Norwegian University of
Science and Technology

Analysis Software for Calculation of Cross-Section Parameters for Complex Beam Cross-Sections

Simen Vandvik Aakhus
Samson Svendsen

Master of Science in Engineering and ICT
Submission date: June 2016
Supervisor: Bjørn Haugen, IPM

Norwegian University of Science and Technology
Department of Engineering Design and Materials

Abstract

Cross-sectional parameters for complex cross-sections are very difficult to calculate by analytical means. Dynamic calculations of aircraft and wind turbines often make use of beam elements for the wings and turbine blades, and are dependent on correct cross-sectional parameters. As a solution to this we have continued the development of an analysis software for complex beam cross-sections, based on the finite element method. Both wind turbine blades and aircraft wings are often made of composite materials with a layered structure. Cross-sections consisting of such materials lend themselves very well to meshes with quadrilateral elements. By implementing support for four- and nine-node quadrilateral elements, as well as developing a unified analysis for torsion and shear, we have greatly increased the composite functionality of the software. A recurring problem with complex geometry is that the analysis can be computationally expensive. By supporting thin-walled approximations to cross-sections we have significantly reduced the computational effort needed, while still retaining sufficient accuracy. The work has been extensively tested, and is shown to provide highly accurate results when compared to both known solutions and similar programs.

Sammendrag

Det er vanskelig å beregne tverrsnittsdata for komplekse tverrsnitt analytisk. Dynamiske beregninger av fly og vind turbiner bruker ofte bjelkemodeller for vingene og turbinbladene, hvor resultatene er avhengig av korrekte tverrsnittsdata. For å løse denne problemstillingen har vi fortsatt utviklingen av et analyseprogram for komplekse tverrsnitt, basert på elementmetoden. Både flyvinger og turbinblader lages ofte av kompositt materialer med en lagvis struktur. Kvadrilaterale elementer passer bedre enn triangulære element til å beskrive tverrsnitt bestående av slike strukturer. Ved å implementere fire- og ni-noders kvadrilaterale elementer, i tillegg til å utvikle en samlet analyse for påsatt torsjon og skjærlast, har vi gjort store fremsteg mot en analyse av kompositt materialer. Et gjennomgående problem for tverrsnitt med vanskelige geometrier er at analysene blir store beregningsmessig. Som en konsekvens av dette har vi implementert støtte for en tynnvegget tilnærming av tverrsnitt. Dette medfører mye raskere beregninger, samtidig som at tilstrekkelig presisjon for resultatene er opprettholdt. Programvaren har blitt testet grundig mot både kjente løsninger og lignende programmer. Resultatene har gjennomgående vært meget presise.

Preface

This master thesis is a continuation of the work done by Kristian Strømstad for his master thesis during the spring of 2014. It was written for the Department of Engineering Design and Materials (IPM) at the Norwegian University of Science and Technology (NTNU). The reader should be familiar with the basic concepts of Finite Element Analysis (FEA), matrix theory, and have experience with object-oriented programming (OOP).

Trondheim, June 9th, 2016

Samson Svendsen

Samson C. Svendsen

Simen Aakhus

Simen V. Aakhus

Acknowledgment

We would like to give our thanks to associate professor Bjørn Haugen for his excellent help and supervision throughout this project. His open door policy combined with his enthusiasm proved very valuable. We would also like to direct a special thanks to Kolbein Bell for allowing us to use multiple illustrations from his book *An engineering approach to Finite Element Analysis of linear structural mechanics problems.*

S.C.S. and S.V.A.

Contents

Acronyms and symbols

Acronyms

CCSC	- Complex Cross-Section Calculator
CST	- Constant Strain Triangle
DFS	- Depth-first search
dofs	- degrees of freedom
FEA	- Finite Element Analysis
FEM	- Finite Element Method
GUI	- Graphical User Interface
GPL	- General Public License
HTML	- HyperText Markup Language
LST	- Linear Strain Triangle
IPM	- Department of Engineering Design and Materials
NTNU	- Norwegian University of Science and Technology
OOP	- Object Oriented Programming
PVD	- Principle of Virtual Displacements
STL	- Stereolithography
T3 element	- Three-node triangular element
T6 element	- Six-node triangular element
Q4 element	- Four-node quadratic element
Q9 element	- Nine-node quadratic element
VTK	- Visualization Toolkit

Scalars

A	- area	s	- local coordinate (thin-wall)
Bi	- bimoment	t_k	- thickness of element k
C	- area center	u, v, w	- displacement components
E	- Young's modulus	x, y, z	- Cartesian coordinates
EI	- bending stiffness	x_s, y_s	- shear center coordinates
EI_ω	- warping stiffness	Π	- total potential energy
G	- shear modulus	α	- angle of principal axis
GI_t	- torsional stiffness	γ	- shear strain
I	- second area of moment	ϵ	- normal strains
I_t	- torsion constant	ζ_i	- area coordinates (i = 1, 2, 3)
I_ω	- warping constant	ξ, η	- natural coordinates
J	- jacobi determinant	η_k	- direction of element k (thin-wall)
L	- length of beam	θ	- rate of twist ($\frac{d\Phi}{dz}$)
M	- bending moment	κ^e	- element shear deformation factor
N	- applied normal force	κ	- shear deformation factor
S	- shear center	ν	- Poisson's ratio
T_{stv}	- applied St.Venant torsion	σ	- axial (normal) stress
T_{wrp}	- applied warping torsion	τ	- shear stress
U	- strain energy	ϕ	- angle of twist
V	- applied shear force	Ψ	- warping displacement
W	- total potential of external forces	ω	- sectorial area around shear center
l_k	- length of element k	ω_c	- sectorial area around area center
q	- shear flow		

