more detail. The most important parts will be explained in pseudo-code algorithms. If further information is required, the entire code will be made available (see section 7.8).

4.1 Linear solver

A very important part of the FEM package is the linear solver, which solves displacements and reactions for a set of applied loads. It can be used separately, but is also implemented in the section sizers and the combined section sizer. The process is described in algorithm 4.1:

Algorithm 4.1: Linear solver

Data: Elements, Boundary conditions, Loads
Result: Resulting displacements, reaction forces and section forces
Get element degrees of freedom;
Assign a unique index to each DOF, ranging from 0 to the number of DOF:s;
Assemble global stiffness matrix;
foreach Loadcombination **do**
    Reset forces in DOF:s;
    Calculate and apply gravity loads to DOF:s;
    Apply nodal forces to DOF:s;
    Solve system (see section 3.2.1.3.1);
    Set results to DOF:s;
    Calculate element section forces;

4.2 Optimisation methods

This section will describe in more detail how the different optimisation methods in the finite element solver CIFem has been implemented.

4.2.1 Iterative section sizer and rotator

The section sizer and rotator works by an iterative outer while-loop that runs until the system is unchanged from previous iteration. Algorithm 4.2 describes the main loop where the whole system is ran until convergence is found. It can be noted that the system can be solved twice for each iteration if sections are both rotated and changed.

Algorithm 4.2: Section sizer loop

```
Data: Initial structure
Result: Updated structure with updated sections
set updated to true;
while updated do
    if rotate sections then
        solve system;
        check and rotate sections;
        set updated to true if sections have been rotated
    if change sections then
        if updated then
            solve system;
        check and change sections;
        set updated to true if sections have been changed
```

The section updating described in algorithm 4.3 first checks if any of the elements are over utilised. If no over utilised elements are found in the whole structure it does not try to update any element, but exits the function. If one or more elements

are over utilised, all elements are checked to see if a better fit can be found.

Algorithm 4.3: Check and update element cross sections

```
Data: Current Structure
Result: Updated structure with updated sections, boolean stating if any updates
        has been done
set over utilised to false;
foreach element in structure do
    calculate maximum utilisation of element;
    if element is over utilised then
        set over utilised to true;
if over utilised = false then
    set updated boolean to false;
    return updated boolean and exit function;
set updated boolean to false;
foreach element in structure do
    try to update element section;
    if element has been updated then
        set updated boolean to true;
return updated boolean and updated structure and exit function;
```

The check if the element cross section can be updated or not is done using algorithm 4.4. Here it is checked if the element has a utilisation within a set limit. If the utilisation is outside the set limit, a new cross section is searched for using algorithm 4.5, 4.6, 4.7 or 4.8;

Algorithm 4.4: Update element cross section

```
Data: results from solution
Result: boolean stating if any updates has been done
check if element utilisation is below set minimum;
check if element utilisation is above set maximum;
if over utilised or underutilised then
    try to find new cross section;
    if update is found then
        update element cross section;
        set updated boolean to true;
        return updated boolean and exit function;
else
    set updated boolean to false;
    return updated boolean and exit function;
```

4.2.1.1 Method 1: First acceptable section

As described in section 3.3.1.2, this method works by checking all possible cross section from smallest to largest in terms of area. As mentioned before, the set of cross sections used to search through is sorted by area, which makes it possible to check the set first to last until one is found that can sustain the applied loads.

Algorithm 4.5: Update cross section by checking smallest to largest

Data: results from solution, current cross section, set of possible cross section
Result: updated section, boolean set to if the section has been updated or not
foreach cross section in possible cross sections **do**
    check utilisation of cross section with current results;
    if utilisation < maximum allowed utilisation **then**
        set updated cross section to loop cross section;
        set updated boolean to true;
        return updated cross section and boolean and exit loop and function;

4.2.1.2 Method 2: Step-wise incrementation

Method described in section 3.3.1.3. If the member is over utilised the current position in the set is found and the cross section is then updated to the one after, as long as the current is not at the end of the list.

Algorithm 4.6: Update section by stepping to next

Data: over utilised boolean, current cross section, set of possible cross section
Result: updated section, boolean set to if the section has been updated or not
if over utilised boolean = false **then**
    set updated boolean to false;
    return boolean and exit function;
find position of current cross section in set of possible cross sections;
if position is last in set **then**
    set updated boolean to false;
    return boolean and exit function;
else
    set updated cross section to next in set of possible cross sections;
    set updated boolean to true;
    return updated cross section and boolean and exit function;

4.2.1.3 Method 3: Step-wise incrementation with down-sizing

Method described in section 3.3.1.4. If the member is over utilised this method works exactly as the one described in section 4.2.1.2. If the member is underutilised a check is done if the section on the previous position in the set of cross sections can handle the load. If so, the section is updated to that previous cross section. If not the cross section remains unchanged.

Algorithm 4.7: Update section by stepping to next or previous

Data: over utilised boolean, results from solution, current cross section, set of possible cross section
Result: updated section, boolean set to if the section has been updated or not
find position of current cross section in set of possible cross sections;
if over utilised boolean = true **then**
    if position is last in set **then**
        set updated boolean to false;
        return boolean and exit function;
    else
        set updated cross section to next in set of possible cross sections;
        set updated boolean to true;
        return updated cross section and boolean and exit function;
else
    check utilisation of cross section in previous position in set;
    if utilisation < maximum allowed utilisation **then**
        set updated cross section to cross section in previous position in set;
        set updated boolean to true;
        return updated cross section and boolean and exit function;
    else
        set updated boolean to false;
        return boolean and exit function;

4.2.1.4 Method 4: First acceptable solution from current section

Method described in section 3.3.1.5. This method works by first finding the position of the current cross section in the set. If the element is overutilised the set is then searched through, starting from the current section property and incrementing until a section is found that has a utilisation below the set limit.
If the member is underutilised the set is instead searched through by moving decrementing section property until a section is found that has a utilisation above the set maximum limit. When such a cross section is found, the updated cross section is set to the one for the previous iteration, i.e. the cross section one step further down

in the list that is slightly larger and was checked in the previous iteration.

Algorithm 4.8: Update section checking all from current position

```
Data: over utilised boolean, results from solution, current cross section, set of
      possible cross section
Result: updated section, boolean set to if the section has been updated or not
find position of current cross section in set of possible cross sections;
if over utilised boolean = true then
    set run to true;
    while run do
        check utilisation of cross section of current position in set;
        if utilisation < maximum allowed utilisation then
            set run to false;
            set updated cross section to current position in set;
        else
            change cross section position to next in set;
else
    set run to true;
    while run do
        check utilisation of cross section of current position in set;
        if utilisation < maximum allowed utilisation then
            change cross section position to previous in set;
        else
            set run to false;
            set updated cross section to next position in set;
```

4.2.1.5 Section rotator

As for the section sizer the outer loop, described in algorithm 4.2, calls a inner loop that runs through all elements in the structure and checks if any update is required. This loop is described by algorithm 4.9.

Algorithm 4.9: Check and update all element cross section orientations

```
Data: Current Structure
Result: Updated structure with updated sections orientations, boolean stating if
        any updates has been done
set updated boolean to false;
foreach element in structure do
    try to update element orientation;
    if element has been updated then
        set updated boolean to true;
return updated boolean and updated structure and exit function;
```

As a first check for the section rotation for each element, the cross section type is

analysed. For example a CHS section is directionally independent, why a rotation of the section would result in no change to the structural behaviour. If the orientation of the cross section have an influence a check is made if any improvements for the orientation can be found.
As described in 2.4.2 and 3.3.2 the rotator is using the principle moment to find the orientation for the cross section. As this is an iterative procedure, the method need to have some convergence criterion. The criterion used is to check if the ratio between the major and minor axis bending moment is sufficiently small. If the ratio is small enough no change will be made, otherwhise the cross section will be updated.

Algorithm 4.10: Update the element orientation based on principle moment

Data: results from solution, current cross section
Result: new element orientation vector, boolean stating if update is found
if cross section is directional dependant **then**
    find the position with highest principle moment, using eq 2.93, from the worst load case for the element;
    get size of major and minor axis moment from position;
    if abs(maxMz/maxMy) < small value **then**
        no update needed;
        set updated boolean to false;
        return update boolean and exit function;
    else
        calculate new element orientation vector using eq 2.94;
        set updated boolean to true;
        return update boolean, new element orientation vector and exit function;
else
    no update needed;
    set updated boolean to false;
    return update boolean and exit function;

4.2.2 Mode shape optimiser

The mode shape optimiser is described in section 3.3.3. unlike the iterative section sizer, it only uses its solver, here the eigensolver described in section 3.2.1.3.2 once. This is done at the start of the process, where after all modes of interest, chosen by the user, are evaluated. The cross sections found, using algorithm 4.7 for each mode is stored by the elements. After all modes have been checked the largest cross section

for each element is chosen. The whole processes is described in algorithm 4.11.

Algorithm 4.11: Mode shape optimiser

Data: Current Structure, modes to optimise for, external scale factor (s_{ext})
Result: Updated structure with updated sections
Calculate eigenvalues and modes for the structure using the eigensolver;
Get the first (lowest) eigenvalue (λ_1);
foreach mode to optimise for **do**
    Calculate the ratio between the first and current eigenvalue ($R = \lambda_i/\lambda_1$);
    Calculate section forces for the mode;
    Find the lowest utilisation for the mode (η_{min});
    Calculate the scalig factor ($s = s_{ext} \cdot 1/\eta_{min} \cdot 1/R$);
    Scale all element section forces with s;
    Find the smallest suitable cross section for the scaled forces;
Update all elements cross to the largest ones found in the loop;

4.2.3 Combined section sizer

The combined section sizer works by a combination of the mode shape optimiser and the iterative section sizer, as described in section 3.3.4. This process is described by algorithm 4.12.

Algorithm 4.12: Combined mode sizer and section sizer

Data: Current Structure, modes to optimise for)
Result: Updated structure with updated sections
Run mode shape optimiser(algorithm 4.11);
Run section sizer(algorithm 4.2);

5
Case Studies

5.1 Small 2d frame

A test was conducted on a small 2d frame of 2x2 bays, see fig. 5.1. Each element has a length of $L = 3m$. All node connections were modelled as stiff.

5.1.1 Geometry

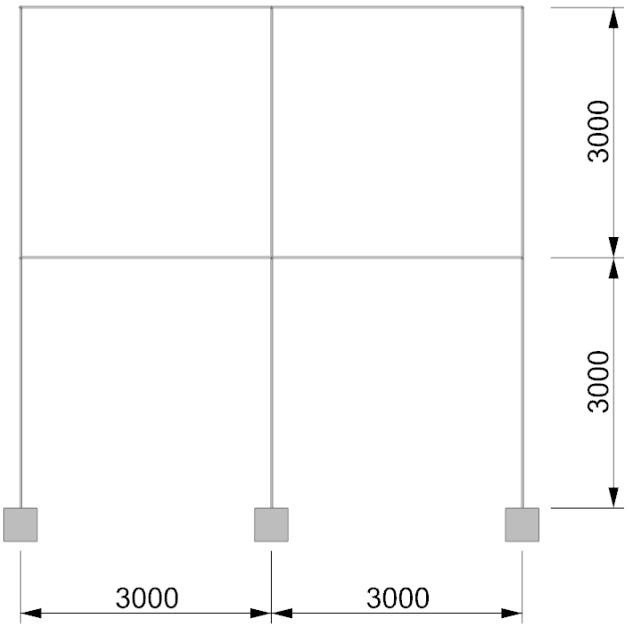
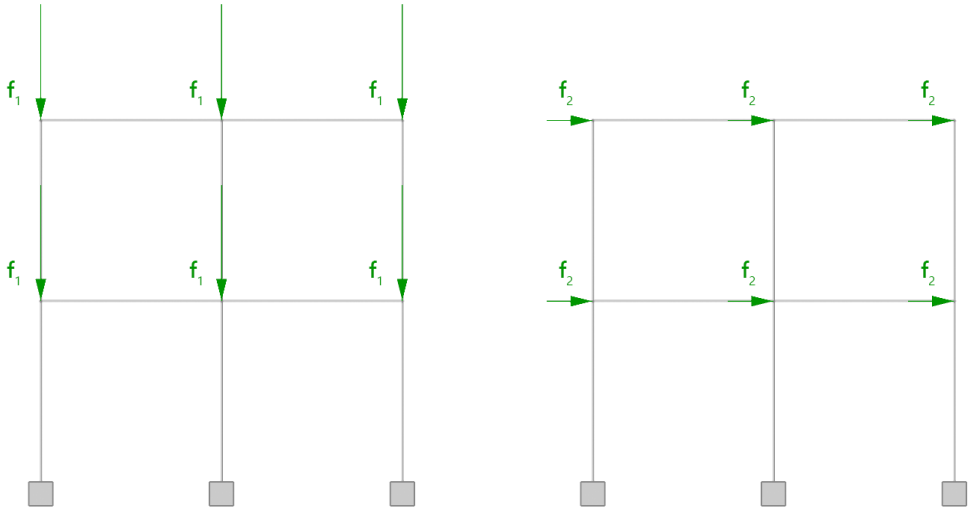


Figure 5.1: The analysed frame with dimensions and boundary conditions.

5.1.2 Boundary conditions

The element is rigidly connected to the ground in all three columns. All out-of-plane actions have been restrained as well.

5.1.3 Loads



(a) Vertical load, f_1=50 [kN] **(b)** Horisontal load, f_2=10 [kN]

Figure 5.2: Load combinations on the structure

The loads are presented in figure 5.2. The frame was loaded in-plane with relatively small forces. The forces were applied as separate loads rather than as one combined. Both load combinations also include self-weight.

5.1.4 Optimisation settings

For the iterative section sizers, the following settings have been used:

Setting	Value
Initial cross section	RHS30x30x8
Minimum utilisation	80%
Maximum utilisation	100%
Maximum iterations	100

Table 5.1: Optimisation settings for the 2d frame

The sections used for the iterative section sizers can be found in Appendix B

5.2 3d frame

5.2.1 Geometry

The geometry of the structure is as follows (see fig. 5.3):

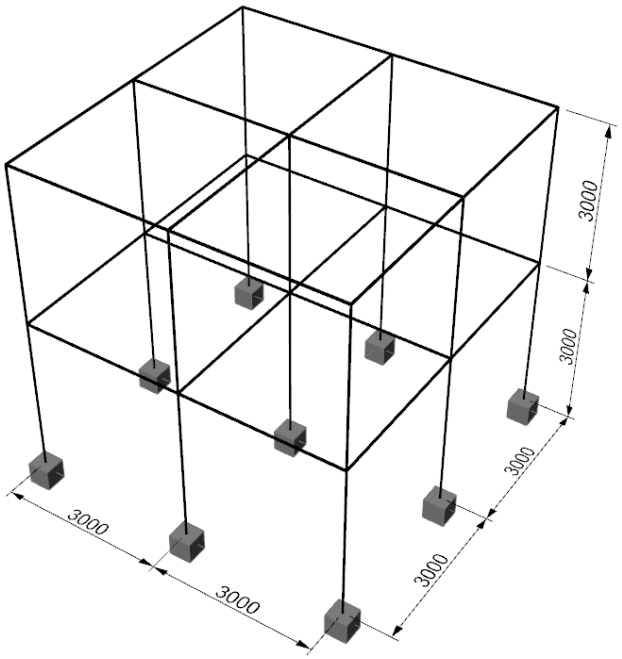


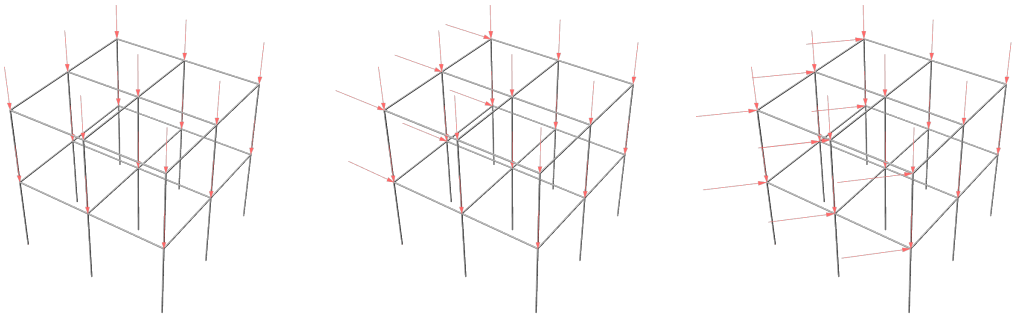
Figure 5.3: The structure consist of two storeys and is symmetrically two bays wide and two bays deep. Each element is 3 meters long, making the entire structure 6x6x6 meters.

5.2.2 Boundary conditions

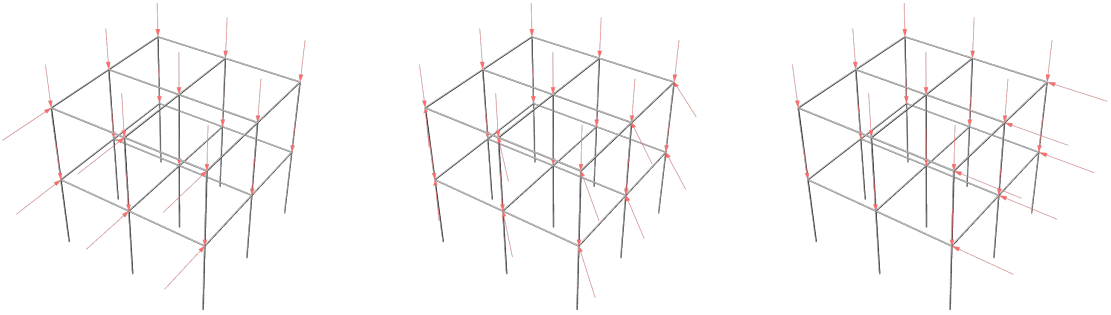
The structure is fixed in all translations and all rotations at the bottom nodes (see fig. 5.3). However, the node stiffness method may alter the rotational stiffnesses in the major and minor axes for any element.

5.2.3 Loads

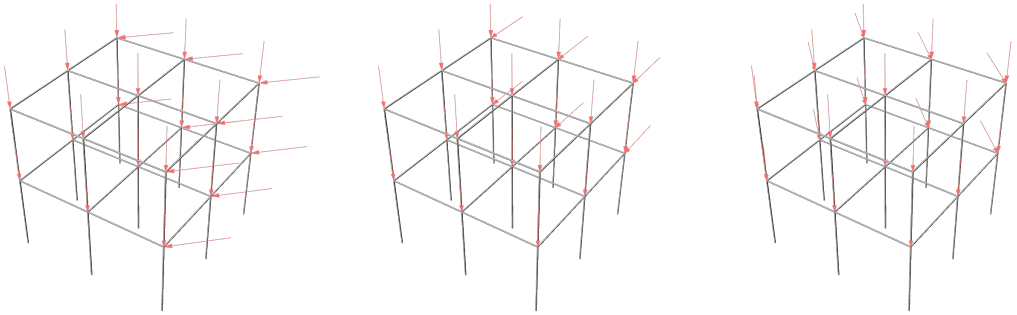
The structure is subjected to several combinations of vertical loads and wind load. The wind loads are subjected to the nodes at the exposed side. All load combinations also include self-weight.
The loads are presented in fig. 5.4.



(a) LC1: Vertical **(b)** LC2: Wind W **(c)** LC3: Wind SW



(d) LC4: Wind S **(e)** LC5: Wind SE **(f)** LC6: Wind E



(g) LC7: Wind NE **(h)** LC8: Wind N **(i)** LC9: Wind NW

Figure 5.4: The load combinations applied to the structure. Vertical loads are 50 kN in the pure vertical case (6.13a) and 10 kN in all other cases. Horizontal loads are 5 kN. All load combinations also include self-weight. Loads not drawn to scale.

5.2.4 Optimisation settings

For the iterative section sizers, the following settings have been used:

Setting	Value
Initial cross section	RHS30x30x8
Minimum utilisation	80%
Maximum utilisation	100%
Maximum iterations	100

Table 5.2: Optimisation settings for the 3d frame

The sections used for the iterative section sizers can be found in Appendix B

5.3 Dome

5.3.1 Geometry

The geometry of the structure consists of a quadratic grid dome with an outer perimeter ring. (see fig. 5.5):

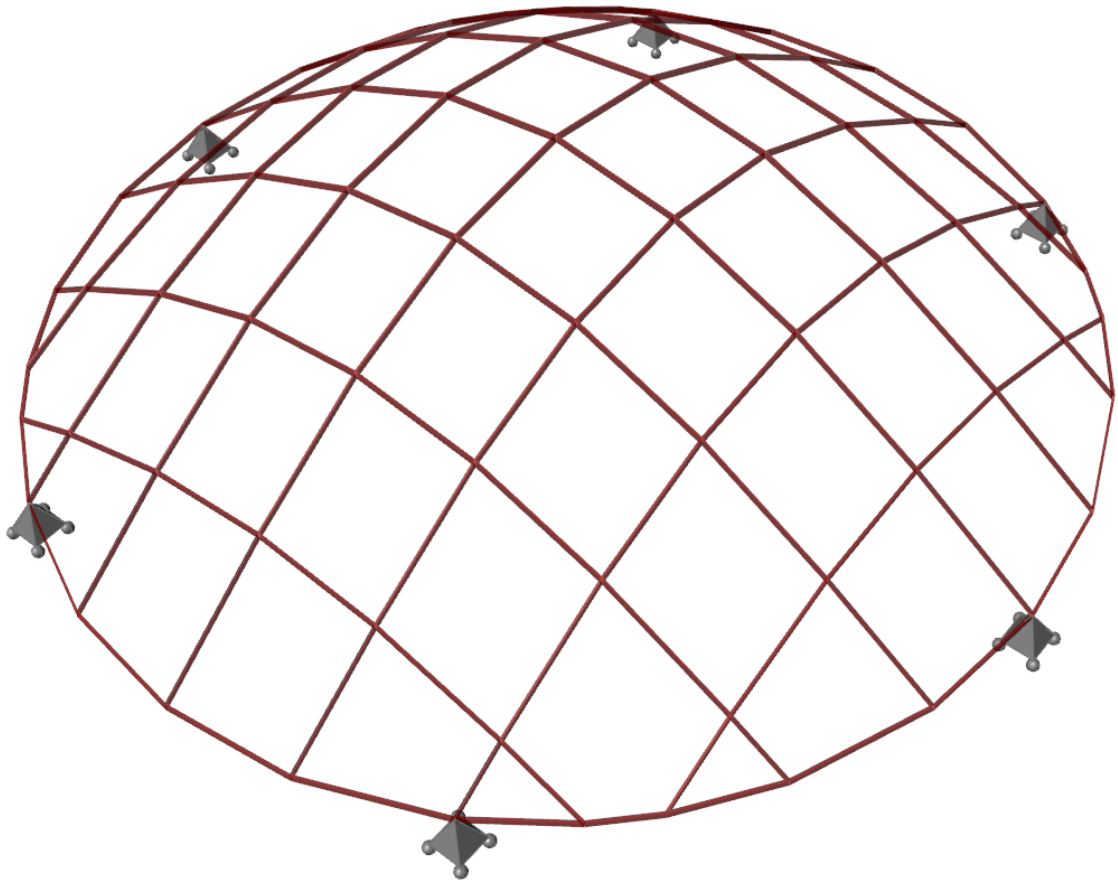


Figure 5.5: The structure consist of a dome resting on six pinned roller supports.

The dome has a diameter of 30 m, with an average length of each element of 3.8 meters.

5.3.2 Boundary conditions

A dome structure with vertical and horizontal supports along it's boundary is a very efficient structure. If the boundary conditions are changed to something not as ideal, though, the structure might change its load bearing behaviour. For the ideal boundary condition case, the dome will carry vertical load in primarily normal force action. To get a structure with less ideal conditions, the dome is instead sat on six rolling supports, only restraining vertical movement. This should change the load bearing behaviour from primarily normal force action to primarily bending moment action. Stiffening the perimeter ring of the dome could be beneficial as it could

make the grid of the dome get a behaviour closer to the ideal boundary condition case.
As the six rolling supports does not prevent horizontal movement and rotation and rotation around the vertical axis, the middle node of the structure is restrained to handle these movements.

5.3.3 Loads

The point of interest for the investigation of this dome like structure is to evaluate how different stiffness distributions can change the load bearing behaviour between bending action and normal force action. To investigate this, five load cases are set up, all solely consisting of vertical loads. The load cases consists of point loads of $10[kN]$. The distribution can be seen in figure 5.6

5.3.4 Cross sections

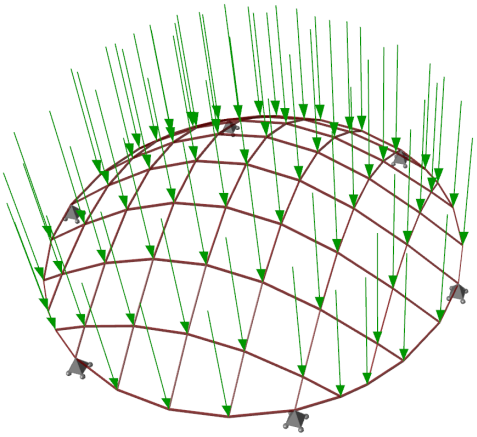
For the section sizer procedures various RHS cross sections are set up as possible candidates for the elements to use. The size of these cross sections range from 30 mm to 650 mm in height and width, all 6 mm thick. The cross sections are limited to an aspect ratio ($height/width$) between 1 and 4. For a full list of the used cross sections refer to appendix C.

5.3.5 Optimisation settings

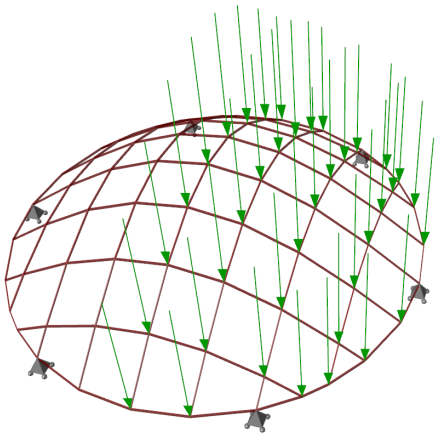
The following settings has been used for the optimisation procedure if nothing else is mentioned, the cross section chosen to start with is the smallest on the list (see appendix C):

Setting	Value
Initial cross section	RHS30x30x6
Minimum utilisation	80%
Maximum utilisation	100%

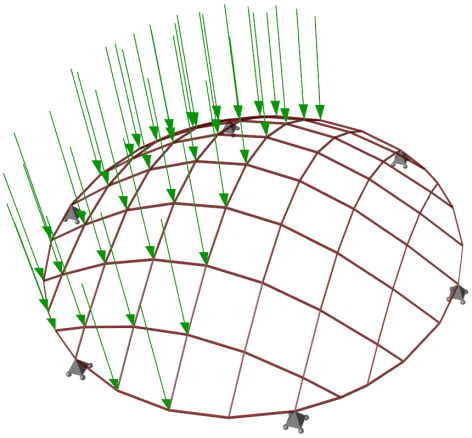
Table 5.3: Optimisation settings for dome structure



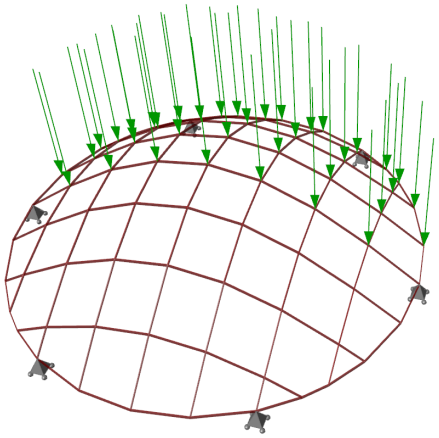
(a) Load combination 1. Whole dome loaded



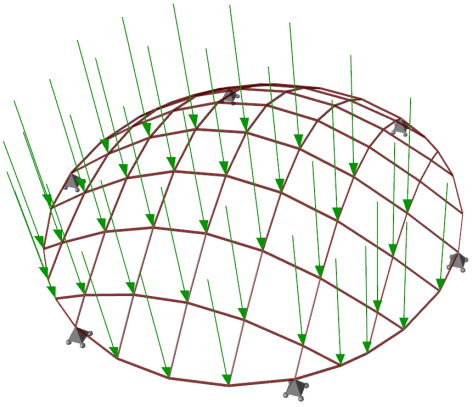
(b) Load combination 2. Half dome loaded



(c) Load combination 3. Half dome loaded



(d) Load combination 4. Half dome loaded



(e) Load combination 5. Half dome loaded

Figure 5.6: Dome load cases.

Figure 5.7: King Abdullah Financial District Metro Station. Picture courtesy of Zaha Hadid Architects [Zaha Hadid (2016)].

5.4 KAFD Metro Station

For the final object to study, a real project was chosen. King Abdullah Financial District Metro Station is a part of a massive infrastructure project currently under construction in Ar Riyadh, Kingdom of Saudi Arabia [Zaha Hadid (2016)]. The station will be one of the major stations for the new Metro in Ar Riyadh, serving three lines. It is designed by Zaha Hadid Architects and Buro Happold Engineering, with facade design by NewTecnic.

This project is of course extremely complex and requires lots of skill and time to design completely. At the time when this thesis was conducted, this project was in the final design stages after several years of development. It was selected as a good test subject because of its complexity, which meant that the methods would really be put to the test. A real project would also hopefully not have the idealised nature that a test structure has, which could highlight the strengths and weaknesses of the methods.

However, as the entire project contains more than 20 000 elements it would take very long time to analyse. Therefore a smaller part of the structure was selected for the case study, with the intention that the complexity of that part alone would be sufficiently complex to be useful for a study. The part that was selected is one of the entryways for the trains. It was chosen because it was a part that worked predominantly in isolation and was relatively unaffected by the rest of the structure, but still being of unconventional structural design. It also contains a transition between a free-form part and a more conventional part.

Figure 5.8: Night render of King Abdullah Financial District Metro Station. The selected part for the case study is where the train at the left side is entering the station. Picture courtesy of Zaha Hadid Architects [Zaha Hadid (2016)].

5.4.1 Geometry

The evaluated part of the station consist of a 3D-frame steel structure. It has a span of ca 22 meters over it's wides parta and 13 meters over it's smallest part, and is ca 23 meters long. The geometry can be seen in fig. 5.9.

5.4.2 Boundary conditions

The structure has been modeled with pinned supports along it's sided. The supports can be seen as grey pyramids in fig. 5.9.

5.4.3 Loads

Since several simplifications had to be done, both in the geometry of the case study and in the implementations of *CIGull*, the loads used for the actual design of the structure generated a very low utilisation. For instance, the driving factor for the design of the actual design may have been deflection limits, dynamic loads, connection design or buckling, all phenomena not currently implemented in *CIGull*. Therefore, the original loads applied to the structure have been scaled so that they generate a utilisation of about 100%, thus making the process of optimisation more

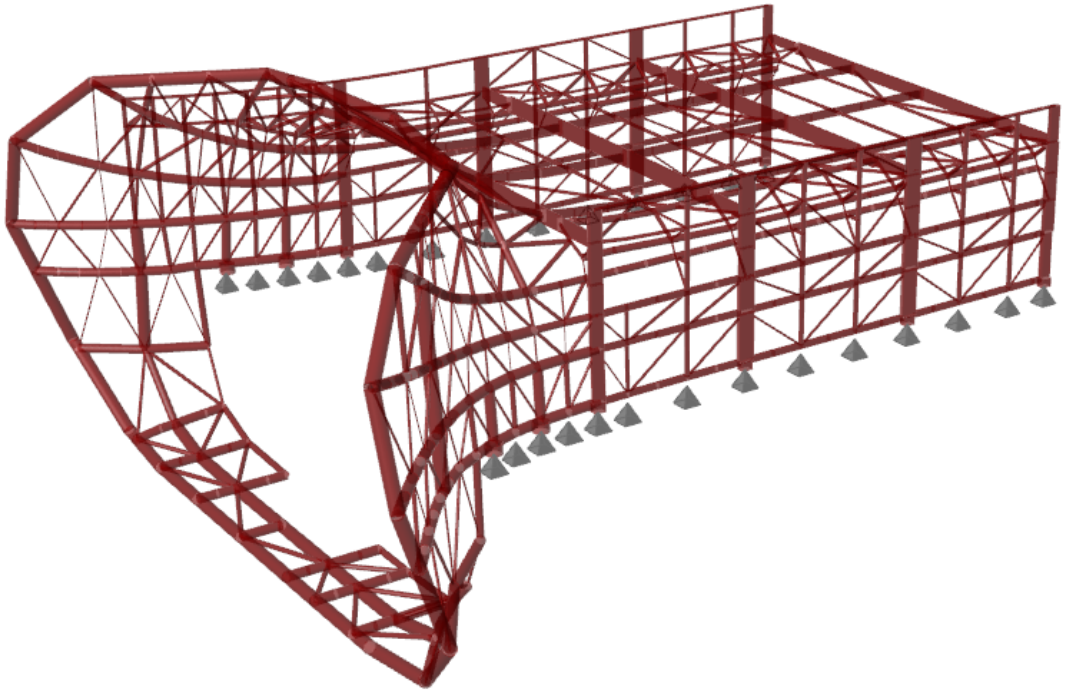
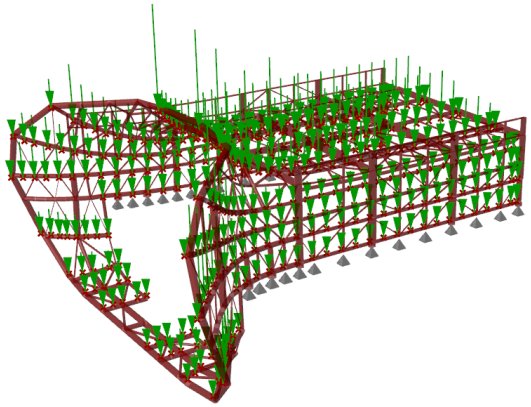


Figure 5.9: Geometry of the evaluated part. Boundary conditions are modeled as pinned supports.

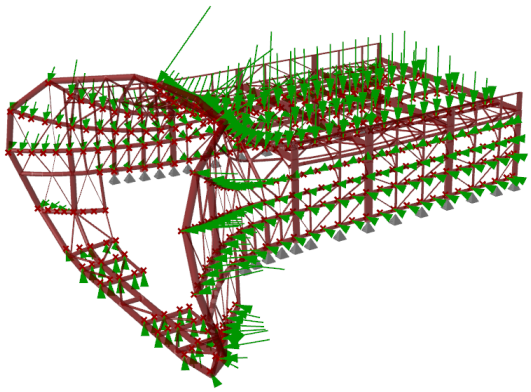
similar to the original design. This of course means that the obtained designs are not comparable to the original, but they are still comparable to each other.
The applied loads are all modeled as point loads. Their amplitude correspond to their tributary area, which means that the further the load applications points are from each other, the larger the load will get. The load situations correspond to dead loads from panels carried by the structure (fig. 5.10a) and wind pressure from the trains entering and exiting the station (fig. 5.10b and fig. 5.10c).

5.4.4 Cross sections

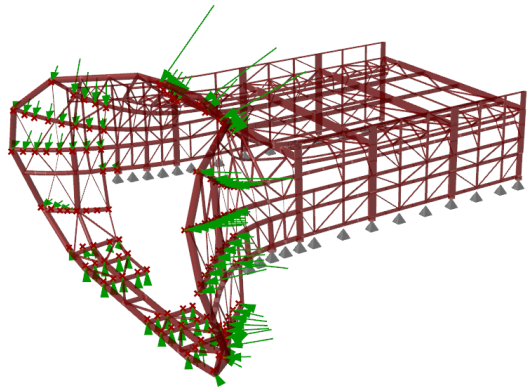
The current state of the geometry of the structure consists of three types of cross sections, RHS:s, CHS:s and I-beams. As CIFem currently do not support I-beams, they where changed to RHS-beams with similiar sections properties. For the optimisation procedures cross sections already used in the structure is taken as possible candidates. Members being RHS:s at the start will coose from the available set of RHS cross sections to keep the section type intakt. The same is true for elements with initial CHS section types. For full lists of the cross sections used refer to appendix D;



(a) Load 1



(b) Load 2



(c) Load 3

Figure 5.10: The three different load setups for the structure

5.4.5 Optimisation settings

The following settings has been used for the optimisation procedure if nothing else is mentioned, the cross section chosen to start with is the smallest on the lists (see appendix D):

Setting	Value
Initial cross section RHS	RHS80x80x5
Initial cross section CHS	CHS139.7x5
Minimum utilisation	80%
Maximum utilisation	100%
Maximum iterations	150

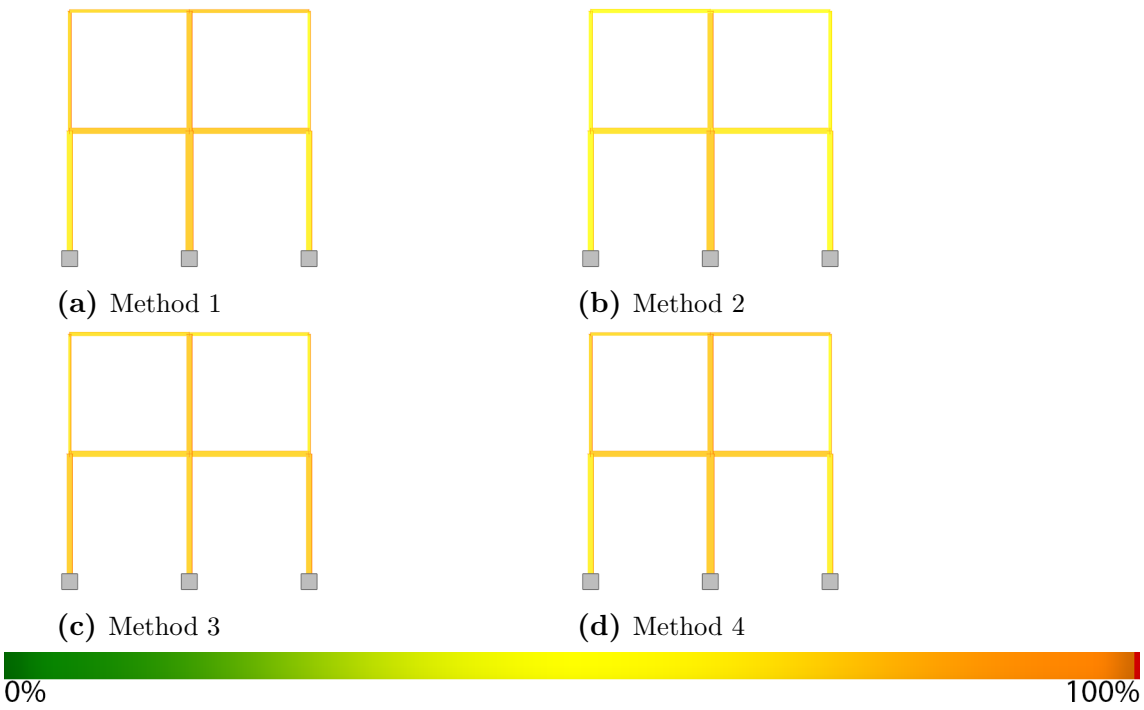
Table 5.4: Optimisation settings for KAFD metro

6
Results

6.1 Small 2d frame

6.1.1 Section sizer

The 2d frame from fig. 5.1 was optimised using the four section sizing tools described in section 3.3.1. Allowing rotations isn't relevant in this case, since the problem is two dimensional. The following results were obtained:



(a) Method 1 **(b)** Method 2

(c) Method 3 **(d)** Method 4

Figure 6.1: Resulting structures from the section sizer, using different modes as input

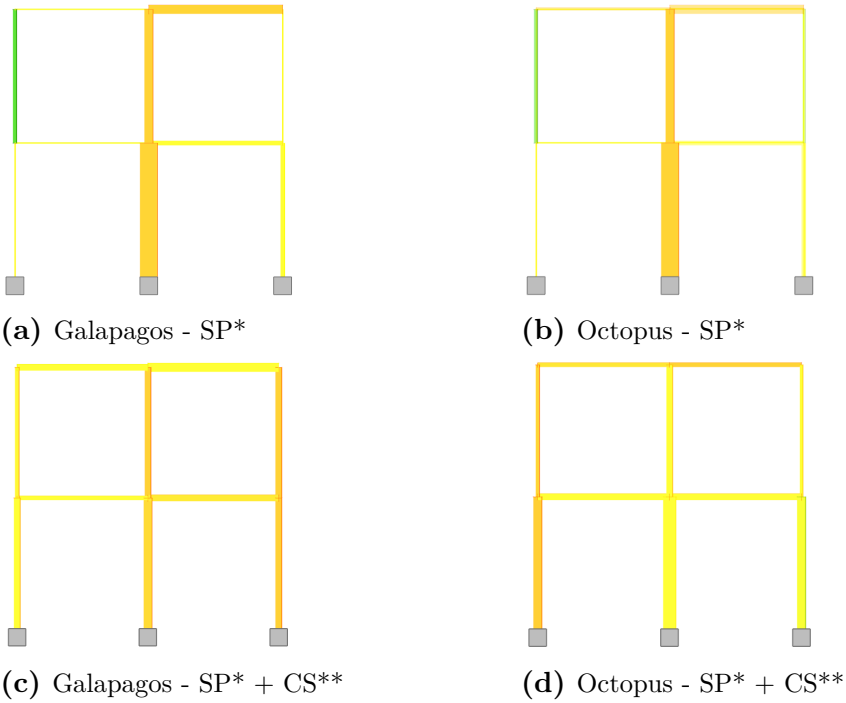
Optimiser	Type	Weight [kg]	Calc. time	Iter.	Utilisation [%]		
					Min	Max	Avg
Section sizer (no rotation)	1	**677**	0.9 s	5	81.6	99.3	92.3
	2	718	2.2 s	35	65.3	97.6	79.6
	3	680	2.2 s	33	84.0	98.7	91.6
	4	**677**	**0.8 s**	5	81.6	99.3	92.3

Table 6.1: Results for the 2d frame from fig. 5.1.

6.1.2 Genetic algorithms

Optimiser	Parameters	Weight [kg]	Calc. time
Genetic algorithm (galapagos)	Section properties	560	1 h
	Section properties + node stiffness	796	1 h
Genetic algorithm (octopus)	Section properties	**512**	1 h
	Section properties + node stiffness	785	1 h

Table 6.2: Results for the 2d frame from fig. 5.1 optimised using genetic algorithms using either only the section properties or section properties and node stiffness as its genome.



(a) Galapagos - SP*

(b) Octopus - SP*

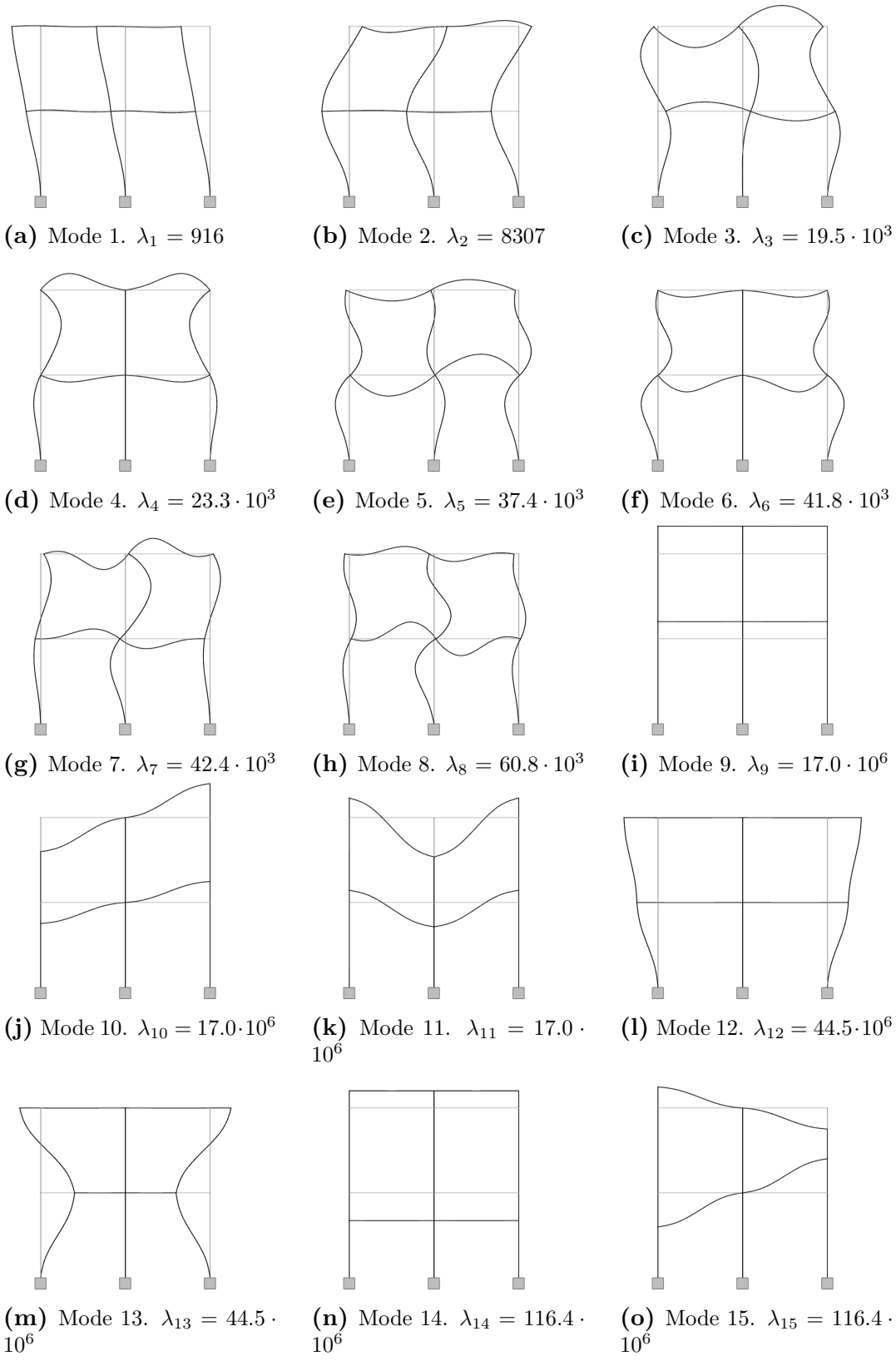
(c) Galapagos - SP* + CS**

(d) Octopus - SP* + CS**

Figure 6.2: Resulting structures from the genetic algorithms, optimised for different variables
* SP - Section properties
** CS - Connection stiffness

The genetic algorithms did find some good results, but when optimising for connection stiffness as well the additional parameters made it very slow to converge, likely because the connection stiffness do not affect the fitness directly. It is assumed that if the algorithm was left to run for a longer time, a better result would have been achieved.

6.1.3 Canonical stiffnesses and eigenmodes



(a) Mode 1. $\lambda_1 = 916$ **(b)** Mode 2. $\lambda_2 = 8307$ **(c)** Mode 3. $\lambda_3 = 19.5 \cdot 10^3$

(d) Mode 4. $\lambda_4 = 23.3 \cdot 10^3$ **(e)** Mode 5. $\lambda_5 = 37.4 \cdot 10^3$ **(f)** Mode 6. $\lambda_6 = 41.8 \cdot 10^3$

(g) Mode 7. $\lambda_7 = 42.4 \cdot 10^3$ **(h)** Mode 8. $\lambda_8 = 60.8 \cdot 10^3$ **(i)** Mode 9. $\lambda_9 = 17.0 \cdot 10^6$

(j) Mode 10. $\lambda_{10} = 17.0 \cdot 10^6$ **(k)** Mode 11. $\lambda_{11} = 17.0 \cdot 10^6$ **(l)** Mode 12. $\lambda_{12} = 44.5 \cdot 10^6$

(m) Mode 13. $\lambda_{13} = 44.5 \cdot 10^6$ **(n)** Mode 14. $\lambda_{14} = 116.4 \cdot 10^6$ **(o)** Mode 15. $\lambda_{15} = 116.4 \cdot 10^6$

Figure 6.3: The first nine eigenmodes for the 2d frame with uniform sections.

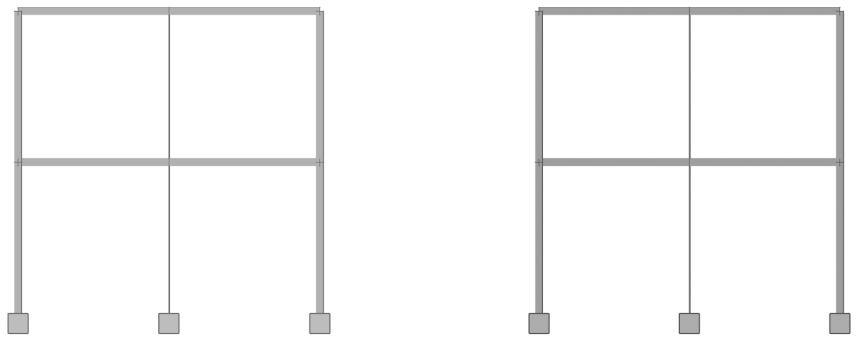
Mode	Eigenvalue	Normalised eigenvalue
1	916	1.0
2	8307	9.07
3	$19.5 \cdot 10^3$	21.31
4	$23.3 \cdot 10^3$	25.51
5	$37.4 \cdot 10^3$	40.9
6	$41.8 \cdot 10^3$	45.63
7	$42.4 \cdot 10^3$	46.31
8	$60.8 \cdot 10^3$	66.39
9	$17.0 \cdot 10^6$	18555
10	$17.0 \cdot 10^6$	18560
11	$17.0 \cdot 10^6$	18569
12	$44.5 \cdot 10^6$	48579
13	$44.5 \cdot 10^6$	48590
14	$116.4 \cdot 10^6$	127178
15	$116.4 \cdot 10^6$	127182

Table 6.3: Eigenvalues for the first fifteen modes, normalized to the first eigenvalue.

When analysing the eigenmodes in figure 6.3 there are some things that can be noted. First, mode 1 corresponds very well to the horisontal load combination (fig 5.2b). What this means is that the most sensitive deformation pattern of the structure corresponds to one of the actual load combinations, which is not ideal. The vertical load case has a corresponding eigenmode in mode 9, but that is less of an issue given the large increase in eigenvalue which can be identified between mode 8 and nine. The reason for this large increase is the introduction of axial deformations in the elements.

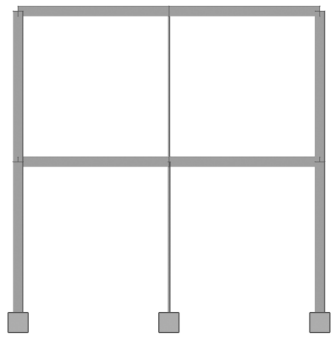
6.1.4 Mode section sizer

Looking at the modes in fig. 6.3, the first two are characterised by lateral deformation, modes 3-8 by various rotations, and modes 9-15 by axial deformations. Choosing the modes for the mode section sizer is therefore done for these 'groups', and one combination.

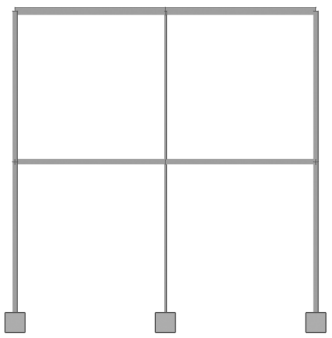


(a) Mode 1 - 2

(b) Modes 3 - 8



(c) Modes 9 - 18



(d) Modes 1 - 8

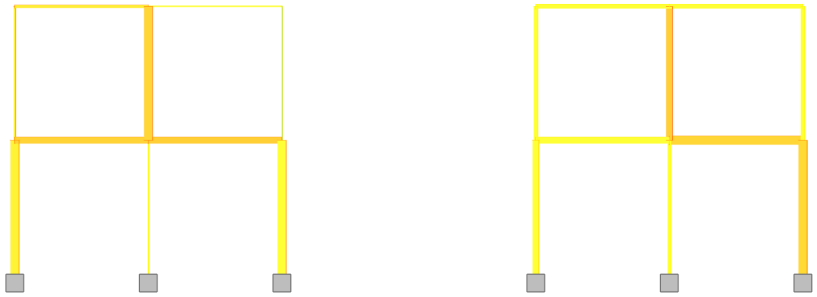
Figure 6.4: Resulting structures from the mode section sizer, using different modes as input

6.1.5 Combined section sizer

Optimiser	Type	Weight [kg]	Calc. time	Iter.	Utilisation [%]		
					Min	Max	Avg
Section sizer (no rotation) modes 1-8	1	**639**	4.5 s	18	33.5	99.2	75.9
	2	796	**3.4 s**	27	45.1	98.3	67.9
	3	647	5.2 s	46	31.8	97.9	73.2
	4	650	4.3 s	18	42.4	97.9	76.4
Section sizer (no rotation) modes 1 + 9	1	575	3.8 s	23	26.8	86.1	62.3
	2	710	**1.2 s**	14	18.8	95.5	77.1
	3	665	2.4 s	30	57.8	98.4	85.2
	4	**568**	3.0 s	19	33.8	99.3	66.7

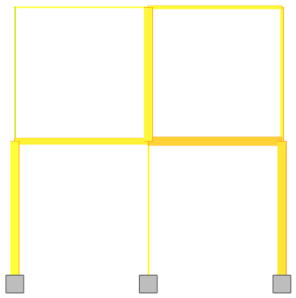
Table 6.4: Results from the combined section sizer.

The combined section sizer was run for a set of different modes in the mode section sizer. In figure 6.5 the first 8 modes were used in order to generate a "stable" structure, in figure 6.6 the two most that most corresponded to the load combinations.

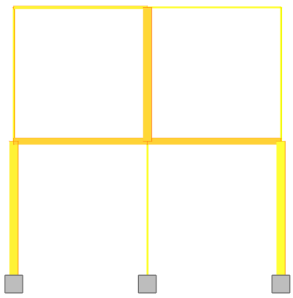


(a) Method 1

(b) Method 2

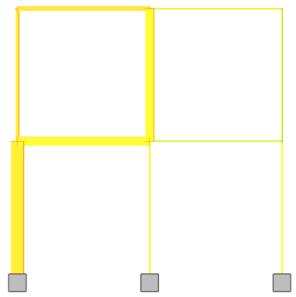


(c) Method 3

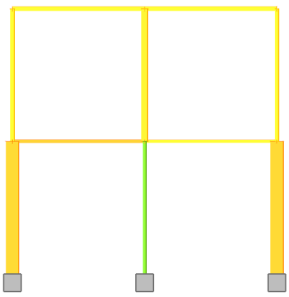


(d) Method 4

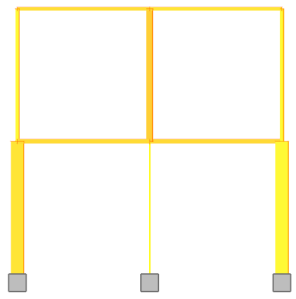
Figure 6.5: Resulting structures from the combined section sizer, starting with a geometry scaled for modes 1-8.



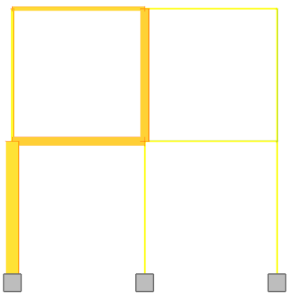
(a) Method 1



(b) Method 2



(c) Method 3



(d) Method 4

Figure 6.6: Resulting structures from the combined section sizer, starting with a geometry scaled for modes 1 and 9, the modes that most correspond to the applied loads.

6.1.6 Comments

It can be noted that although the genetic algorithms produce end results that are better than the ones produced by the section sizers, they do it at a much slower speed. For this relatively simple example the genetic algorithm had a calculation time of about one hour, compared to the section sizers and combined section sizer which converged in a matter of seconds.

The problem with the genetic algorithms is that they basically are blind while searching for good solutions in an enormous search space. It is thus expected for the process to be slow. However, most of the computation time is spent searching for an 'acceptable' solution, and only a small part of the time is spent on finding a 'good' solution. If one could cut this first initial calculation part it would likely generate a large increase in speed. This can be done by letting the outcome of the iterative section sizer be the input of the genetic algorithm. Octopus (see section 3.3.5) includes an option of "run with presets", allowing the user to add one genome to the set of the randomly generated ones. By adding a genome with very good fitness and using combined settings of high mutation probability and low mutation rate (for more information on these, [Bedney (2016)] can be consulted) the result obtained from the iterative section sizer may be increased very quickly. Since the majority of the generations of the genetic algorithms were spent searching for a fitness *as* good as the ones obtained from the iterative section sizers (rather than actually improving them), eliminating that part of the process increases the efficiency by a huge amount! The downside with this technique is however that the point of using genetic algorithms, to explore unexpected solutions, is somewhat reduced.

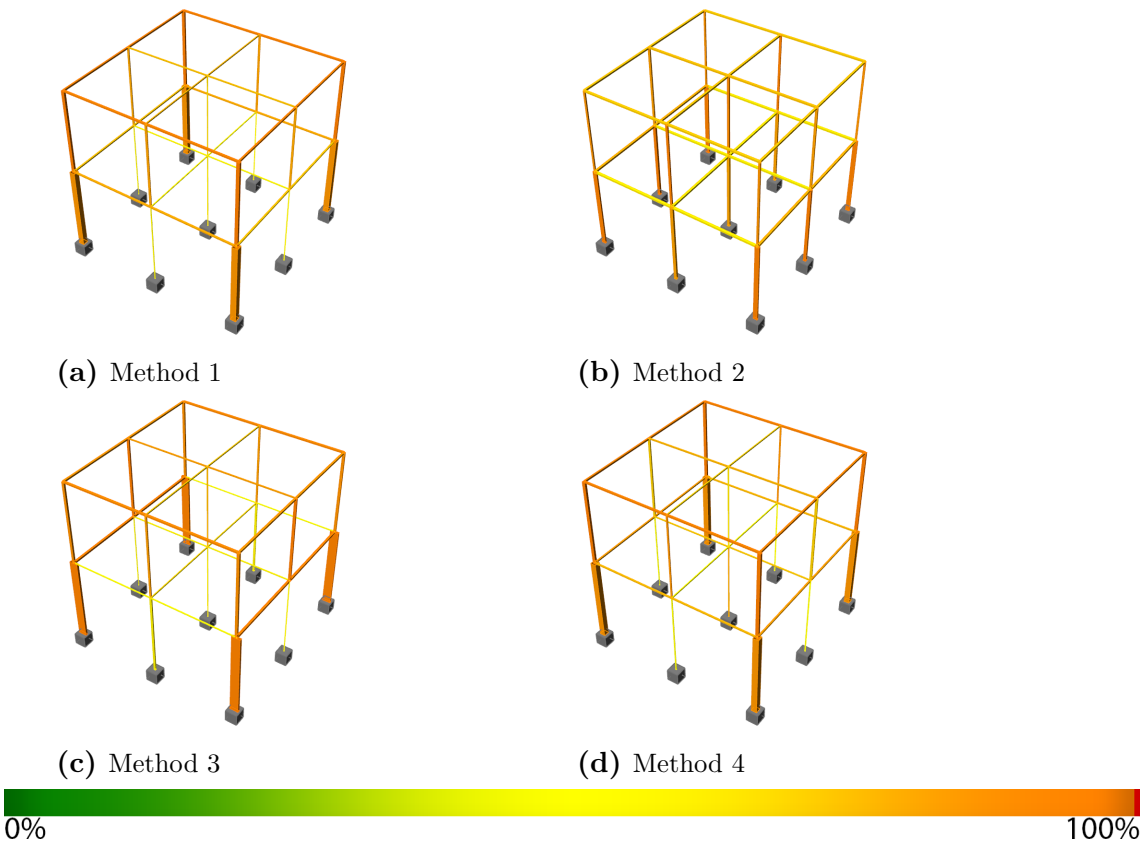
The perhaps most interesting result from this studied example was that the method that generated the best results was the combined section sizer using section sizer method 4 (see fig. 6.6d). Choosing the first eight modes resulted in a more homogenous structure, but choosing the relevant modes for the current load combinations resulted in a more effective structure.

6.2 3d frame

6.2.1 Section sizer and rotator

Optimiser	Type	Weight [kg]	Calc. time	Iter.	Utilisation [%]		
					Min	Max	Avg
Section sizer (no rotation)	1	**1688**	3.6 min	21	57.6	98.7	79.5
	2	1965	**1.4 min**	27	62.5	97.5	79.2
	3	1751	4.2 min	66	47.7	99.6	79.5
	4	1718	3.1 min	23	46.3	99.7	79.3
Section sizer (incl. rotation)	1	**1587**	6.8 min	36	37.1	100	76.0
	2	1845	**3.8 min**	38	65.0	95.0	77.8
	3	1733	8.5 min	77	39.0	98.5	83.6
	4	1676	5.1 min	34	46.2	99.3	78.9

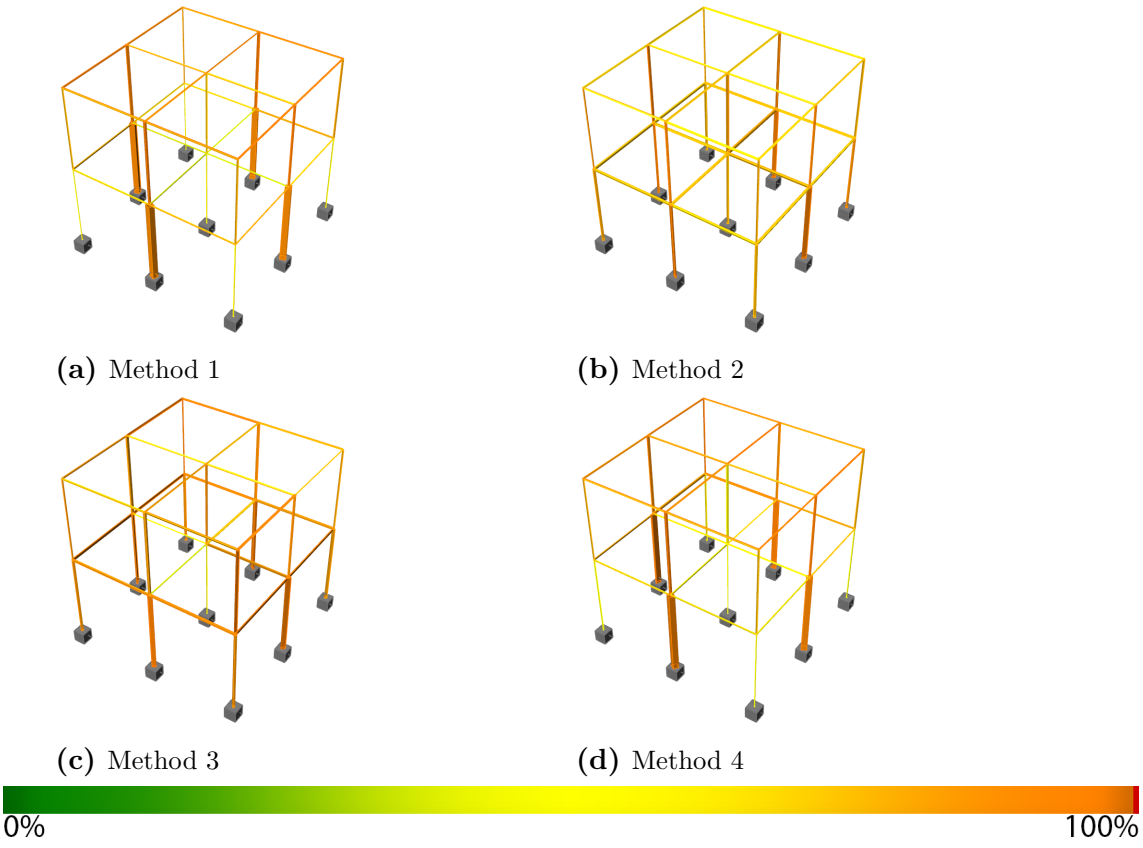
Table 6.5



(a) Method 1 **(b)** Method 2

(c) Method 3 **(d)** Method 4

Figure 6.8: Resulting structure from iterative section sizer - not allowing rotations.



(a) Method 1 **(b)** Method 2

(c) Method 3 **(d)** Method 4

Figure 6.9: Resulting structure from iterative section sizer - allowing rotations.

When analysing the results for the section sizer with and without rotation it can be noted that method 2 is the fastest one for both cases. This isn't surprising since it performs less checks than the others. Including rotations improved the results for all methods. Another thing to notice is that the rotating and non-rotating methods have produced one family of results each (cf. figs. 6.8 and 6.9).

6.2.2 Genetic algorithms

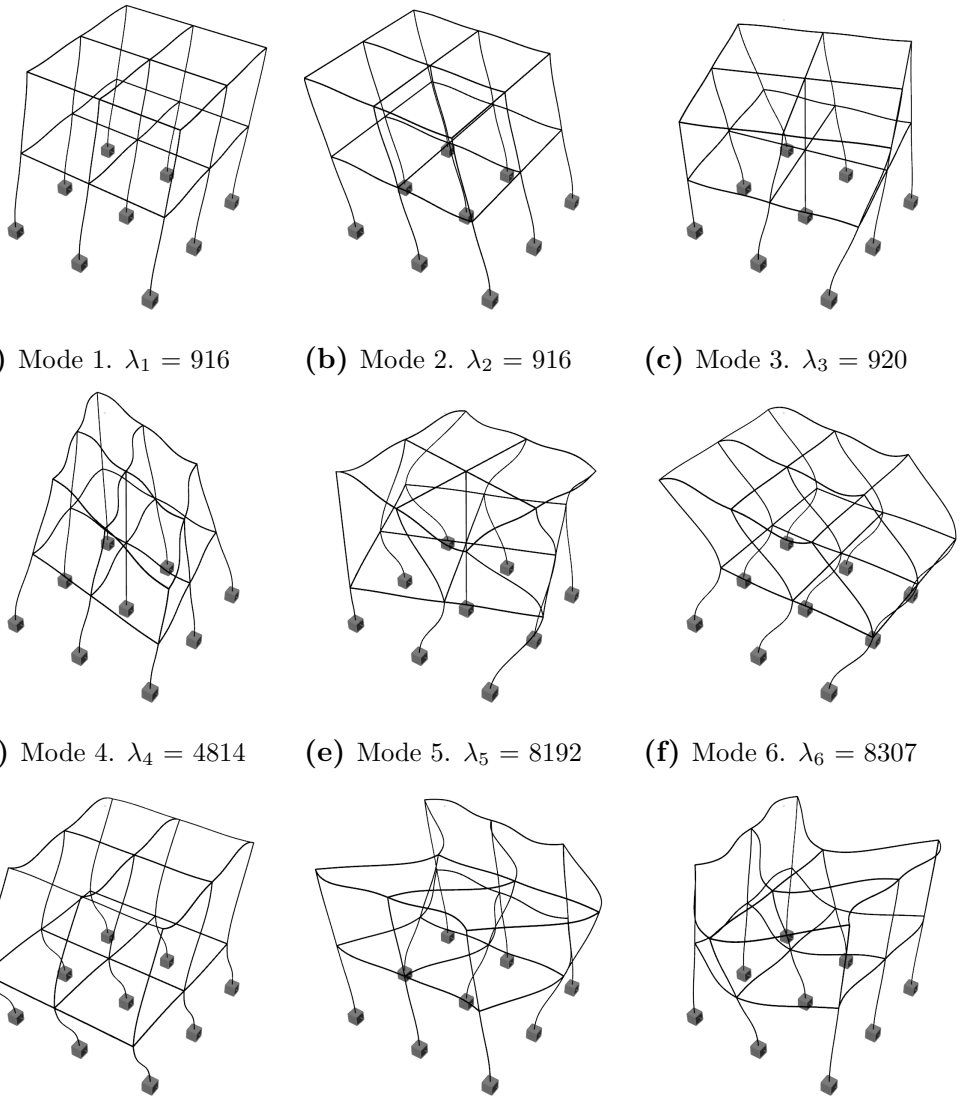
Optimiser	Parameters	Weight [kg]	Generations	Calc time
GA (Octopus)	Section properties	2399	17	2 h
GA with presets*		**1679**	11	1 h**

Table 6.6: Results from an optimisation process using genetic algorithms.
* Genetic algorithm using Octopus starting from the resulting design from section sizer type 1 (see 3.3.1.2)
** The best solution was found after approximately 5 mins, but the algorithm was allowed to continue running for one hour before aborting.

The genetic algorithms were run using only the section properties as input, because of the time increase it would cause (see the 2d frame example, section 6.1). This time the results (see table 6.6) showed that the genetic algorithm didn't achieve a better result than the section sizers (see table 6.5) in the test time. It is expected

that a better result would have been possible to achieve if enough time would have been given, as well as optimal algorithm settings. Using a preset geometry did allow Octopus to produce a better result than the input data, but it is still a slow process which doesn't improve the end result by a great deal.

6.2.3 Canonical stiffnesses and eigenmodes



(a) Mode 1. $\lambda_1 = 916$ **(b)** Mode 2. $\lambda_2 = 916$ **(c)** Mode 3. $\lambda_3 = 920$

(d) Mode 4. $\lambda_4 = 4814$ **(e)** Mode 5. $\lambda_5 = 8192$ **(f)** Mode 6. $\lambda_6 = 8307$

(g) Mode 7. $\lambda_7 = 8307$ **(h)** Mode 8. $\lambda_8 = 8986$ **(i)** Mode 9. $\lambda_9 = 8986$

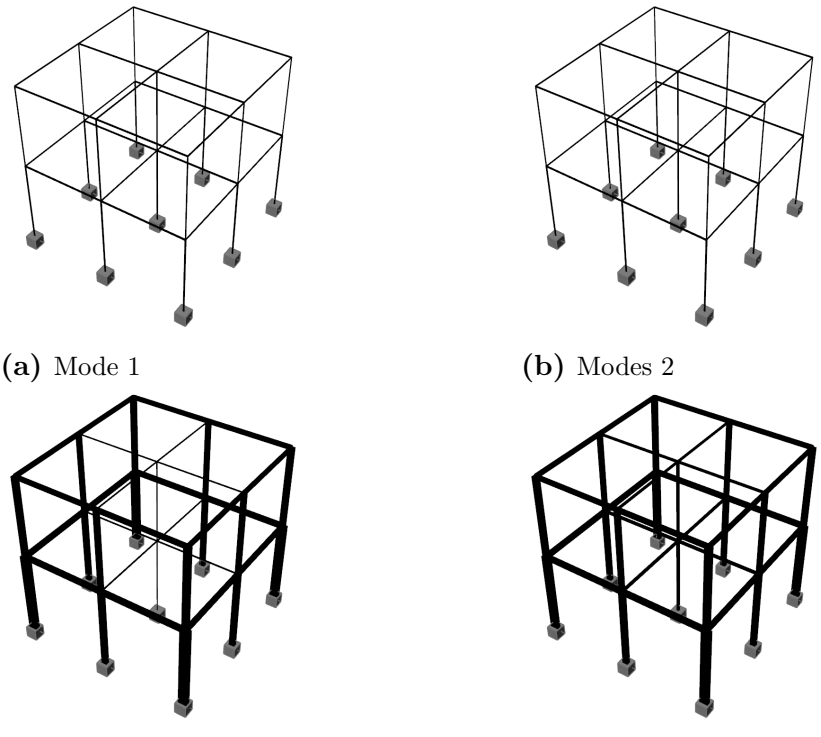
Figure 6.10: The first nine eigenmodes for the 3d frame with uniform sections. Deformations scaled 5 times.

Mode	Eigenvalue	Normalised eigenvalue
1	916	1.0
2	916	1.0
3	920	1.01
4	4814	5.26
5	8192	8.95
6	8307	9.07
7	8307	9.07
8	8986	9.82
9	8986	9.82

Table 6.7: Eigenvalues for the first nine modes, normalized to the first eigenvalue. A longer version of this list can be found in Appendix E.

In table 6.7 the first 9 eigenmodes are presented. A list of the first 67 eigenvalues is presented in Appendix E. For this problem, the eigenvalues of the first three eigenmodes were very similar, their normalised values very close to one. After that the eigenvalues 4-66 are somewhat similar, with a large increase after that.

6.2.4 Mode section sizer



(a) Mode 1

(b) Modes 2

(c) Modes 3

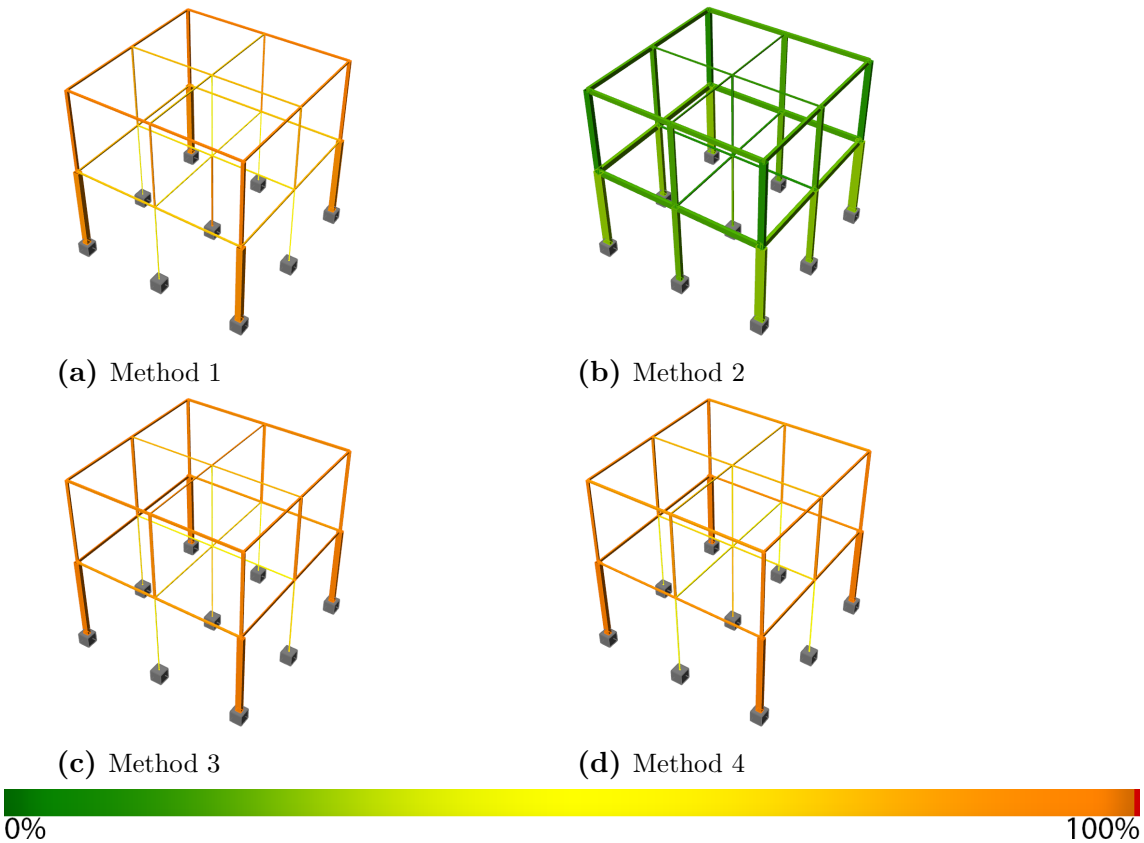
(d) Modes 1 - 3

Figure 6.11: Resulting structures from the mode section sizer, using different modes as input

When analysing the different modes as seen in table 6.7 it can be seen that modes 1 - 3 have almost equally low eigenvalues. When looking at these modes in figure

6.10 the first two modes can be identified as actually being the same mode, only acting in different directions. It is a 'shearing mode' which is corresponding fairly well to the loads applied in this case. The third mode is a twisting mode that is not found in the load combinations, but given the structures inherent weakness to it, it is included in the combined section sizer check anyway.

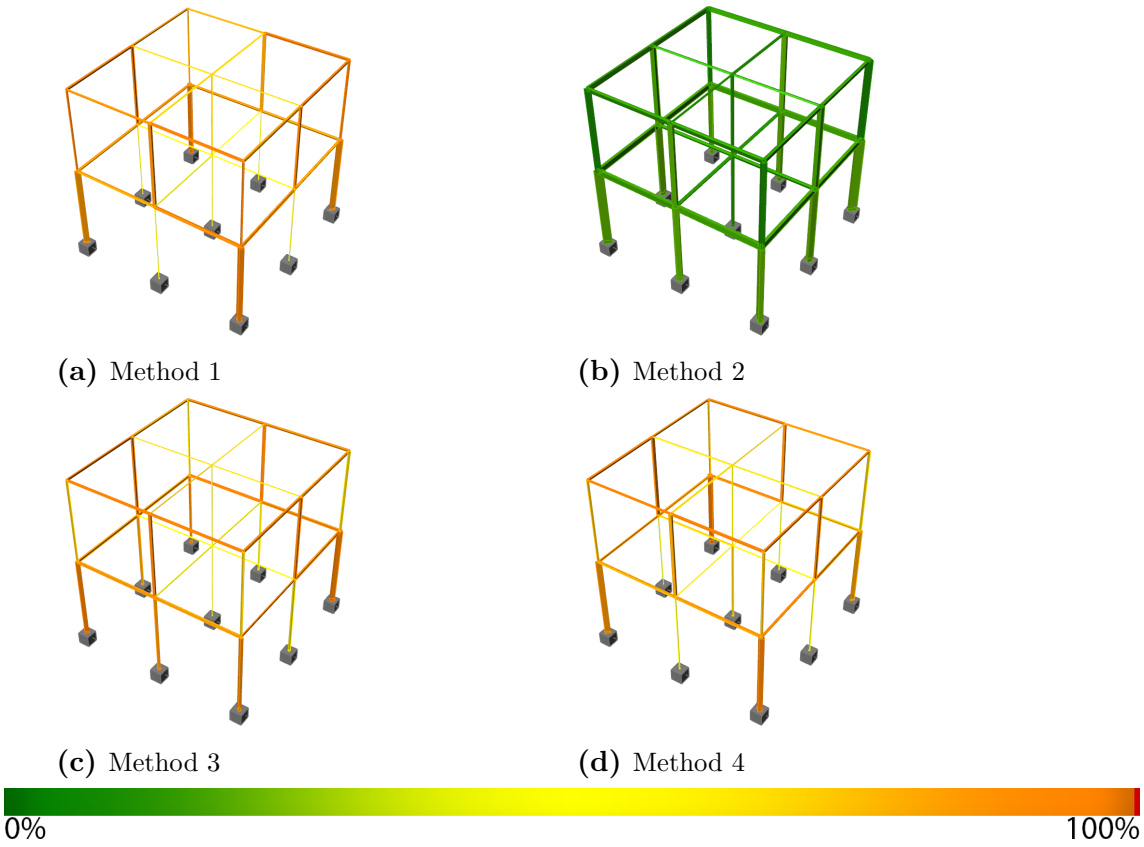
6.2.5 Combined section sizer



(a) Method 1 **(b)** Method 2

(c) Method 3 **(d)** Method 4

Figure 6.12: Resulting structure from the combined iterative section sizer - not allowing rotations.



(a) Method 1 **(b)** Method 2

(c) Method 3 **(d)** Method 4

Figure 6.13: Resulting structure from the combined iterative section sizer - allowing rotations.

Optimiser	Type	Weight [kg]	Calc. time	Iter.	Utilisation [%]		
					Min	Max	Avg
Combined sizer (no rotation)	1	**1673**	3.8 min	23	49.8	99.1	80.1
	2	3781	5 s	1	10.6	27.0	18.4
	3	1680	3.9 min	52	57.8	99.6	85.4
	4	1703	2.6 min	14	46.4	98.4	82.4
Combined sizer (incl. rotation)	1	**1658**	7.0 min	44	51.0	98.7	79.6
	2	4061	**1.3 min**	13	10.1	21.5	15.7
	3	1736	5.5 min	43	52.7	98.3	79.3
	4	1710	5.3 min	24	46.3	96.2	77.6

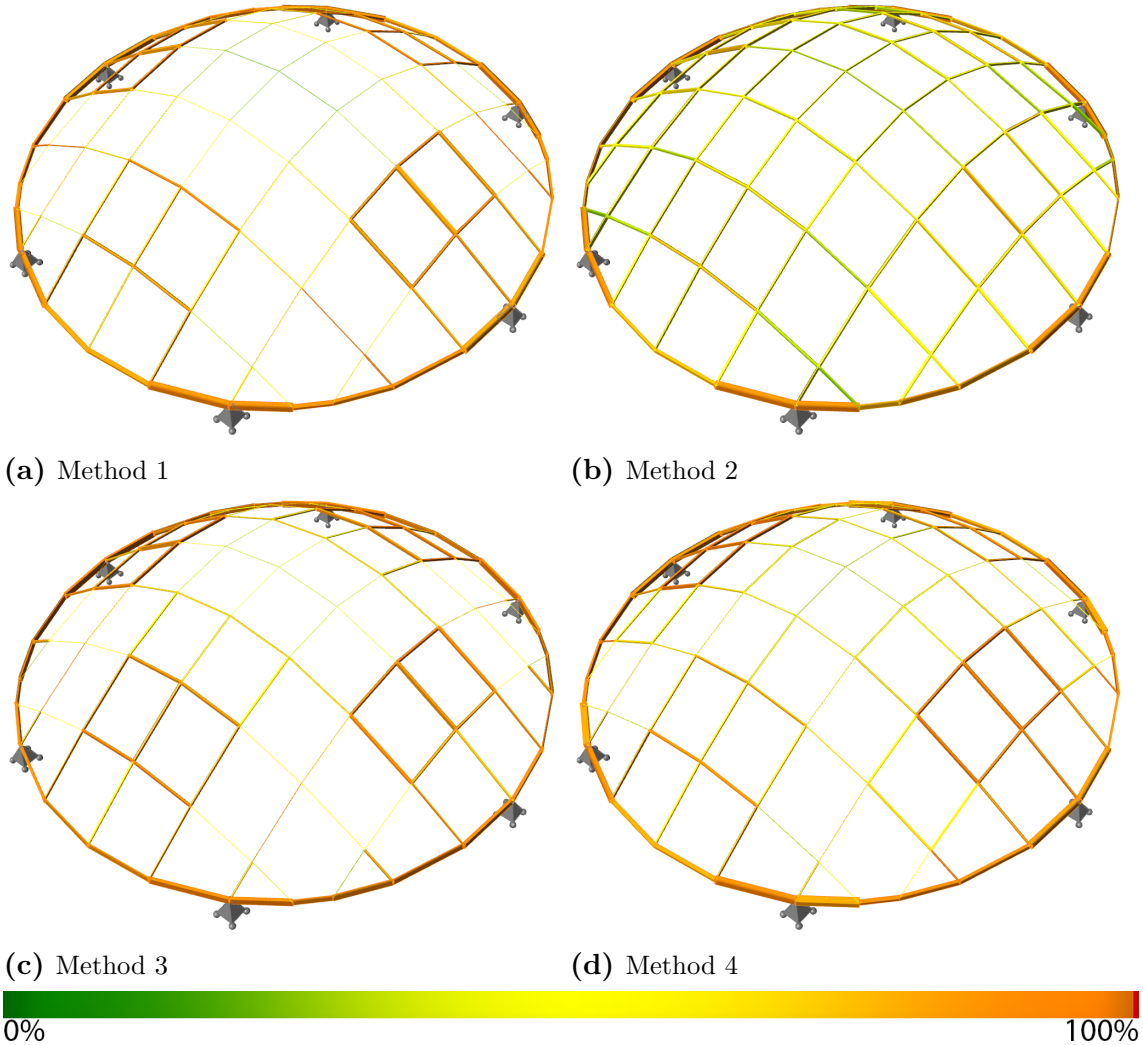
Table 6.8

6.2.6 Comments

The results for the 3d frame were quite similar to the results obtained from the 2d structure. It seems that increasing the complexity of the model increases the computation time for the genetic algorithms severely. The reason for this is probably that both the solution space and the computation time for each iteration increases, which together makes the search very slow. The most remarkable result for this structure is however that the combined section sizer (figure 6.12a) produced a better result than any other (not allowing rotations).

6.3 Dome

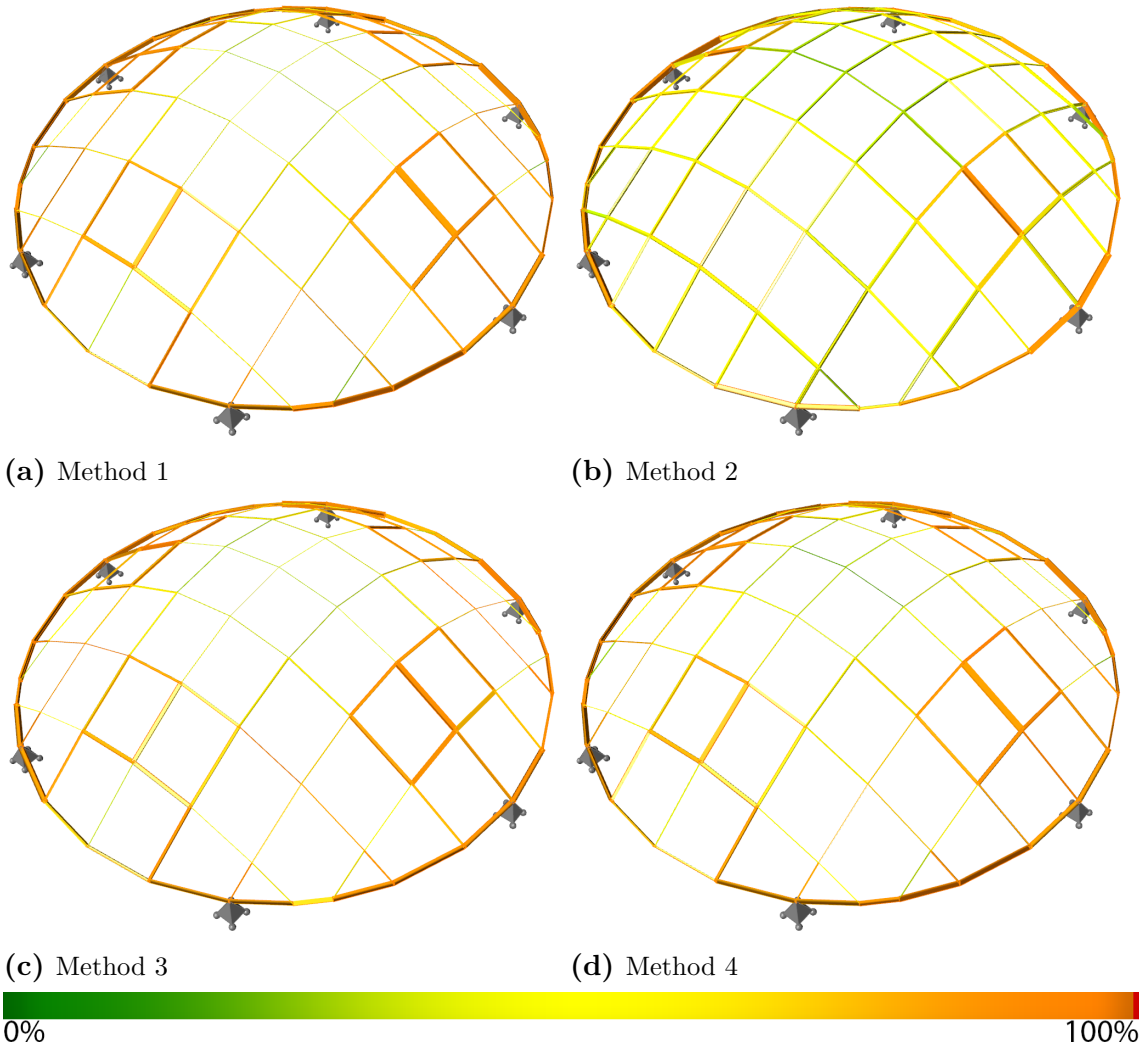
6.3.1 Section sizer and rotator



(a) Method 1 **(b)** Method 2

(c) Method 3 **(d)** Method 4

Figure 6.14: Resulting structure from iterative section sizer - not allowing rotations. Colour corresponding to utilisation from worst load case for each element



(a) Method 1 **(b)** Method 2

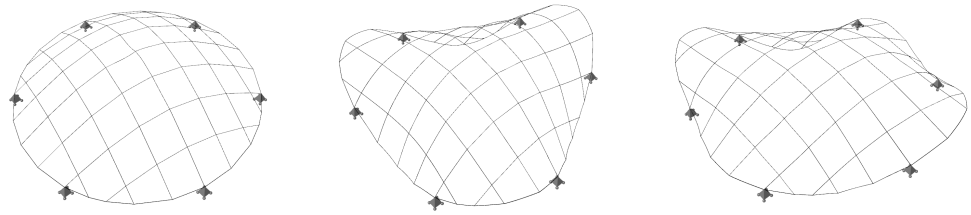
(c) Method 3 **(d)** Method 4

Figure 6.15: Resulting structure from iterative section sizer - allowing rotations. Colour corresponding to utilisation from worst load case for each element

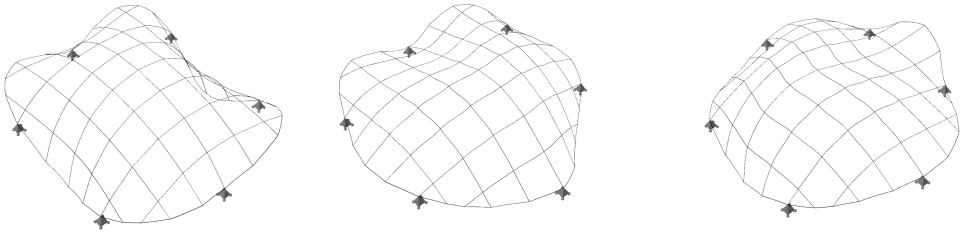
Optimiser	Type	Weight [kg]	Calc. time	Iter.	Utilisation [%]		
					Min	Max	Avg
Section sizer (no rotation)	1	**10060**	**1.74 min**	10	29.0	99.5	76.7
	2	13743	2.86 min	44	25.8	94.1	58.4
	3	10268	4.185 min	52	29.2	97.2	74.9
	4	10877	2.18 min	12	36.9	98.8	75.5
Section sizer (incl. rotation)	1	**9467**	**7.20 min**	38	23.7	99.8	73.9
	2	12787	8.45 min	65	25.1	97.0	58.4
	3	9510	11.05 min	77	29.4	99.3	74.3
	4	9672	9.03 min	47	21.6	99.1	72.7

Table 6.9: Results from the section sizer with and without rotation

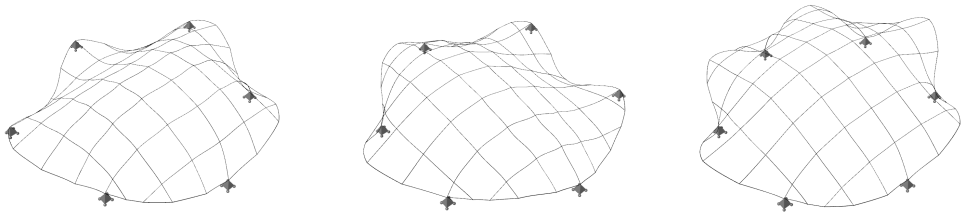
6.3.2 Canonical stiffnesses and eigenmodes



(a) Mode 1. $\lambda_1 = 3.42$ **(b)** Mode 2. $\lambda_2 = 60.9$ **(c)** Mode 3. $\lambda_3 = 84.8$



(d) Mode 4. $\lambda_4 = 94.7$ **(e)** Mode 5. $\lambda_5 = 123$ **(f)** Mode 6. $\lambda_6 = 127$



(g) Mode 7. $\lambda_7 = 192$ **(h)** Mode 8. $\lambda_8 = 214$ **(i)** Mode 9. $\lambda_9 = 234$

Figure 6.16: The 9 first eigenmodes for the dome with uniform sections

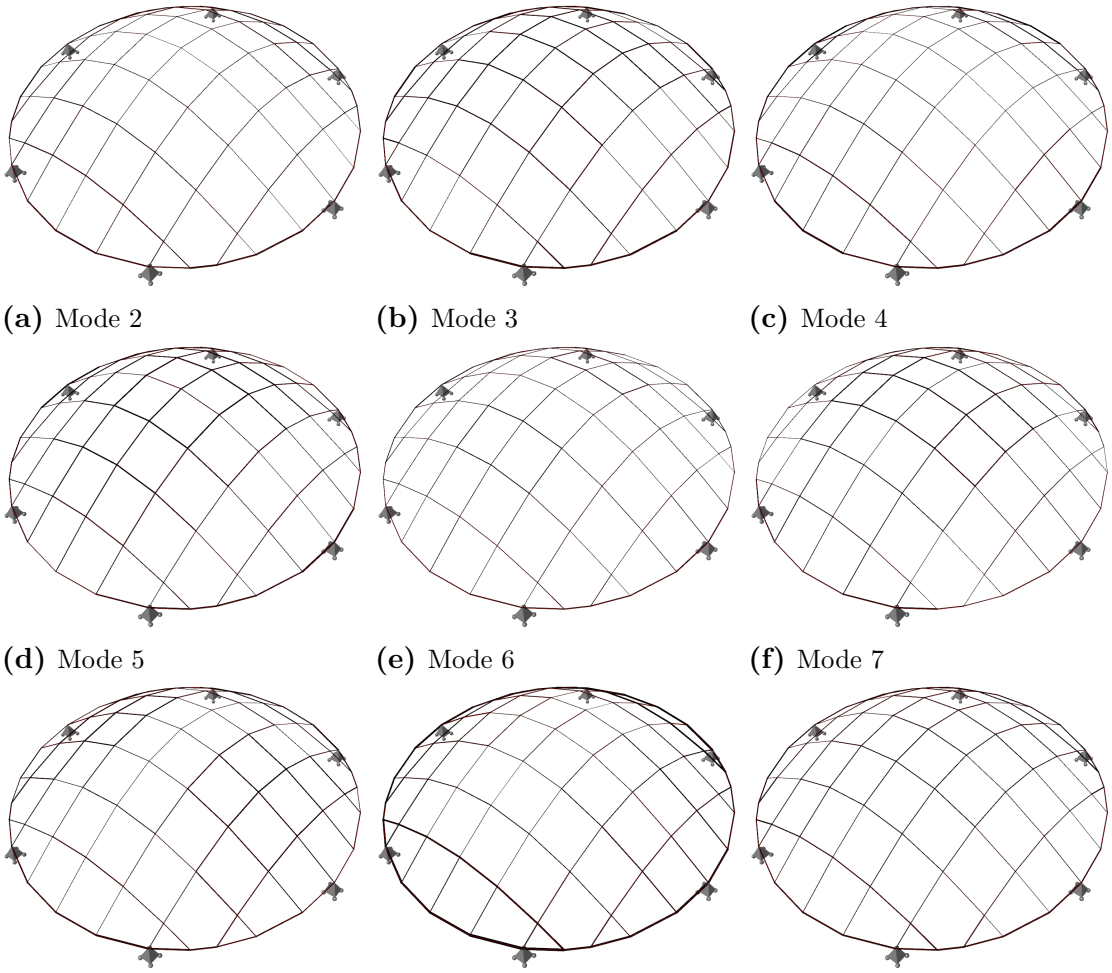
Mode	Eigenvalue	Normalised eigenvalue
1	3.42	0.0562
2	60.9	1.0
3	84.8	1.39
4	94.7	1.55
5	123	2.02
6	127	2.09
7	192	3.15
8	214	3.51
9	234	3.84
10	252	4.13

Table 6.10: Eigenvalues for the first modes. Normalized to the second eigenvalue

Looking at the mode shapes and eigenvalues one can note that the first is of little interest, as it reassembles something very close to a rigid body mode. This mode

do not give much information on the stiffness and behaviour of the structure, but is a consequence of the low degree of restrain from the chosen boundary conditions. This is why the eigenvalues have been normalised to the second eigenvalue, and why the mode shape optimiser and the combined section sizer in the upcoming sections were set to only work with higher modes.

6.3.3 Mode section sizer



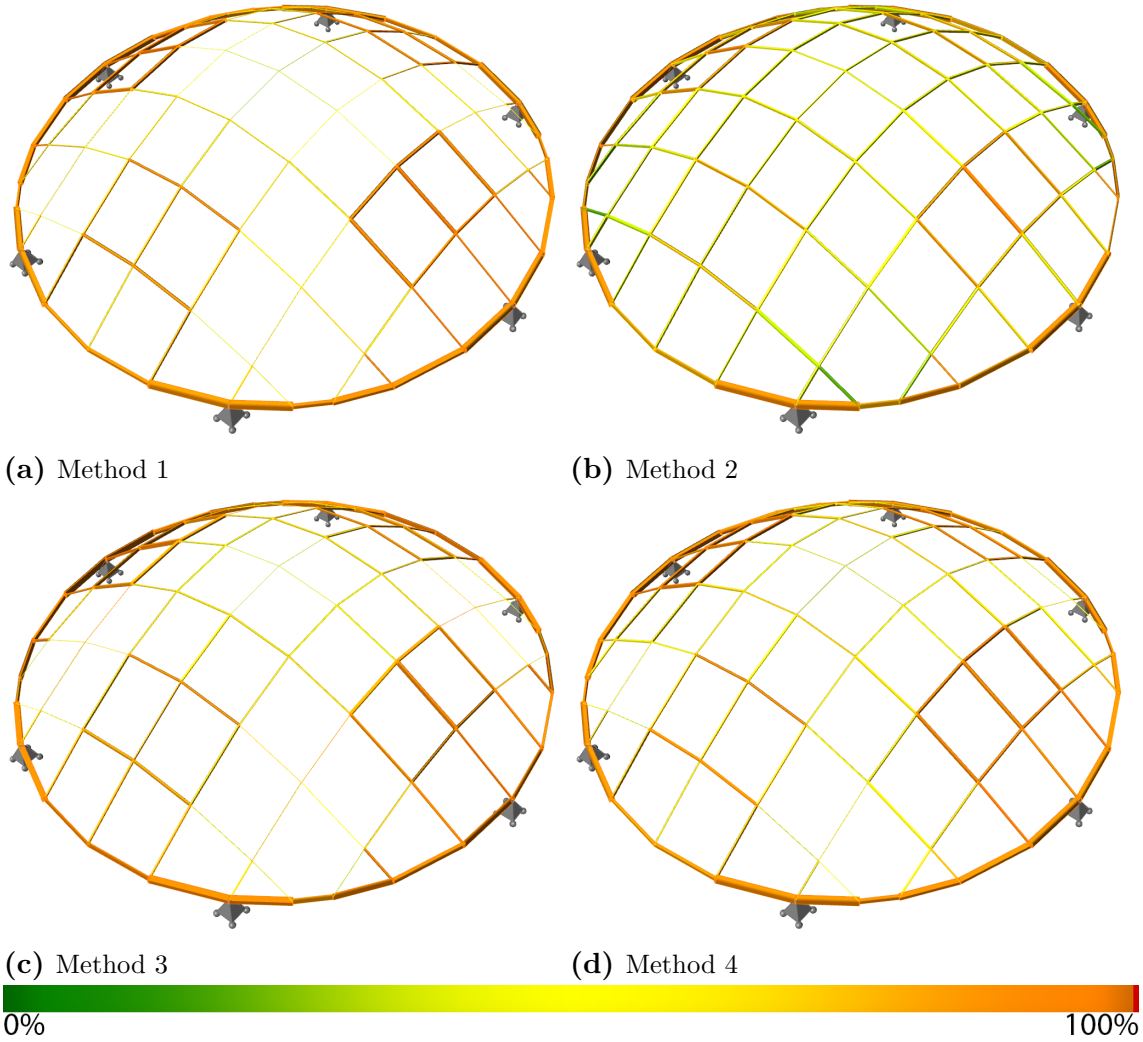
(a) Mode 2 **(b)** Mode 3 **(c)** Mode 4

(d) Mode 5 **(e)** Mode 6 **(f)** Mode 7

(g) Mode 8 **(h)** Mode 9 **(i)** Mode 2-10

Figure 6.17: Mode section sizer. The first eight figures are showing results from mode 2-9 run in solitude. The last figure shows modes 2-10 run together.

6.3.4 Combined section sizer

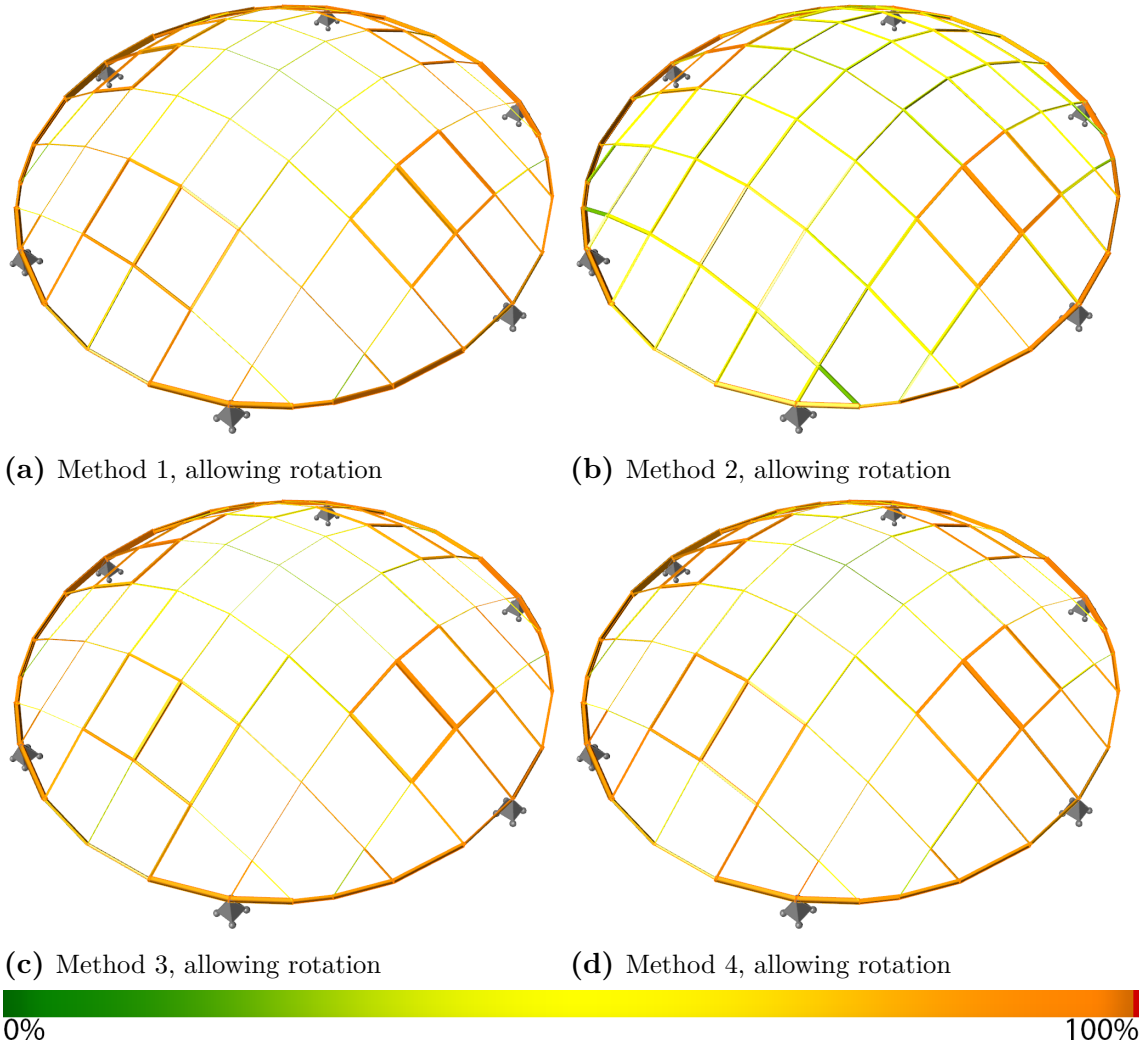


(a) Method 1 **(b)** Method 2

(c) Method 3 **(d)** Method 4

Figure 6.18: Resulting structure from combined section sizer. Modes used for mode shape optimiser are mode 2-10 (see fig. 6.28i), that can be seen in figure 6.17

The combined section sizer was set to work with modes 2-10. This means that the initial geometry for the second step in the process (the iterative section sizer) was the one from fig. 6.28i.



(a) Method 1, allowing rotation

(b) Method 2, allowing rotation

(c) Method 3, allowing rotation

(d) Method 4, allowing rotation

Figure 6.19: Resulting structure from combined section sizer. Modes used for mode shape optimiser are mode 2-10, that can be seen in figure 6.17

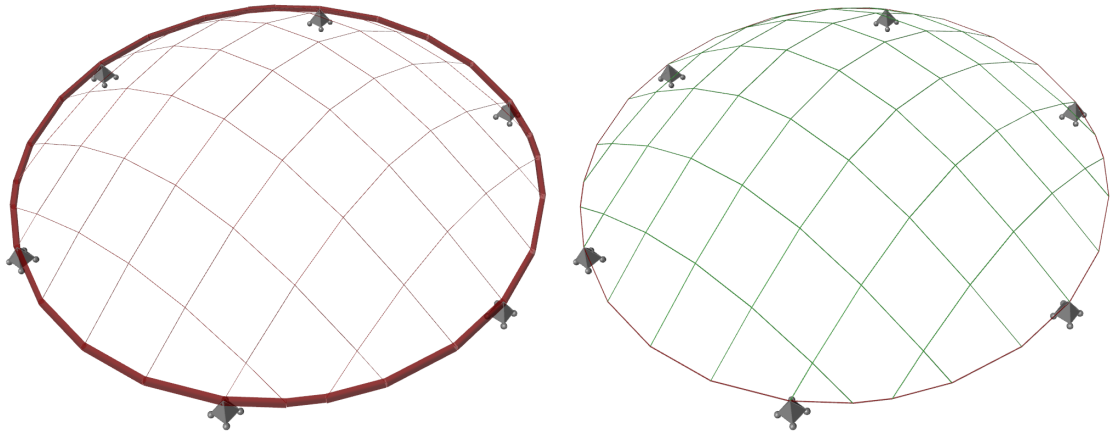
Optimiser	Type	Weight [kg]	Calc. time	Iter.	Utilisation [%]		
					Min	Max	Avg
Combined sizer (no rotation)	1	10544	**2.06 min**	9	29.4	98.4	77.6
	2	13317	2.78 min	37	20.1	93.1	60.5
	3	**10401**	3.83 min	42	33.5	98.1	76.9
	4	11267	2.20 min	11	36.9	99.4	76.5
Combined sizer (incl. rotation)	1	9353	8.01 min	29	25.9	98.5	76.3
	2	12054	11.7 min	59	22.6	98.9	62.7
	3	**9352**	7.90 min	54	27.6	98.5	74.4
	4	9573	**6.56 min**	33	25.9	99.1	75.1

Table 6.11: Combined section sizer set, setting initial conditions from mode 2-10

6.3.5 Section sizer with custom settings

The geometry of the structure reassembles one of a grid shell dome, which is a very efficient structure that can carry load more or less purely through compressive normal force. The set up boundary conditions, though, makes the structure carry the load predominately in bending. It is believed that an increase of stiffness of the outer ring surrounding the mid grid could enhance the dome like behaviour and thereby make a large part of the structure more efficient in that the way the load is handled.

To try to enhance this behaviour two different approaches were tested. The first was to increase the initial stiffness of the perimeter ring by starting from a significantly larger cross section (see fig. 6.20a). The second was to use different section chooser methods for the ring and the grid (see fig. 6.20b). The hypothesis is that using method 1 or 4 on the ring and method 3 on the grid will enable the ring to grow a lot "quicker" in terms of numbers of iteration, and thereby attract more force.

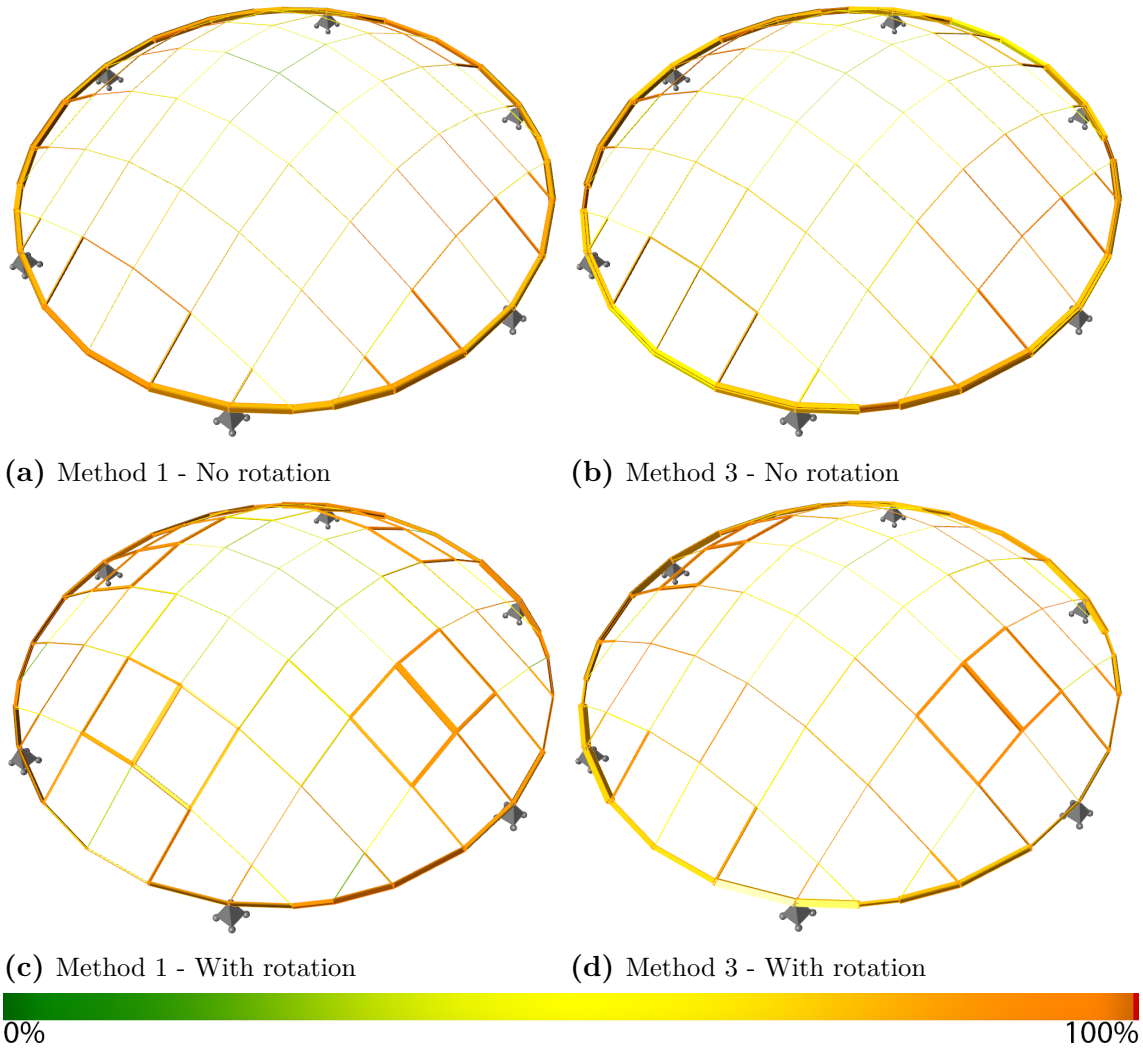


(a) Initial set up with larger sections on the perimeter ring

(b) Combination of methods. Green - method 3, Red - method 1 or 4

Figure 6.20: Figures showing the different settings

6.3.5.1 Change of initial condintions



(a) Method 1 - No rotation

(b) Method 3 - No rotation

(c) Method 1 - With rotation

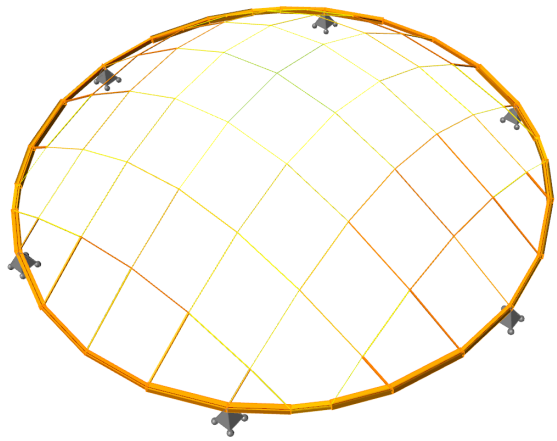
(d) Method 3 - With rotation

Figure 6.21: Section sizer ran with altered initial conditions.
Initial section 400x300x6 on perimeter ring, 30x30x6 on grid

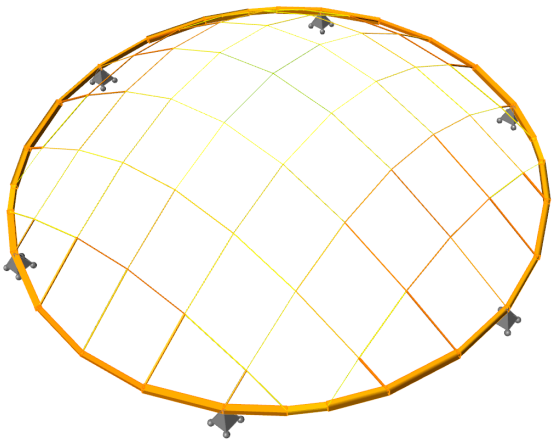
Optimiser	Type	Weight [kg]	Calc. time	Iter.	Utilisation [%]		
					Min	Max	Avg
Section sizer	1	9358	2.64 min	16	35.2	97.8	76.0
(no rotation)	3	9427	1.35 min	17	33.2	96.7	68.6
Section sizer	1	9467	10.4 min	38	23.7	99.8	73.9
(incl. rotation)	3	8904	7.75 min	56	31.9	99.9	73.2

Table 6.12: Section sizer run with different initial conditions

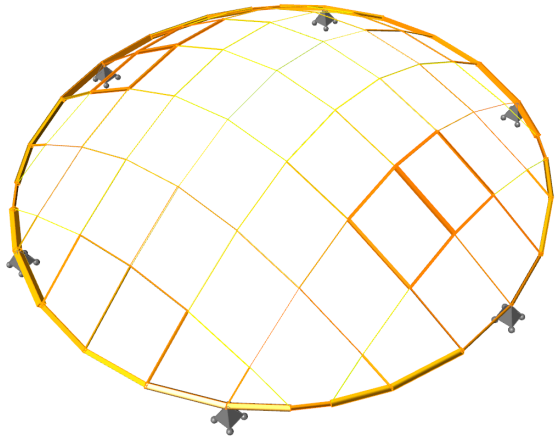
6.3.5.2 Combination of methods



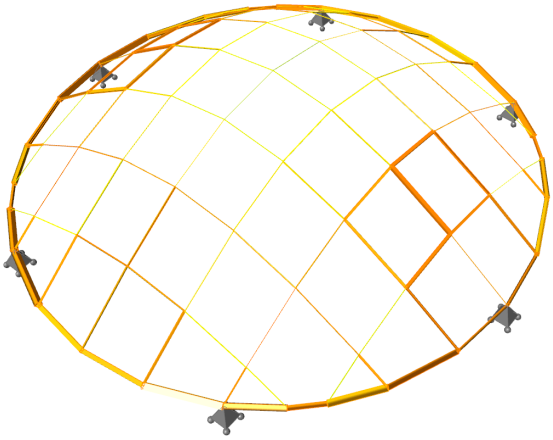
(a) Method 1 on perimeter ring, method 3 on grid - No rotation



(b) Method 4 on perimeter ring, method 3 on grid - No rotation



(c) Method 1 on perimeter ring, method 3 on grid - With rotation



(d) Method 4 on perimeter ring, method 3 on grid - With rotation

Figure 6.22: Section sizer ran with combination of methods on different parts of the structure

Optimiser	Type	Weight [kg]	Calc. time	Iter.	Utilisation [%]		
					Min	Max	Avg
Section sizer	1 & 3	9274	1.31 min	15	28.0	99.9	73.9
(no rotation)	4 & 3	9274	1.30 min	15	28.0	99.9	73.9
Section sizer	1& 3	9028	7.34 min	53	31.4	99.8	75.3
(incl. rotation)	4 & 3	8965	8.31 min	53	30.8	99.7	74.9

Table 6.13: Section sizer ran with combination of methods

6.3.6 Comments

It can be noted that when the section sizer was run with all initial sections set to the minimal, the best results in terms of weight were obtained using method 1. This is not very surprising as method 1 is the only method that with 100% certainty ensures that each element will get the smallest possible section in terms of area able to handle the load for a given set of section forces. It can also be noted that the orientation of the sections is important and that it was possible to reduce the weight by 5-10% if the cross sections were rotated to match their worst moments.

Since none of the mode shapes corresponded perfectly to the applied loads they did not give any real improvements when used in the combined section sizer. This can be seen by comparing the results of table 6.9 (Section sizer) with the results from table 6.11 (Combined section sizer), where the results in terms of weight are better in the first table.

The best results by far were achieved when some presets were made and a specific way of carrying the load was enforced. Looking at the results from section 6.3.5 one can note that the hypothesis that one would be able to reduce the overall weight of the structure by enforcing a dome-like behaviour, either by changing initial conditions or by using a combination of methods, was correct. The best results when not allowing rotation was achieved using method 3 on the grid and method 1 or 4 on the outer ring. The best result when allowing rotation was achieved when method 3 was used on the whole structure, but with a significantly larger cross section on the outer ring. This shows that one can get very good results with the section sizer if some evaluation on how one believes that the structure should work is done beforehand, and that that behaviour is enforced, either by choice of method or by a choice of initial conditions. This suggests that the tool is best used by someone with engineering experience, who can accurately interpret the results, and predict the behaviour.

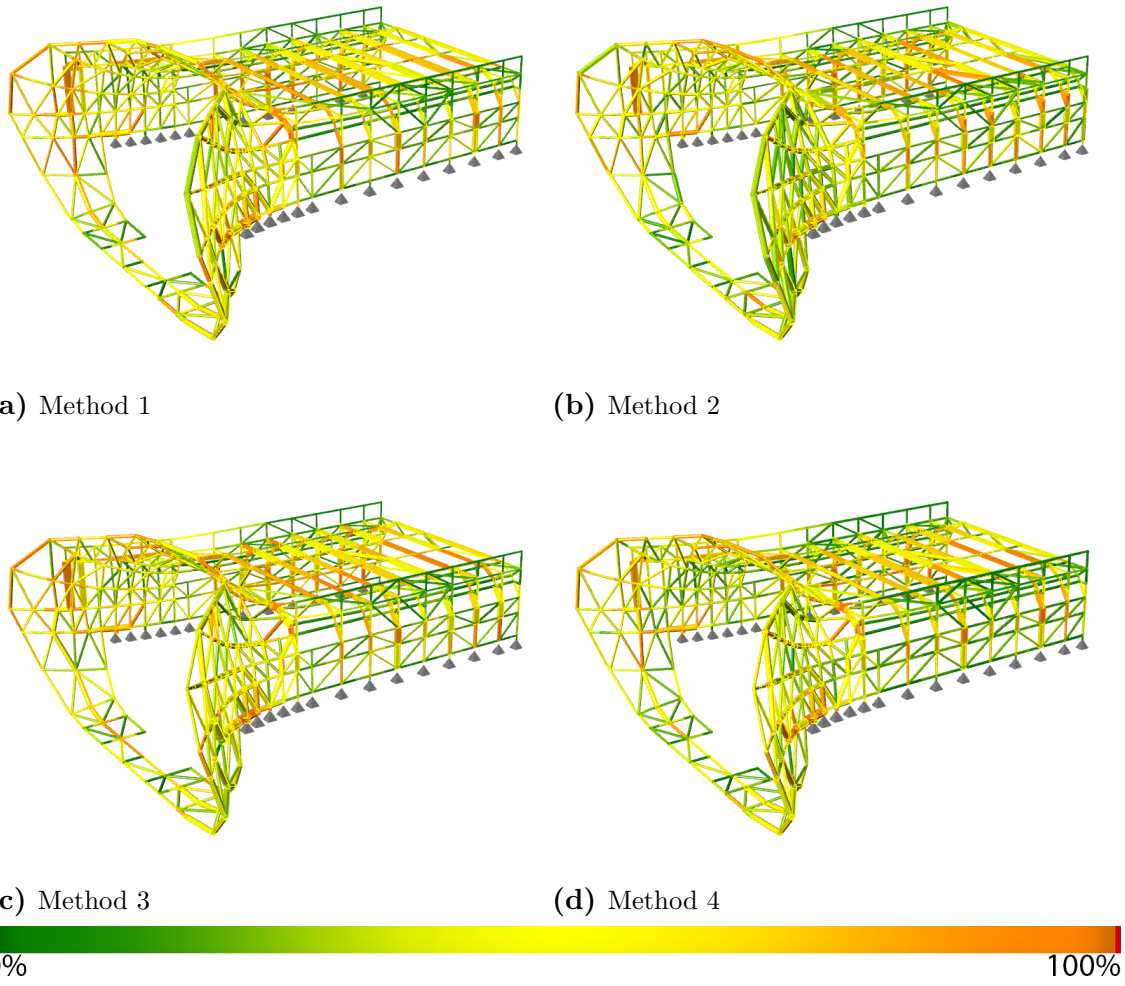
One can get a grasp on how efficient a certain setup of cross sections is in relation to another by comparing the weight and average utilisation of the two. For example, one can compare the results from the section sizer (without any initial presets) using method 1, $m = 10060kg$ and $\eta_m = 76.7\%$ (see table 6.11), with the results from the sections sizer used with a combination of methods, $m = 9274kg$ and $\eta_m = 73.9\%$ (see table 6.13). Here one can note that the latter method has both lower weight and lower average utilisation, which means that the load is handled in a better way globally, as one will get lower stresses in smaller sections using the second approach.

6.4 KAFD Metro Station

6.4.1 Section sizer and rotator

The section sizer and rotator was run with two different initial settings for the cross sections. The first run was made using the smallest cross section for each element that shared a type with it (CHS/RHS). The second run was made with initial cross sections set to the existing ones when the model was received. It should be noted here, as pointed out earlier, that the loads used for the optimisation were scaled to make the original design fail in terms of utilisation. This was done to give the optimisers 'more to work with', as the current utilisation checks only cover basic ULS checks and do not cover checks like deflection or buckling.

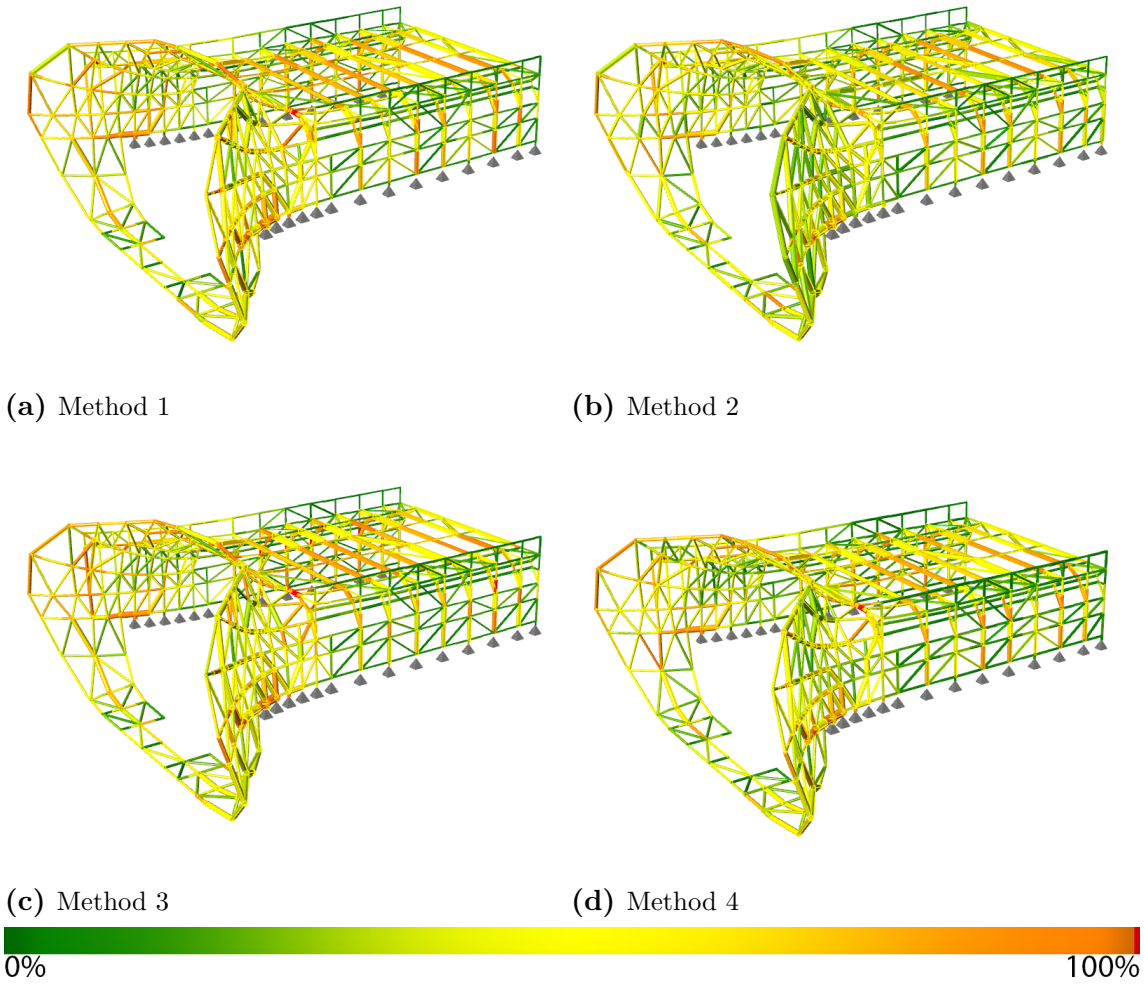
6.4.1.1 Minimal initial sections



(a) Method 1 **(b)** Method 2

(c) Method 3 **(d)** Method 4

Figure 6.23: Resulting structure from iterative section sizer - not allowing rotations. Colour corresponding to utilisation.



(a) Method 1

(b) Method 2

(c) Method 3

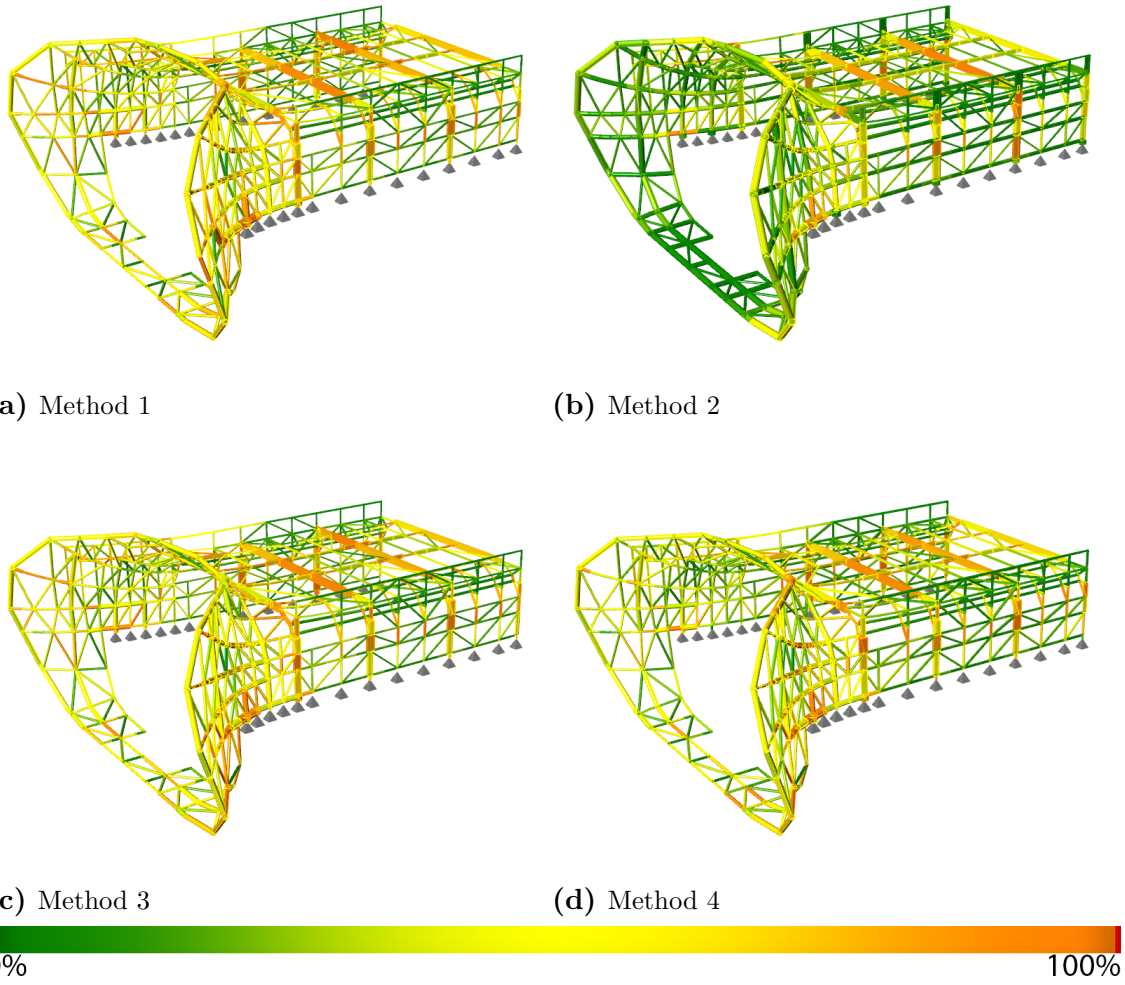
(d) Method 4

Figure 6.24: Resulting structure from iterative section sizer allowing rotations. Colour corresponding to utilisation.

Optimiser	Type	Weight [kg]	Calc. time	Iter.	Utilisation [%]		
					Min	Max	Avg
Section sizer (no rotation)	1	39520	1.97 min	13	0.27	96.8	41.4
	2	49430	**1.06 min**	11	0.27	97.2	37.6
	3	**39440**	2.52 min	22	0.27	99.5	41.6
	4	46960	1.89 min	13	0.29	99.9	36.6
Section sizer (incl. rotation)	1	**38690**	10.0 min	37	0.27	111.0	41.4
	2	48620	**7.41 min**	32	0.27	98.9	36.0
	3	38960	32.2 min	150	0.27	145.0	41.3
	4	46810	8.85 min	33	0.29	122.0	36.6

Table 6.14: Results from the section sizer, ran with initial section sat to minimum sections

6.4.1.2 Original initial sections



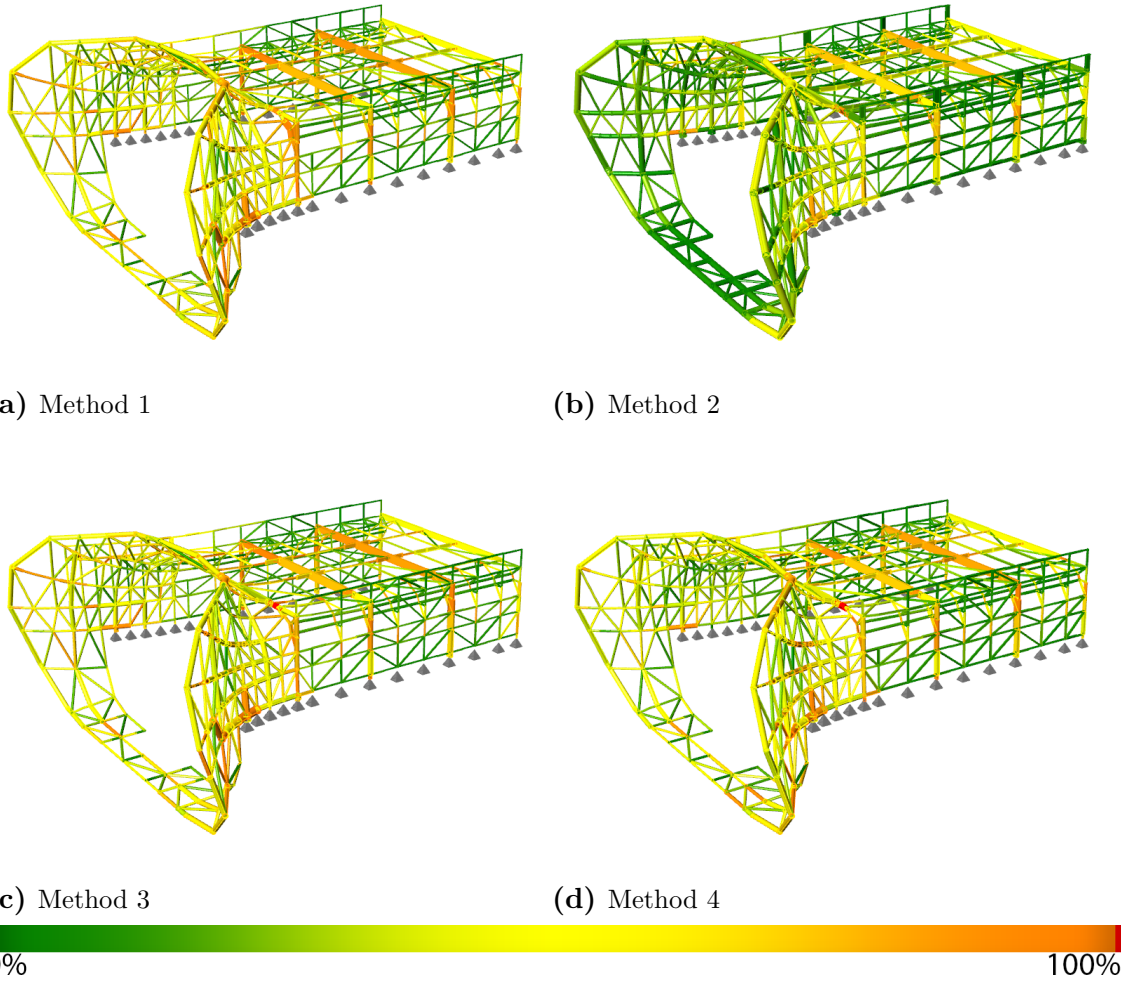
(a) Method 1

(b) Method 2

(c) Method 3

(d) Method 4

Figure 6.25: Resulting structure from iterative section sizer - not allowing rotations. Colour corresponding to utilisation.



(a) Method 1 **(b)** Method 2

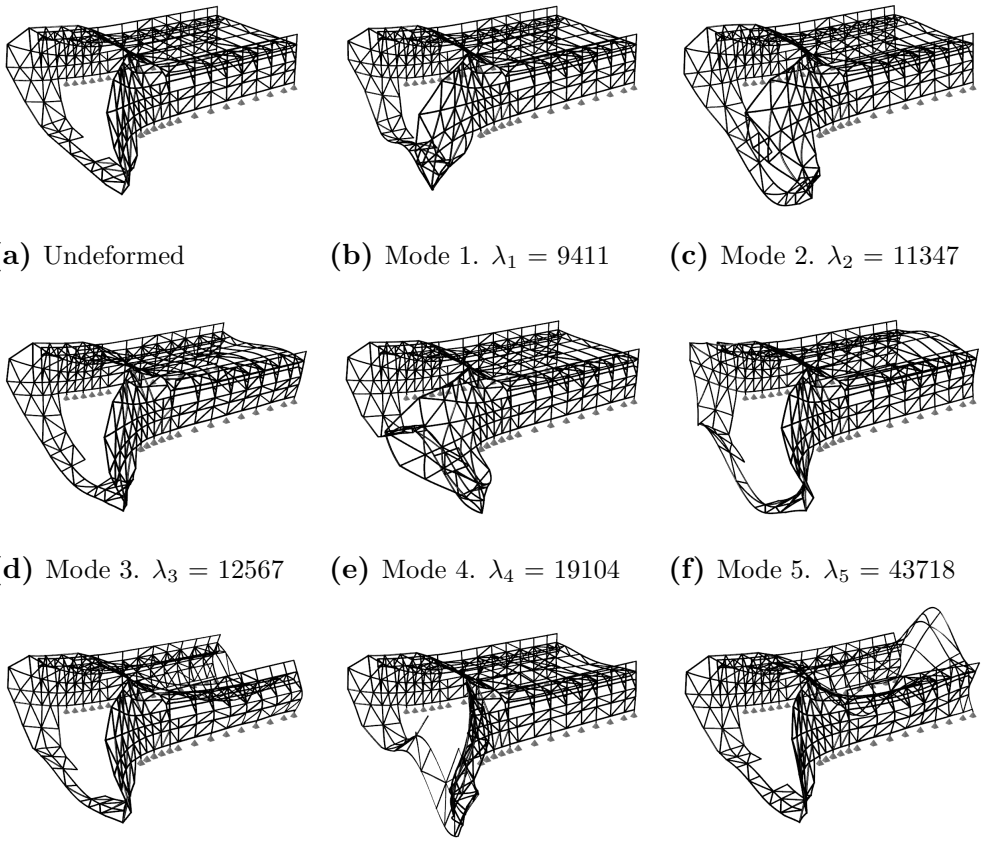
(c) Method 3 **(d)** Method 4

Figure 6.26: Resulting structure from iterative section sizer allowing rotations. Colour corresponding to utilisation.

Optimiser	Type	Weight [kg]	Calc. time	Iter.	Utilisation [%]		
					Min	Max	Avg
Section sizer (no rotation)	1	**39390**	2.81 min	17	0.27	98.6	40.7
	2	70140	**33.3 s**	5	0.2	94.7	25.3
	3	40650	1.15 min	11	0.27	99.3	40.9
	4	48240	1.33 min	8	0.29	99.8	36.4
Section sizer (incl. rotation)	1	**39210**	40.6 min	150	0.27	99.8	39.2
	2	69840	**7.36 min**	34	0.2	97.4	23.1
	3	40010	34.3 min	150	0.27	104.0	38.9
	4	48110	41.5 min	150	0.29	104.0	33.9

Table 6.15: Results from the section sizer, ran with initial section set to original sections.

6.4.2 Canonical stiffnesses and eigenmodes



(a) Undeformed **(b)** Mode 1. $\lambda_1 = 9411$ **(c)** Mode 2. $\lambda_2 = 11347$

(d) Mode 3. $\lambda_3 = 12567$ **(e)** Mode 4. $\lambda_4 = 19104$ **(f)** Mode 5. $\lambda_5 = 43718$

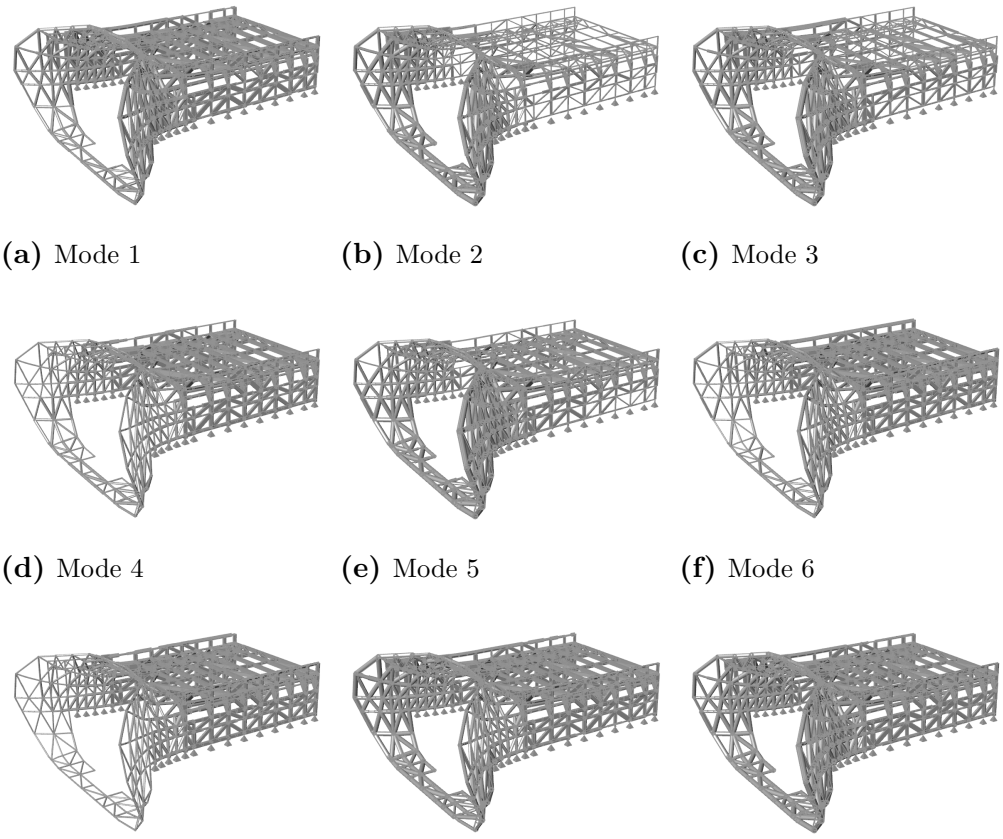
(g) Mode 6. $\lambda_6 = 44938$ **(h)** Mode 7. $\lambda_7 = 57626$ **(i)** Mode 8. $\lambda_8 = 69074$

Figure 6.27: The 8 first eigenmodes. The analysis was run with as uniform sections as possible. This means RHS100x100x10 for the rectangular sections and CHS139.7x5 for the circular ones.

Mode	Eigenvalue	Normalised eigenvalue
1	9411	1.0
2	11347	1.21
3	12567	1.34
4	19104	2.03
5	43718	4.65
6	44938	4.77
7	57626	6.12
8	69074	7.34
9	87129	9.26
10	89519	9.51
11	97326	10.3

Table 6.16: Eigenvalues for the first modes. Normalized to the first eigenvalue.

6.4.3 Mode section sizer



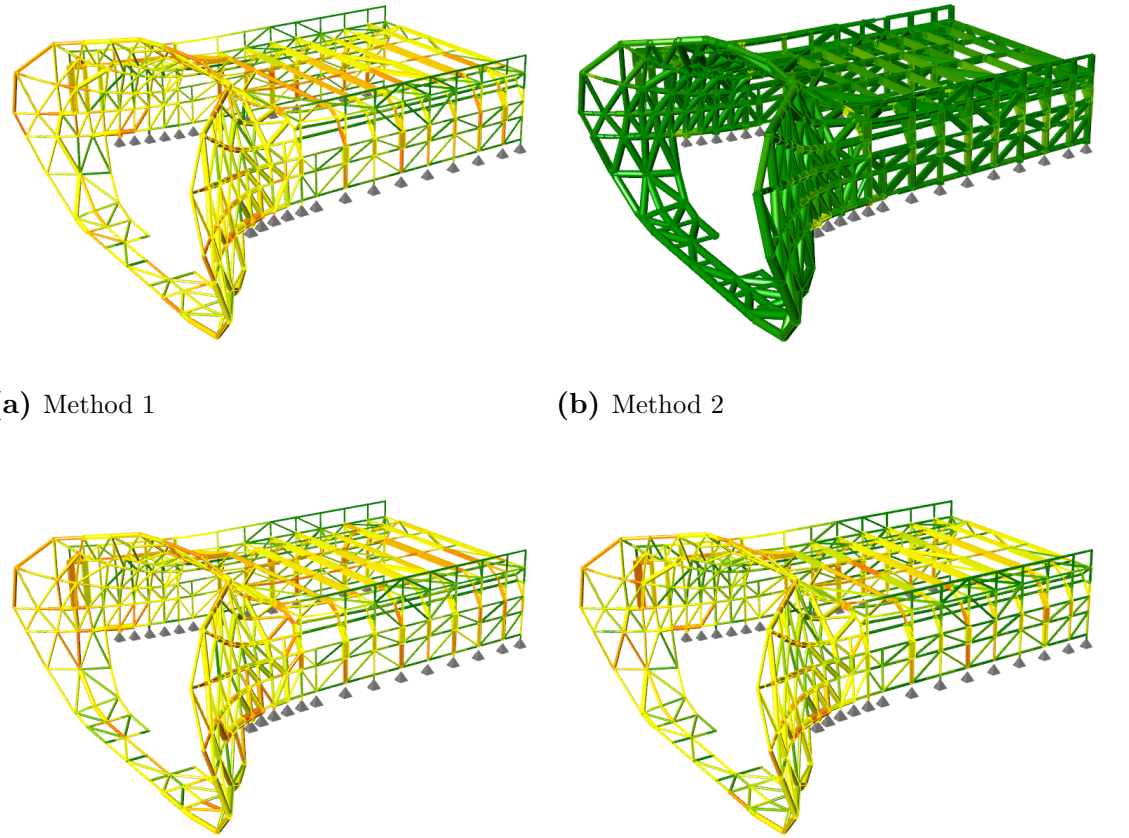
(a) Mode 1 **(b)** Mode 2 **(c)** Mode 3

(d) Mode 4 **(e)** Mode 5 **(f)** Mode 6

(g) Mode 7 **(h)** Mode 8 **(i)** Mode 1-8

Figure 6.28: Mode section sizer. The first eight figures are showing results from mode 2-9 ran in solitude. The last shows modes 2-10 run together.

6.4.4 Combined section sizer



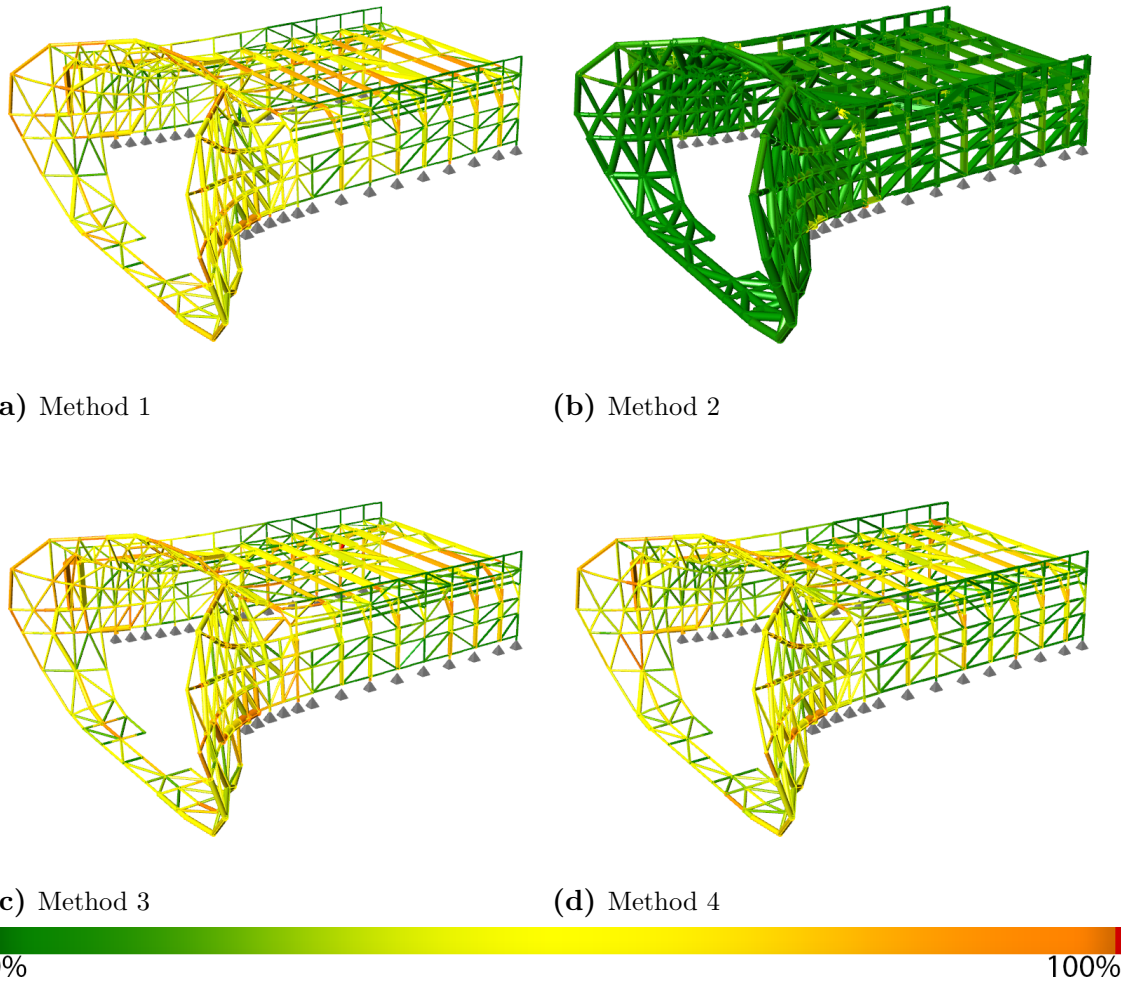
(a) Method 1

(b) Method 2

(c) Method 3

(d) Method 4

Figure 6.29: Resulting structure from iterative section sizer - not allowing rotations. Colour corresponding to utilisation.



(a) Method 1 **(b)** Method 2

(c) Method 3 **(d)** Method 4

Figure 6.30: Resulting structure from iterative section sizer allowing rotations. Colour corresponding to utilisation.

Optimiser	Type	Weight [kg]	Calc. time	Iter.	Utilisation [%]		
					Min	Max	Avg
Combined sizer (no rotation)	1	40920	6.4 min	13	0.27	99.9	40.6
	2	282500	**4.42 min**	1	0.22	92.6	7.12
	3	**39540**	6.48 min	17	0.27	99.6	41.6
	4	49700	6.31 min	10	0.29	97.5	34.7
Combined sizer (incl. rotation)	1	**40610**	14.2 min	39	0.27	98.3	39.7
	2	282500	13.4 min	44	0.15	84.6	7.01
	3	39210	38.6 min	150	0.27	118.0	41.0
	4	49440	**12.2 min**	25	0.29	98.5	33.9

Table 6.17: Combined section sizer set, setting initial conditions from mode 2-10

6.4.5 Comments

One thing one can note looking at the results from the section sizer is that method 1 and 3 gives the best results in terms of weight for both of the two different initial cross section setups. It is quite surprising is that method 4 gives significantly higher weight, even higher than the ones achieved using method 2 for some cases. This result has not been observed in the other case studies. The reason for this behaviour might be the small set of cross sections used in the optimisation process, which could lead to a smaller possibility for method 4 to reduce the cross section, if a situation like the one described in section 3.3.1.6 appears, where different aspect ratios of the cross sections might "block" the possibility to find a working smaller one.
The small set of cross sections could probably be the reason why the section sizer have convergence problems when rotation is allowed. Increasing the list of allowable cross sections would allow for smaller "jumps" in terms of stiffness and strength, and could also give the possibility for smaller and or larger cross sections for under respectively over utilised elements.
One can note that the initial conditions have a large impact on the final results. Looking at the results from the section sizer when run with the initial sections set to the original ones (see fig. 6.25) one can note a clear similarity to the original setup (see fig 5.9. Here the force and stiffness have been concentrated over a smaller set of portal frames, compared to to solutions obtained when the section sizer was run with all initial sections set to the minimum one (see fig. 6.23).

The mode shape optimiser, when run for mode 1-8, gave very large sections for all of the members. This is probably also a consequence of the small set of cross sections used. This makes the influence of the mode section sizer smaller as more or less all of the element will start with the largest possible cross section. Looking at the results from the combined section sizer one can note that this leads to method 2 giving poor results, as it only allows for an increase in sections size (see fig. 6.29 and fig. 6.30). This can also be noted from the results in table 6.17 where method 2 had a seven times larger weight compared to the other methods and converged after only one iteration.

7
Discussion

7.1 CIGull

Evaluating the creation of *CIGull* has rendered an overall positive response. The comparative study to CALFEM (see Appendix A) showed that the calculations produced coherent results with very small discrepancies. The results when using the iterative procedures (see chapter 6) shows low calculation times for the given tasks. For the 2d frame the calculation is sufficiently quick so that it could be used continuously while changing section properties, connection stiffnesses or node positions. The current calculations are also set up in a way so that solving the system of equations, extracting element forces and performing section checks are all performed in the same process. In order to increase speed these actions could be broken up and implemented in separate components, giving the user the choice to only use the ones that they currently need.

The optimisation methods studied in this thesis work and implemented in *CIGull* have all shown different strengths and weaknesses, which are discussed in detail below. The assessment of the methods as a whole has shown that there are no clear "best method", but that the best solution is obtained by trying several methods to really understand the structure. Another reason for this is that the definition of what the best design is may not be decided solely from the weight, but also from other factors. These reasons both indicates that the tool is best used for 'structural sketching' in the early design stages.

7.2 Iterative Section Sizer

The different methods for choosing cross section used by the section sizer have been shown to give different results. Method 1, 3 and 4 have all given the best results in terms of minimising structural weight, depending on the set-up. The comments below are all based on the results from the conducted analyses in chapter 6.

7.2.1 Method 1

Method 1 works by choosing a new cross section by checking the provided cross sections from smallest to largest in terms of area until a cross section is found that can withstand the internal section forces. For a more detailed description see section 3.3.1.2. This method has produced the best results in terms of desired lowest

weight for most of the conducted studies. This is not overly surprising as this method always gives the smallest possible cross section to each element for every iteration, something that only can be guaranteed by this method.

7.2.2 Method 2

Method 2 works by updating the cross section to a cross section one step larger in terms of area, if the current cross section is unable to withstand the section forces. For a more detailed description see section 3.3.1.3. This method has been shown to be quick, but the resulting structures tend to have higher weight than the ones generated with the other methods. As a result of this, many of the elements tend to be underutilised. All elements sized by method 2 have at some point had a utilisation which is good (close to 100%). However, as the stiffness of other elements is increased, the utilisation drops, and without a way of downsizing, the elements are bound to be underutilised.

This method usually gives, based on the conducted studies, a more uniform solution in terms of cross section types. This might be beneficial if one seeks a more even distribution of either internal forces in the structure or reaction forces at the supports. Another positive effect of this could be a higher degree of structural redundancy.

7.2.3 Method 3

Method 3 works by updating the cross section to one which is one step larger if the cross section is overutilised, and checks if it is possible to update to one which is one step smaller if the cross section is underutilised. For a more detailed description see section 3.3.1.4. This method usually requires the highest number of iterations to find convergence. This can be explained by the fact that this method only allows for small changes in cross section area which means that it will take more iterations to obtain the "best" one if the stresses are very high/low compared to the capacity of the analysed cross section. This also means that method 3 is the overall slowest approach, even though each iteration usually is quicker than one iteration for method 1 and 4. For example, looking at the results for the 3D-frame in table 6.5 one can note that method 1 and 3 converges in roughly the same time, even though method 3 requires three times as many iterations.
Method 3 can, for some cases, give the best results in terms of low structural weight. This could originate from some structures benefiting from a slower change in cross sections, and thereby slower change in the stiffness and force patterns.

7.2.4 Method 4

Method 4 works by checking all provided cross sections larger than the current used until a cross section able to withstand the section forces is found, if the cross section is overutilised. If the cross section is underutilised it instead checks all smaller cross sections until the smallest working cross section is found. For a more detailed

description see section 3.3.1.5. This method has been shown to give the best results for some setups. It can be significantly more vulnerable to the cross section list used, something that can be seen in the evaluation of the KAFD structure (section 6.4) and is discussed in section 6.4.5. From the studied examples one can conclude that method 1 seem to be preferable to use for almost all cases if one seeks to use a method that allows for large changes in section size for each iteration.

7.2.5 Combining methods

In section 6.3.5 it was shown that a combination of the methods could give better results. It shows that with some insight to how both the methods and a structure works, it is possible to achieve more. This is something that could be developed further, by for example allowing some chosen sections to find their section properties before the other ones, for example the primary structure before the secondary structure.

7.2.6 Section rotator

The section rotator, when used together with the section sizer, has in almost all cases produced more efficient structures compared to the structures produced when only the section sizer has been run. This can be explained almost by definition, since the section rotator aligns the section to the direction of the largest principal moment. It should therefore, if used only once, always improve the response of elements unless they are already aligned correctly. When used in an iterative procedure, the behaviour is more unpredictable and may produce some less efficient solutions.

A practical issue concerning the section rotator is how the structural details will be affected. A node with many elements, which all are rotated independently, connected to it may be very tricky to manufacture. This issue can be handled in several ways:

- Use the section rotator to influence design, and make manual adjustments.
- Further develop the process, implementing some set rotations that are allowed (for instance only 90° or 45°).
- Develop/use a construction scheme that allows custom rotations

7.3 Genetic algorithms

The somewhat limited evaluation of genetic algorithms as a design tool for structural design, that has been conducted in this thesis work, has shown that it is possible to obtain good solutions using them. However, it is a time consuming process, which quickly grows into an impossible problem within the usual time constraints of a design project. The 3d frame structure in section 6.2 contained 42 elements, each of which could take any of 63 different cross sections. This means roughly $3.73 \cdot 10^{75}$ solutions (see eq. 2.91), a number in the vicinity of some current estimations of how many atoms there are in the universe. When the calculations were made each

calculation took about 2.5 seconds, resulting in a huge required time for the entire process to produce a result. In order to increase the viability of using this method, three possible solutions are identified:

- Starting from a qualified guess
- Increase the speed of calculation
- Decrease the number of solutions

For the studied example of the 3d frame (see section 6.2.2), the genetic algorithms were initialised with a guess state which was produced by one of the section sizer methods. This process did increase the efficiency of the method a great deal, and gave sligtly better results. The downside of this approach is that the genetic algorithm is steered toward a particular solution.

Some effort has been put into making the solver as efficient as possible, but some further measures can still be made. With the increasing speed of processing affordable high-performance hardware may also tend to this problem. The problem could also be simplified, something that should be done for any engineering problem. In the case of the 3d frame, the loads are doubly symmetric, which means that the problem may be reduced to containing only four load combinations instead of nine, cutting the computation time in about half.

Decreasing the number of solutions is really a matter of simplifying the problem. Suggestions for ways of doing this is reducing the number of section properties available. Simplifying the problem could also be done by for instance prescribing that two or more elements should have the same section property, which probably also simplifies construction.

Another problem with the implementations of the genetic algorithms that have been tested is that they require large amounts of RAM which may cause crashes. Another thing that may be noted when performing genetic algorithms is that the set-up of the project needs to very rigorous. This isn't necessarily a problem, but could actually force the engineer to structure their model in a way that allows them to have better overview.

7.4 Connection stiffness optimisation

In this thesis work, the connection stiffness optimisation has been a somewhat left out topic. In the initial example (see 1.1.1) connection stiffness seemed like an interesting way of redistribution the flow of forces from the overutilised to the underutilised elements. It was further investigated in the 2d frame example, by using genetic algorithms and shown to produce good results. However, as seen in the previous section on genetic algorithms in general, this method gets very time consuming as the size of the structure is increased. When the GA was tested on the 3d frame (see section 6.2) they were unable to find a good result for using section properties as

the only design variable. Therefore the method of optimising for connection stiffness was abandoned going forward.

The measures to increase the efficiency of the GAs also apply for connection stiffness optimisation. By for instance only using a few discrete stiffnesses instead of a range, the number of solutions is reduced. The set of stiffnesses could for instance be some actual connections that have been properly analysed. Another way of increasing the efficiency would be to not combine the connection stiffness and section property search, but rather perform a connection stiffness optimisation in isolation, optimising for reduced utilisations. When a good solution has been found the sections could get re-evaluated and updated if possible. This process involves more steps but could be more reliable.

In conclusion connection stiffness optimisation is still a promising topic, if a more efficient method of automating the process was to be developed.

7.5 Mode section sizer

The mode section sizer has in this implementation not been tested to its full capacity. The way that the method has been used previously (cf. [Olsson, Thelin (2003)], [Olsson (2006)]) have been using the eigenmodes to identify deformation patterns that may occur in a structure. This suggest that the mode section sizer could be more useful for a case where the loads aren't known, which requires a different study (see 7.10).

The mode section sizer also has some issues regarding its stability. Since it sizes the elements based on fictitious loads derived from the eigenmodes the elements may become very large in some cases. To handle this, a scale factor has been implemented in the mode section sizer, but not the combined section sizer.

The way that the mode section sizer is used could also be explored further. In the studied examples for in this thesis, the mode analysis have been made for one case each, often using the smallest sections available. However, the mode shapes are sorted by their eigenvalue and the mode shape optimiser uses the eigenvalue to scale the strains. Since the canonical stiffnesses can be interpreted as a measure of the sensitivity to a displacement, this may cause a problem. For instance, using small sections makes the structure relatively more sensitive to bending moments than axial forces. Therefore it is of importance what sections are used in the eigenmode calculation.

7.6 Combined section sizer

The combined section sizer has performed over expectation in the studied examples. For the 2d structure, it produced the best results apart from the genetic algorithms, but at a fraction of the time. It seems that by identifying deformation modes that correspond approximately to the applied load combinations and optimising for them, the stiffness was steered toward this behaviour. By optimising for the "worst version" of the actual load combination, the stiffness distribution seems to have become a good starting position for the coming iterations. However with that said, it was not always easy to identify the worst eigenmodes, as seen in the analysis of the dome and the KAFD Metro Station.

7.7 Package release

The intention is that the package will be made available as a downloadable plug-in for Grasshopper. The platform for this will most likely be the website *Food4Rhino* where most plug-ins for Rhino and Grasshopper are collected. The website address is:
<http://www.food4rhino.com/>

In order to release it, some further work needs to be done, such as error handling, bug searching, drawing of icons et cetera.

7.8 Open source code development

The entire code package is developed as open-source and available for anyone to download and develop under a Creative Commons Attribution-ShareAlike 4.0 International License. To view a copy of this license, visit:
<http://creativecommons.org/licenses/by-sa/4.0/>

The project is accessible from <https://github.com/IsakNaslund/MasterThesis>

Armadillo is available as open source under the restrictions of Mozilla Public Licence (MPL) 2.0. For further information, consult
<http://arma.sourceforge.net/>

7.9 Future implementation

This section describes various ideas and suggestions for future implementations and research that have arisen during this thesis work.

7.9.1 Structural Mechanics

Solvers

The currently implemented solvers can handle most standard situations. A further development could be to develop a solver for structural analysis according to the second order theory and/or a solver able to handle non-linear materials.

Element releases

The element releases are currently set to work in global coordinates. A change that could be an improvement for some situations would be to allow one to set these in local coordinates instead.

Restraint directions

In the restraint nodes, there is an optional input of the plane in which the restraint works. This is currently only implemented for the default XY plane. This means that the restraint directions are in global coordinates, which is acceptable for most cases. However, a future implementation of this function would make all restraints possible. For information on how this could be done, consult [Dahlblom, Olsson (2010), fig 5.18, p.138]

Distributed loads

Currently, loads are applied as point loads or point moments, with an additional option of adding a gravity load. However, no easy way of applying distributed loads is currently implemented. It can be approximated by exchanging the distributed load by a combination of point loads and moments, but this requires extra manual labour. Information on how to perform this can be found in [Austrell et al. (2004), p. 5.6 - 26]

Elements

The solver could be expanded to allow for more elements, such as plate and shell elements.

Section checks

The section checks currently implemented are all relatively basic. The problem with implementing more rigorous checks is that a choice has to be made of which code (Eurocode, British Standard et cetera) should be used. Some checks, like buckling, are simply difficult to implement in coded format. Some checks that have been considered are presented below.

Torsion check

The following simplified method could be used to calculate the stresses induced by torsional moments in the elements, calculated using the theory of St. Venant and disregarding the warping effect. It would therefore be assumed that the torsion resistance is $GI_t = GI_p$. Note that the torsion check would be performed in isolation, and not combined with the shear check. The utilisation could then be calculated as:

$$\eta = \frac{f_{vST}}{f_u} \tag{7.1}$$

where:

General

$$f_{vST} = \frac{T * r}{J} \tag{7.2}$$

$$J = \frac{\pi * R^4}{2} \tag{7.3}$$

Circular solid sections

$$f_{vST} = \frac{T * r}{J} \tag{7.4}$$

$$J = \frac{\pi * R^4}{2} \tag{7.5}$$

Circular hollow sections

$$f_{vST} = \frac{T * r}{J} \tag{7.6}$$

$$J = \frac{\pi}{2}(R_o^4 - R_i^4) \tag{7.7}$$

7.10 Future studies

7.10.1 Mode shape optimiser

The mode shape optimiser didn't prove to be very useful as a design tool in this thesis work (see 7.5). However, it would be interesting to see if it could be useful for a design scenario where the loads were unknown. An interesting study to make would be to study a structure at an early design stage and compare the loads to what they are at a later stage. Are the loads roughly the same or are there differences? Are there loads in directions that weren't present at the earlier stage? And if so, could the mode shape optimiser be a useful tool for handling it?

Bibliography

[Austrell et al. (2004)] Austrell, P-E et al., (2004) CALFEM, Version 3.4, KFS i Lund AB, Lund. ISBN: 91-8855823-1

[Bedney (2016)] Debney, Peter, (2016), An introduction to engineering optimisation methods, *The Structural Engineer*, Vol 94, Issue 3

[Christensen, Klarbring (2009)] Christensen, Peter W., Klarbring, Anders, (2009), An Introduction to Structural Optimization, *Solid Mechanics and its Applications*, Vol 153, Springer. ISBN: 978-1-4020-8665-6

[Dahlblom, Olsson (2010)] Dahlblom, Ola, Olsson, Karl-Gunnar, (2010), *Strukturmekanik, Modellering och analys av ramar och fackverk*, Lund: Studentlitteratur AB.

[Davidson (2016)] Davidson, Scott, Robert McNeel & Associates, (2016), *Grasshopper*, [online], accessed 2016-05-02, <http://www.grasshopper3d.com>

[ECMA International (2005)] ECMA International, (2005), *Standard ECMA-372: C++/CLI Language Specification*, Geneva

[ECMA International (2012)] ECMA International, (2012), *Standard ECMA-335: Common Language Infrastructure (CLI)*, 6th Edition, Geneva

[Freeman, Freeman (2004)] Freeman, Eric, Freeman, Elisabeth, (20104), *Head First Design Patterns*, O'Reilly Media Inc., Sebastopol, CA, USA

[Greiner, Emperador (2015)] Greiner, David, Emperador, José M., Galván, Blas and Winter, Gabriel, (2015), Evolutionary Algorithms and Metaheuristics in Civil Engineering and Construction Management, *Computational Methods in Applied Sciences*, 39, DOI 10.1007/978-3-319-20406-2_2, (accessed 2016-01-28).

[Lawson et al (1979)] Lawson, C.L., Hanson, R.J., Kincaid, D.R., Krogh, F.T., (1979) *Basic Linear Algebra Subprograms for Fortran Usage* Pages 308-23, ACM Transactions on Mathematiccal Software, Vol 5, No. 3, September 1979, DOI: 10.1145/355841.355847, (accessed 2016-05-02).

[Mitchell (1999)] Mitchell, M., & Books24x7, (1999), *An Introduction to Genetic Algorithms*, The MIT Press, Cambridge, Mass., accessed 2016-05-23, <http://common.books24x7.com.proxy.lib.chalmers.se/toc.aspx?bookid=438>

[Moler (2000)] Moler, Cleve, MathWorks (2000), accessed 2016-05-02, *MATLAB Incorporates LAPACK*, [online], <http://se.mathworks.com/company/newsletters/articles/matlab-incorporates-lapack.html?requestedDomain=www.mathworks.com>

[Mueller, Burns (2001)] Mueller, Keith M., Burns, Scott A., (2001), Fully stressed frame structures unobtainable by conventional design methodology, *International journal for numerical methods in engineering*, DOI: 10.1002/nme.261 (accessed 2016-02-02)

[Olsson, Thelin (2003)] Olsson, Karl-Gunnar and Thelin, Carl, (2003), Use of Static Eigenmodes in Mechanical Design, *Building Design Papers nr 2*, Centraltryckeriet Linköping, Linköping.

[Olsson (2006)] Olsson, Pierre, (2006), Conceptual Studies in Structural Design, Diss., Chalmers University of Technology, Majornas Copyprint, Göteborg. ISBN: 91-7291-753-9

[Ottosen, Pettersson (1992)] Ottosen, Niels, Petersson, Hans, (1992) Introduction to the Finite Element Method Harlow: Paerson Education Limited

[Robert McNeel & Associates (2016)] Robert McNeel & Associates, (2016), *Rhinoceros*, [online], accessed 2016-04-28, <http://www.rhino3d.com>

[Sanderson (2010)] Sanderson, Conrad (2010) *Armadillo: An Open Source C++ Linear Algebra Library for Fast Prototyping and Computationally Intensive Experiments*. Technical Report, NICTA

[Tedeschi (2011)] Tedeschi, Arturo, (2011), *MixExperience Tools1*, pp. 28-31 accessed 2016-05-02, <http://content.yudu.com/Library/A1qies/mixexperiencetoolsnu/resources/28.htm>

[TU Delft (2015)] TU Delft, (2015), *Getting Started with Grasshopper*, [online], accessed 2016-05-02, <http://wiki.bk.tudelft.nl/toi-pedia/Getting_Started_with_Grasshopper>

[Univ. of Tennessee et al (2016)] Univ. of Tennessee; Univ. of California, Berkeley; Univ. of Colorado Denver; NAG Ltd; (2016), *LAPACK — Linear Algebra PACKage*, [online], accessed 2016-05-02, project sponsored in part by MathWorks and Intel, <http://www.netlib.org/lapack>

[Vierling, R (2014)] Vireling, R., (2014), *octopus*, [online], Food4Rhino, page sponsor Robert McNeel &Associates, accessed 2016-05-24, <http://www.food4rhino.com/project/octopus?etx>

[Zaha Hadid (2016)] Zaha Hadid Architects, (2016), *King Abdullah Financial District Metro Station*, [online], accessed 2016-05-24, <http://www.zaha-hadid.com/architecture/king-abdullah-financial-district-metro-station/>

Figure references

All figures are the work of the authors of this report, if not otherwise stated.

Figure 3.4

Foofy (Wikipedia user), (2005), *Overview of the Common Language Infrastructure*, accessed 2016-05-04, <https://upload.wikimedia.org/wikipedia/commons/6/6a/Overview_of_the_Common_Language_Infrastructure.png>

Figure 5.7

Courtesy of Zaha Hadid Architects, n.d., render, accessed 2016-05-24, <http://www.zaha-hadid.com/architecture/king-abdullah-financial-district-metro-station/#asset/slide/b1e546>

Figure 5.8

Courtesy of Zaha Hadid Architects, n.d., render, accessed 2016-05-24, <http://www.zaha-hadid.com/architecture/king-abdullah-financial-district-metro-station/#asset/slide/d20221>

Appendices

A

Comparison of FE calculations

A test structure was designed and analysed using both MATLAB and *CIGull*. In order to verify only the finite element calculations, the geometrical and material constants used in MATLAB were taken directly from *CIGull*.

The following was the problem analysed:

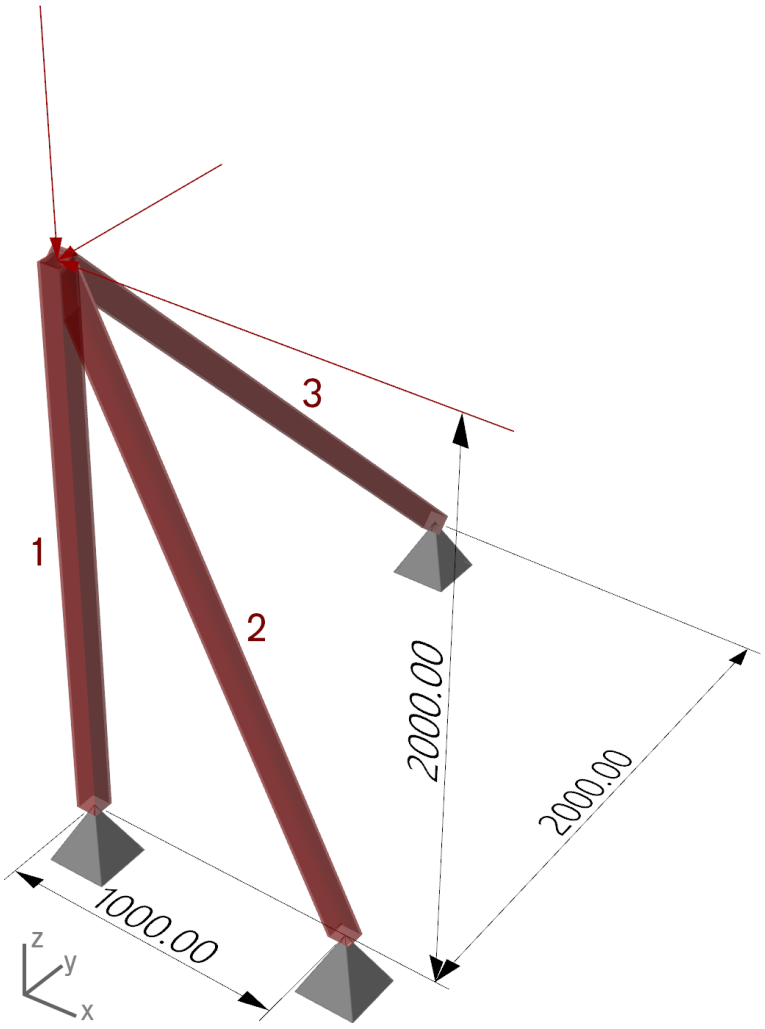


Figure A.1: A three-dimensional structure consisting of three beams. The beams are rigidly connected at the top. The structure is restrained in the x-, y- and z-directions at the bottom, but free to rotate. It is loaded with forces in the free node, in all three directions. All elements have their starting point at the top.

The following input data was used:

Section property	RHS80x80x6	[-]
E	210	[GPa]
G	80.77	[GPa]
A	19.2	[cm^2]
I_y	$1.7303 \cdot 10^{-6}$	[m^4]
I_z	$1.7303 \cdot 10^{-6}$	[m^4]
K_v	$2.5005 \cdot 10^{-6}$	[m^4]

Applied loads (at the top node):

Direction	Magnitude	Unit
x	-10	[kN]
y	-5	[kN]
z	-20	[kN]

In order to verify the calculations in *CIGull*, the element displacements were compared. The results from the *CIGull* calculations are presented in fig. A.2 below, and the entire MATLAB/CALFEM script is attached after that. Note that *CIGull* calculations are in [mm] and MATLAB results in [m]. The resulting displacements can be summarised as follows:

El.	Pos.	u		v		w		ϕ	
		CIGull	M/C*	*CIGull*	M/C*	*CIGull*	M/C*	*CIGull*	M/C*
1	1	0.247	0.247	-0.324	-0.324	-0.800	-0.800	0.005	0.005
	2	0.185	0.185	-0.245	-0.245	-0.604	-0.604	0.005	0.005
	3	0.123	0.123	-0.164	-0.164	-0.405	-0.405	0.005	0.005
	4	0.062	0.062	-0.082	-0.082	-0.203	-0.203	0.005	0.005
	5	0.0	0	0.0	0	0.0	0.000	0.005	0.005
2	1	-0.137	-0.137	-0.324	-0.324	-0.826	-0.826	0.075	0.075
	2	-0.103	-0.103	-0.246	-0.246	-0.613	-0.613	0.075	0.075
	3	-0.069	-0.069	-0.165	-0.165	-0.405	-0.406	0.075	0.075
	4	-0.034	-0.034	-0.083	-0.083	-0.202	-0.202	0.075	0.075
	5	0.0	0	0.0	-0.000	0.0	0	0.075	0.075
3	1	-0.055	-0.055	-0.404	-0.404	-0.800	-0.800	-0.270	-0.270
	2	-0.041	-0.041	-0.296	-0.296	-0.602	-0.602	-0.270	-0.270
	3	-0.027	-0.027	-0.194	-0.194	-0.403	-0.403	-0.270	-0.270
	4	-0.014	-0.014	-0.096	-0.096	-0.202	-0.203	-0.270	-0.270
	5	0.0	0	0.0	0.000	0.0	0.000	-0.270	-0.270

Table A.1: Comparison of displacements and rotations in the three elements for calculations made in *CIGull* and CALFEM. For all elements position 1 is at the top and position 5 at their respective supports.
*M/C - MATLAB / CALFEM

As can be seen in table A.1 the element displacements do not differ significantly. There are some discrepancies, which to some extent could be explained by rounding errors. When comparing the element forces (not included here) the differences were also small.

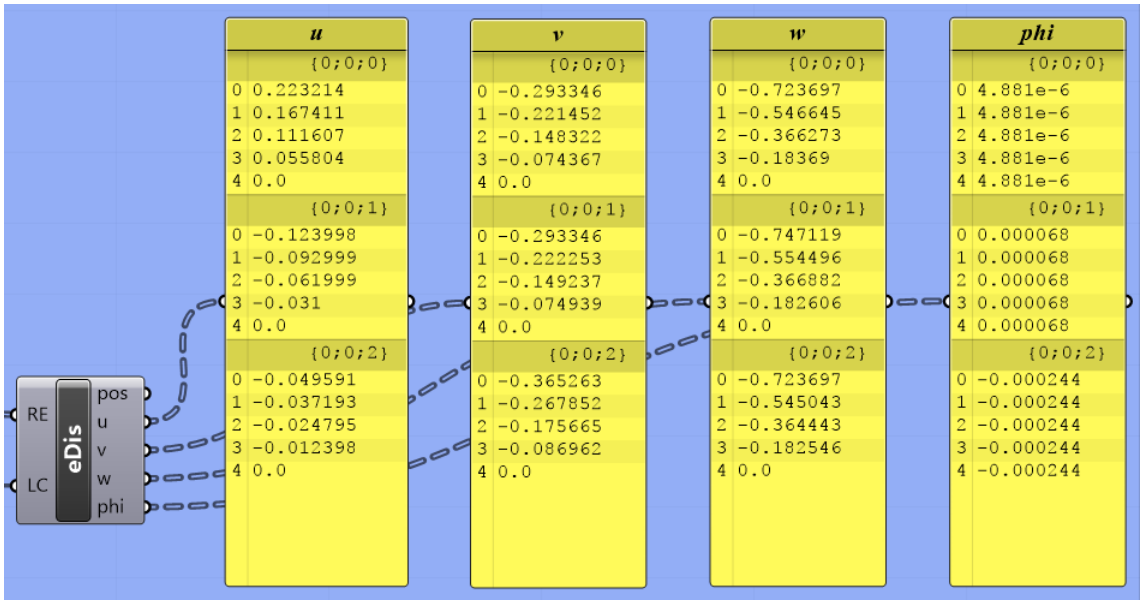


Figure A.2: The resulting element displacements in local directions. The results are structured in the grasshopper "tree-structure". Each branch (three branches in total) is an element and each item (five items on each branch) is an evaluation point. Results in [mm]

```
% Dofs
Edof = [1 1 2 3 4 5 6 7 8 9 10 11 12;
    2 1 2 3 4 5 6 13 14 15 16 17 18;
    3 1 2 3 4 5 6 19 20 21 22 23 24];

% Boundary conditions
bc = zeros(9,2);
bc(:,1)=[7 8 9 13 14 15 19 20 21]';

maxDOF=max(max(Edof));
K = zeros(maxDOF);

% Loads
f=zeros(maxDOF,1);
f(1)=-10e3;
f(2)=-5e3;
f(3)=-20e3;

% Coordinates
ex1=[0 0]; ex2=[0 1]; ex3=[0 0];
ey1=[0 0]; ey2=[0 0]; ey3=[0 2];
ez1=[2 0]; ez2=[2 0]; ez3=[2 0];
eo1=[1 0 0]; eo2=[0.894427 0 0.447214]; eo3=[1 0 0];

% Element properties
E=210e9;
G=80769230769.2308;
A=0.001737;
Iy=1.564008e-6;
Iz=Iy;
Kv=2.5005e-6;
ep=[E G A Iy Iz Kv];

% Elements
Ke1=beam3e(ex1,ey1,ez1,eo1,ep);
Ke2=beam3e(ex2,ey2,ez2,eo2,ep);
Ke3=beam3e(ex3,ey3,ez3,eo3,ep);

% Assembly
K = assem(Edof(1,:), K, Ke1);
K = assem(Edof(2,:), K, Ke2);
K = assem(Edof(3,:), K, Ke3);

% Solve
[a,r]=solveq(K,f,bc);

% Displacements
Ed=extract(Edof, a);
```

```
[es1,edi1,eci1]=beam3s(ex1, ey1, ez1, eo1, ep, Ed(1,:), [0 0 0 0], 5);
[es2,edi2,eci2]=beam3s(ex2, ey2, ez2, eo2, ep, Ed(2,:), [0 0 0 0], 5);
[es3,edi3,eci3]=beam3s(ex3, ey3, ez3, eo3, ep, Ed(3,:), [0 0 0 0], 5);

edi1

edi1 =

   1.0e-03 *

    0.2467   -0.3243   -0.7999    0.0054
    0.1850   -0.2448   -0.6042    0.0054
    0.1234   -0.1639   -0.4049    0.0054
    0.0617   -0.0822   -0.2030    0.0054
         0         0    0.0000    0.0054

edi2

edi2 =

   1.0e-03 *

   -0.1371   -0.3243   -0.8258    0.0752
   -0.1028   -0.2457   -0.6129    0.0752
   -0.0685   -0.1650   -0.4055    0.0752
   -0.0343   -0.0828   -0.2018    0.0752
         0   -0.0000         0    0.0752

edi3

edi3 =

   1.0e-03 *

   -0.0548   -0.4037   -0.7999   -0.2698
   -0.0411   -0.2961   -0.6025   -0.2698
   -0.0274   -0.1942   -0.4028   -0.2698
   -0.0137   -0.0961   -0.2018   -0.2698
         0    0.0000    0.0000   -0.2698


echo off
```

B

Cross sections 2d and 3d Frame

Index	Cross section	Area$[mm^2]$	$I_y[mm^4]$	$I_z[mm^4]$
1	RHS30x30x8	635	$0.05 \cdot 10^6$	$0.05 \cdot 10^6$
2	RHS40x30x8	795	$0.11 \cdot 10^6$	$0.07 \cdot 10^6$
3	RHS50x30x8	955	$0.22 \cdot 10^6$	$0.09 \cdot 10^6$
4	RHS40x40x8	955	$0.15 \cdot 10^6$	$0.15 \cdot 10^6$
5	RHS60x30x8	1115	$0.37 \cdot 10^6$	$0.11 \cdot 10^6$
6	RHS50x40x8	1115	$0.29 \cdot 10^6$	$0.2 \cdot 10^6$
7	RHS70x30x8	1275	$0.58 \cdot 10^6$	$0.13 \cdot 10^6$
8	RHS60x40x8	1275	$0.48 \cdot 10^6$	$0.24 \cdot 10^6$
9	RHS50x50x8	1275	$0.36 \cdot 10^6$	$0.36 \cdot 10^6$
10	RHS80x30x8	1435	$0.85 \cdot 10^6$	$0.15 \cdot 10^6$
11	RHS70x40x8	1435	$0.73 \cdot 10^6$	$0.28 \cdot 10^6$
12	RHS60x50x8	1435	$0.59 \cdot 10^6$	$0.43 \cdot 10^6$
13	RHS90x30x8	1595	$1.2 \cdot 10^6$	$0.17 \cdot 10^6$
14	RHS80x40x8	1595	$1.06 \cdot 10^6$	$0.32 \cdot 10^6$
15	RHS60x60x8	1595	$0.7 \cdot 10^6$	$0.7 \cdot 10^6$
16	RHS70x50x8	1595	$0.89 \cdot 10^6$	$0.5 \cdot 10^6$
17	RHS90x40x8	1755	$1.47 \cdot 10^6$	$0.36 \cdot 10^6$
18	RHS100x30x8	1755	$1.62 \cdot 10^6$	$0.19 \cdot 10^6$
19	RHS80x50x8	1755	$1.27 \cdot 10^6$	$0.57 \cdot 10^6$
20	RHS70x60x8	1755	$1.04 \cdot 10^6$	$0.81 \cdot 10^6$
21	RHS90x50x8	1915	$1.74 \cdot 10^6$	$0.65 \cdot 10^6$
22	RHS80x60x8	1915	$1.48 \cdot 10^6$	$0.92 \cdot 10^6$
23	RHS100x40x8	1915	$1.96 \cdot 10^6$	$0.4 \cdot 10^6$
24	RHS70x70x8	1915	$1.2 \cdot 10^6$	$1.2 \cdot 10^6$
25	RHS80x70x8	2075	$1.68 \cdot 10^6$	$1.35 \cdot 10^6$
26	RHS100x50x8	2075	$2.3 \cdot 10^6$	$0.72 \cdot 10^6$
27	RHS90x60x8	2075	$2.01 \cdot 10^6$	$1.02 \cdot 10^6$
28	RHS100x60x8	2235	$2.64 \cdot 10^6$	$1.13 \cdot 10^6$
29	RHS90x70x8	2235	$2.28 \cdot 10^6$	$1.51 \cdot 10^6$
30	RHS80x80x8	2235	$1.89 \cdot 10^6$	$1.89 \cdot 10^6$
31	RHS100x70x8	2395	$2.98 \cdot 10^6$	$1.66 \cdot 10^6$
32	RHS90x80x8	2395	$2.55 \cdot 10^6$	$2.1 \cdot 10^6$

33	RHS100x80x8	2555	$3.32 \cdot 10^6$	$2.31 \cdot 10^6$
34	RHS90x90x8	2555	$2.81 \cdot 10^6$	$2.81 \cdot 10^6$
35	RHS150x40x8	2715	$6.02 \cdot 10^6$	$0.61 \cdot 10^6$
36	RHS100x90x8	2715	$3.66 \cdot 10^6$	$3.08 \cdot 10^6$
37	RHS150x50x8	2875	$6.83 \cdot 10^6$	$1.07 \cdot 10^6$
38	RHS100x100x8	2875	$4.0 \cdot 10^6$	$4.0 \cdot 10^6$
39	RHS150x60x8	3035	$7.64 \cdot 10^6$	$1.68 \cdot 10^6$
40	RHS150x70x8	3195	$8.45 \cdot 10^6$	$2.43 \cdot 10^6$
41	RHS150x80x8	3355	$9.25 \cdot 10^6$	$3.35 \cdot 10^6$
42	RHS150x90x8	3515	$10.1 \cdot 10^6$	$4.43 \cdot 10^6$
43	RHS200x50x8	3675	$15.0 \cdot 10^6$	$1.43 \cdot 10^6$
44	RHS150x100x8	3675	$10.9 \cdot 10^6$	$5.69 \cdot 10^6$
45	RHS200x60x8	3835	$16.4 \cdot 10^6$	$2.22 \cdot 10^6$
46	RHS200x70x8	3995	$17.9 \cdot 10^6$	$3.21 \cdot 10^6$
47	RHS200x80x8	4155	$19.4 \cdot 10^6$	$4.39 \cdot 10^6$
48	RHS200x90x8	4315	$20.9 \cdot 10^6$	$5.78 \cdot 10^6$
49	RHS200x100x8	4475	$22.3 \cdot 10^6$	$7.39 \cdot 10^6$
50	RHS150x150x8	4475	$14.9 \cdot 10^6$	$14.9 \cdot 10^6$
51	RHS200x150x8	5275	$29.7 \cdot 10^6$	$18.9 \cdot 10^6$
52	RHS300x80x8	5755	$56.2 \cdot 10^6$	$6.47 \cdot 10^6$
53	RHS300x90x8	5915	$59.6 \cdot 10^6$	$8.48 \cdot 10^6$
54	RHS300x100x8	6075	$63.0 \cdot 10^6$	$10.8 \cdot 10^6$
55	RHS200x200x8	6075	$37.1 \cdot 10^6$	$37.1 \cdot 10^6$
56	RHS300x150x8	6875	$80.1 \cdot 10^6$	$27.0 \cdot 10^6$
57	RHS400x100x8	7675	$134 \cdot 10^6$	$14.2 \cdot 10^6$
58	RHS300x200x8	7675	$97.2 \cdot 10^6$	$51.8 \cdot 10^6$
59	RHS400x150x8	8475	$165 \cdot 10^6$	$35.1 \cdot 10^6$
60	RHS400x200x8	9275	$196 \cdot 10^6$	$66.6 \cdot 10^6$
61	RHS300x300x8	9275	$131 \cdot 10^6$	$131 \cdot 10^6$
62	RHS400x300x8	10875	$257 \cdot 10^6$	$165 \cdot 10^6$
63	RHS400x400x8	12475	$319 \cdot 10^6$	$319 \cdot 10^6$

Table B.1: Cross sections used in 3d frame structure in section 6.2. Sections sorted by area

C

Cross sections Dome

Index	Cross section	Area$[mm^2]$	$I_y[mm^4]$	$I_z[mm^4]$
1	RHS30x30x6	537	$0.05 \cdot 10^6$	$0.05 \cdot 10^6$
2	RHS50x30x6	777	$0.2 \cdot 10^6$	$0.08 \cdot 10^6$
3	RHS50x50x6	1017	$0.32 \cdot 10^6$	$0.32 \cdot 10^6$
4	RHS75x30x6	1077	$0.62 \cdot 10^6$	$0.13 \cdot 10^6$
5	RHS75x50x6	1317	$0.91 \cdot 10^6$	$0.47 \cdot 10^6$
6	RHS100x30x6	1377	$1.37 \cdot 10^6$	$0.17 \cdot 10^6$
7	RHS100x50x6	1617	$1.9 \cdot 10^6$	$0.61 \cdot 10^6$
8	RHS75x75x6	1617	$1.26 \cdot 10^6$	$1.26 \cdot 10^6$
9	RHS125x50x6	1917	$3.41 \cdot 10^6$	$0.76 \cdot 10^6$
10	RHS100x75x6	1917	$2.57 \cdot 10^6$	$1.62 \cdot 10^6$
11	RHS150x50x6	2217	$5.51 \cdot 10^6$	$0.9 \cdot 10^6$
12	RHS125x75x6	2217	$4.47 \cdot 10^6$	$1.98 \cdot 10^6$
13	RHS100x100x6	2217	$3.23 \cdot 10^6$	$3.23 \cdot 10^6$
14	RHS150x75x6	2517	$7.07 \cdot 10^6$	$2.34 \cdot 10^6$
15	RHS125x100x6	2517	$5.53 \cdot 10^6$	$3.9 \cdot 10^6$
16	RHS200x50x6	2817	$11.9 \cdot 10^6$	$1.2 \cdot 10^6$
17	RHS150x100x6	2817	$8.62 \cdot 10^6$	$4.56 \cdot 10^6$
18	RHS125x125x6	2817	$6.6 \cdot 10^6$	$6.6 \cdot 10^6$
19	RHS200x75x6	3117	$14.7 \cdot 10^6$	$3.05 \cdot 10^6$
20	RHS150x125x6	3117	$10.2 \cdot 10^6$	$7.66 \cdot 10^6$
21	RHS200x100x6	3417	$17.5 \cdot 10^6$	$5.89 \cdot 10^6$
22	RHS150x150x6	3417	$11.7 \cdot 10^6$	$11.7 \cdot 10^6$
23	RHS250x75x6	3717	$26.3 \cdot 10^6$	$3.77 \cdot 10^6$
24	RHS200x125x6	3717	$20.4 \cdot 10^6$	$9.79 \cdot 10^6$
25	RHS250x100x6	4017	$30.7 \cdot 10^6$	$7.21 \cdot 10^6$
26	RHS200x150x6	4017	$23.2 \cdot 10^6$	$14.8 \cdot 10^6$
27	RHS300x75x6	4317	$42.4 \cdot 10^6$	$4.49 \cdot 10^6$
28	RHS250x125x6	4317	$35.2 \cdot 10^6$	$11.9 \cdot 10^6$
29	RHS300x100x6	4617	$48.9 \cdot 10^6$	$8.54 \cdot 10^6$
30	RHS200x200x6	4617	$28.8 \cdot 10^6$	$28.8 \cdot 10^6$
31	RHS250x150x6	4617	$39.6 \cdot 10^6$	$18.0 \cdot 10^6$
32	RHS300x125x6	4917	$55.4 \cdot 10^6$	$14.0 \cdot 10^6$
33	RHS350x100x6	5217	$72.9 \cdot 10^6$	$9.87 \cdot 10^6$
34	RHS250x200x6	5217	$48.6 \cdot 10^6$	$34.5 \cdot 10^6$

35	RHS300x150x6	5217	$61.9 \cdot 10^6$	$21.1 \cdot 10^6$
36	RHS350x125x6	5517	$81.8 \cdot 10^6$	$16.2 \cdot 10^6$
37	RHS350x150x6	5817	$90.7 \cdot 10^6$	$24.2 \cdot 10^6$
38	RHS400x100x6	5817	$103 \cdot 10^6$	$11.2 \cdot 10^6$
39	RHS300x200x6	5817	$74.9 \cdot 10^6$	$40.1 \cdot 10^6$
40	RHS250x250x6	5817	$57.5 \cdot 10^6$	$57.5 \cdot 10^6$
41	RHS400x125x6	6117	$115 \cdot 10^6$	$18.3 \cdot 10^6$
42	RHS400x150x6	6417	$127 \cdot 10^6$	$27.3 \cdot 10^6$
43	RHS300x250x6	6417	$87.8 \cdot 10^6$	$66.4 \cdot 10^6$
44	RHS350x200x6	6417	$108 \cdot 10^6$	$45.8 \cdot 10^6$
45	RHS450x125x6	6717	$156 \cdot 10^6$	$20.4 \cdot 10^6$
46	RHS400x200x6	7017	$150 \cdot 10^6$	$51.4 \cdot 10^6$
47	RHS450x150x6	7017	$171 \cdot 10^6$	$30.4 \cdot 10^6$
48	RHS300x300x6	7017	$101 \cdot 10^6$	$101 \cdot 10^6$
49	RHS350x250x6	7017	$126 \cdot 10^6$	$75.4 \cdot 10^6$
50	RHS500x125x6	7317	$205 \cdot 10^6$	$22.5 \cdot 10^6$
51	RHS450x200x6	7617	$200 \cdot 10^6$	$57.1 \cdot 10^6$
52	RHS500x150x6	7617	$224 \cdot 10^6$	$33.5 \cdot 10^6$
53	RHS350x300x6	7617	$144 \cdot 10^6$	$114 \cdot 10^6$
54	RHS400x250x6	7617	$173 \cdot 10^6$	$84.3 \cdot 10^6$
55	RHS350x350x6	8217	$162 \cdot 10^6$	$162 \cdot 10^6$
56	RHS550x150x6	8217	$286 \cdot 10^6$	$36.6 \cdot 10^6$
57	RHS450x250x6	8217	$230 \cdot 10^6$	$93.2 \cdot 10^6$
58	RHS400x300x6	8217	$197 \cdot 10^6$	$127 \cdot 10^6$
59	RHS500x200x6	8217	$260 \cdot 10^6$	$62.7 \cdot 10^6$
60	RHS550x200x6	8817	$330 \cdot 10^6$	$68.4 \cdot 10^6$
61	RHS400x350x6	8817	$220 \cdot 10^6$	$179 \cdot 10^6$
62	RHS600x150x6	8817	$359 \cdot 10^6$	$39.8 \cdot 10^6$
63	RHS500x250x6	8817	$297 \cdot 10^6$	$102 \cdot 10^6$
64	RHS450x300x6	8817	$259 \cdot 10^6$	$140 \cdot 10^6$
65	RHS600x200x6	9417	$411 \cdot 10^6$	$74.0 \cdot 10^6$
66	RHS550x250x6	9417	$375 \cdot 10^6$	$111 \cdot 10^6$
67	RHS400x400x6	9417	$243 \cdot 10^6$	$243 \cdot 10^6$
68	RHS450x350x6	9417	$289 \cdot 10^6$	$197 \cdot 10^6$
69	RHS500x300x6	9417	$333 \cdot 10^6$	$153 \cdot 10^6$
70	RHS550x300x6	10017	$419 \cdot 10^6$	$166 \cdot 10^6$
71	RHS600x250x6	10017	$464 \cdot 10^6$	$120 \cdot 10^6$
72	RHS500x350x6	10017	$370 \cdot 10^6$	$215 \cdot 10^6$
73	RHS650x200x6	10017	$504 \cdot 10^6$	$79.7 \cdot 10^6$
74	RHS450x400x6	10017	$319 \cdot 10^6$	$266 \cdot 10^6$
75	RHS600x300x6	10617	$517 \cdot 10^6$	$179 \cdot 10^6$
76	RHS650x250x6	10617	$567 \cdot 10^6$	$129 \cdot 10^6$
77	RHS500x400x6	10617	$407 \cdot 10^6$	$290 \cdot 10^6$
78	RHS550x350x6	10617	$464 \cdot 10^6$	$233 \cdot 10^6$
79	RHS450x450x6	10617	$348 \cdot 10^6$	$348 \cdot 10^6$
80	RHS600x350x6	11217	$570 \cdot 10^6$	$250 \cdot 10^6$

81	RHS650x300x6	11217	$629 \cdot 10^6$	$192 \cdot 10^6$
82	RHS500x450x6	11217	$443 \cdot 10^6$	$378 \cdot 10^6$
83	RHS550x400x6	11217	$508 \cdot 10^6$	$313 \cdot 10^6$
84	RHS500x500x6	11817	$480 \cdot 10^6$	$480 \cdot 10^6$
85	RHS650x350x6	11817	$691 \cdot 10^6$	$268 \cdot 10^6$
86	RHS550x450x6	11817	$552 \cdot 10^6$	$407 \cdot 10^6$
87	RHS600x400x6	11817	$623 \cdot 10^6$	$336 \cdot 10^6$
88	RHS600x450x6	12417	$676 \cdot 10^6$	$437 \cdot 10^6$
89	RHS550x500x6	12417	$597 \cdot 10^6$	$516 \cdot 10^6$
90	RHS650x400x6	12417	$753 \cdot 10^6$	$360 \cdot 10^6$
91	RHS650x450x6	13017	$815 \cdot 10^6$	$466 \cdot 10^6$
92	RHS600x500x6	13017	$729 \cdot 10^6$	$553 \cdot 10^6$
93	RHS550x550x6	13017	$641 \cdot 10^6$	$641 \cdot 10^6$
94	RHS600x550x6	13617	$782 \cdot 10^6$	$685 \cdot 10^6$
95	RHS650x500x6	13617	$878 \cdot 10^6$	$590 \cdot 10^6$
96	RHS650x550x6	14217	$940 \cdot 10^6$	$730 \cdot 10^6$
97	RHS600x600x6	14217	$835 \cdot 10^6$	$835 \cdot 10^6$
98	RHS650x600x6	14817	$1000 \cdot 10^6$	$888 \cdot 10^6$
99	RHS650x650x6	15417	$1060 \cdot 10^6$	$1060 \cdot 10^6$

Table C.1: Cross sections used in dome structure in section 6.3. Sections sorted by area

D

Cross sections KAFD

Index	Cross section	Area$[mm^2]$	$I_y[mm^4]$	$I_z[mm^4]$
1	RHS80x80x5	1473	$1.37 \cdot 10^6$	$1.37 \cdot 10^6$
2	RHS100x100x5	1873	$2.79 \cdot 10^6$	$2.79 \cdot 10^6$
3	RHS100x100x10	3493	$4.62 \cdot 10^6$	$4.62 \cdot 10^6$
4	RHS250x100x7	4651	$35.2 \cdot 10^6$	$8.18 \cdot 10^6$
5	RHS150x150x10	5493	$17.7 \cdot 10^6$	$17.7 \cdot 10^6$
6	RHS350x170x9	8949	$141 \cdot 10^6$	$45.5 \cdot 10^6$
7	RHS450x150x10	11493	$272 \cdot 10^6$	$47.2 \cdot 10^6$
8	RHS350x250x16	17901	$300 \cdot 10^6$	$177 \cdot 10^6$

Table D.1: RHS cross sections used in the KAFD case studie in section 6.3. Sections sorted by area

Index	Cross section	Area$[mm^2]$	$I_y[mm^4]$	$I_z[mm^4]$
1	CHS139.7x5	4310	$40.6 \cdot 10^6$	$40.6 \cdot 10^6$
2	CHS139.7x10	8463	$76.9 \cdot 10^6$	$76.9 \cdot 10^6$
3	CHS193.7x8	9535	$172 \cdot 10^6$	$172 \cdot 10^6$
4	CHS193.7x10	11856	$211 \cdot 10^6$	$211 \cdot 10^6$
5	CHS273x20	33050	$1140 \cdot 10^6$	$1140 \cdot 10^6$
6	CHS355.6x20	43429	$2600 \cdot 10^6$	$2600 \cdot 10^6$
7	CHS457x40	109830	$10500 \cdot 10^6$	$10500 \cdot 10^6$

Table D.2: CHS cross sections used in the KAFD case studie in section 6.3. Sections sorted by area

E

3D-frame eigenvalues

Mode	Eigenvalue	Norm. eigenval.
1	916	1.0
2	916	1.0
3	920	1.01
4	4814	5.26
5	8192	8.95
6	8307	9.07
7	8307	9.07
8	8986	9.82
9	8986	9.82
10	11867	12.96
11	15300	16.71
12	15300	16.71
13	17208	18.8
14	19508	21.31
15	19508	21.31
16	20727	22.64
17	21157	23.11
18	21350	23.32
19	23354	25.51
20	23354	25.51
21	23708	25.9
22	23708	25.9
23	24027	26.24
24	24027	26.24
25	25014	27.32
26	25014	27.32
27	26685	29.15
28	28242	30.85
29	28242	30.85
30	28331	30.95
31	28338	30.95
32	30397	33.2
33	37446	40.9

Mode	Eigenvalue	Norm. eigenval.
34	37446	40.9
35	38928	42.52
36	39043	42.65
37	40187	43.9
38	40740	44.5
39	41775	45.63
40	41775	45.63
41	42297	46.2
42	42297	46.2
43	42394	46.31
44	42394	46.31
45	43067	47.04
46	43218	47.21
47	43436	47.44
48	43436	47.44
49	44820	48.96
50	45239	49.41
51	46593	50.89
52	46593	50.89
53	46751	51.07
54	46761	51.08
55	47606	52.0
56	47606	52.0
57	52603	57.46
58	52603	57.46
59	60784	66.39
60	60784	66.39
61	61705	67.4
62	62523	68.29
63	66044	72.14
64, 65	66207	72.32
66	70207	76.69
67	16987143	18554.96

Table E.1: Eigenvalues for the first 46 modes. Normalized to the first eigenvalue