Matrices and vectors

$\boldsymbol{a}$	- cell area matrix	$\boldsymbol{S}$	- nodal point forces of an element
$\boldsymbol{B}^*$	- strain-displacement matrix	$\mathbf{S}^0$	- element load vector
$\boldsymbol{B}^{**}$	- coefficient matrix	$\boldsymbol{f}$	- system load vector (thin-wall)
$\boldsymbol{C}$	- compliance matrix	$\boldsymbol{q}_0$	- mid-point shear flow (thin-wall)
$\boldsymbol{E}$	- stress recovery matrix	$\boldsymbol{x}$	- nodal x-coordinates of an element
$\boldsymbol{F}$	- volume forces	$\boldsymbol{y}$	- nodal y-coordinates of an element
$\boldsymbol{G}$	- shear modulus matrix	$\boldsymbol{u}$	- displacement components
$\boldsymbol{J}$	- Jacobian matrix	$\boldsymbol{v}$	- nodal warping displacements
$\boldsymbol{K}$	- global stiffness matrix	$\boldsymbol{\epsilon}$	- strain vector
$\boldsymbol{k}_e$	- element stiffness matrix	$\boldsymbol{\epsilon}_0$	- initial strain vector
$\boldsymbol{N}$	- shape function matrix	$\tilde{\boldsymbol{W}}_e$	- external virtual work
$\boldsymbol{p}$	- modified shear flow vector	$\tilde{\boldsymbol{W}}_i$	- internal virtual work
$\boldsymbol{R}$	- global load vector	$\boldsymbol{\phi}$	- tractions
$\boldsymbol{r}$	- global displacement vector		

Chapter 1

Introduction

1.1 Background

Cross-sectional parameters for complex cross-sections are very hard to calculate by analytical means. Dynamic calculations of aircraft and wind turbines often make use of beam elements for the wings and turbine blades, and are dependent on correct cross-sectional parameters. For this reason, the Department of Engineering Design and Materials (IPM) started the work on a software solving this problem with the use of finite element analysis (FEA). Bjørn Haugen, associate professor at IPM, initiated the development of the software now named Complex Cross-Section Calculator (CCSC). It was further developed by Kristian Strømstad during his master thesis[1] (spring 2014). At the end of his thesis the software had all the functionality needed to calculate basic cross-sectional parameters such as area, second moments of area, and angle of principal axis. Additionally the software was able to correctly compute axial stresses due to applied normal force and pure bending. An analysis of stress distributions due to applied torsional moment using three-node triangular elements was also implemented, but it was producing incorrect numerical values.

During our project thesis (fall 2015) we extended the functionality to include a stress analysis due to shear loading. Additionally we implemented support for analysis using six-node triangular elements. These elements are quadratic and consequently CCSC can obtain linear variation of stress over the element, which provides more accurate analysis. By the end of the project we concluded that the quadratic element implementation was correct, while the shear analysis was producing incorrect numerical values.

1.2 Objectives

The main objectives of this master thesis are to

1. Develop support for analysis of thin-walled cross-sections, using two- and three-node line elements
2. Develop four- and nine-node quadrilateral element functionality for massive cross-sections
3. Implement support for composite materials for the existing elements
4. Evaluate and restructure existing code

In addition to the main goals we also had to correct both the torsion and shear analysis, as well as implement calculations for parameters such as the torsion constant, shear center, and shear deformation factors. As a long term goal CCSC should fully support composite materials. An extensive HTML documentation was also written, to further facilitate future development of the application.

1.3 Approach

Much of the theory for massive analysis is based on Kolbein Bell's *An engineering approach to Finite Element Analysis of linear structural mechanics problems*[2]. This includes element formulation and numerical integration. The approach used for the thin-wall analysis is based on the theory presented in the CrossX documentation[3], and CCSC is in many ways developed to be a spiritually successor to CrossX, written in modern C++ and with added functionality for composites.

All code for this project was written in C++ using Microsoft Visual Studio 2015. For matrix operations and equation solving we used the linear algebra template library Eigen[4]. GitHub[5] was used for version control and as a collaborative tool. To create the mesh files for the analyses we have used the open source software Gmsh 2.11[6]. For visualization of our results we used the visualizing tool ParaView[7], and the VTK file format[8]. All documentation for CCSC was generated by Doxygen[9]

1.4 Structure of the report

The rest of the report is organized as follows. Chapter 2 describes in detail how the finite element method (FEM) is used for analysis of beams subjected to torsion and shear loading. This includes descriptions of element types, numerical integration, stress calculations, and a detailed introduction to composite analysis. Chapter 3 elaborates on analysis of thin-walled cross-sections of a cantilever subjected to torsion, shear, normal force, and bimoment loading. In Chapters 4 and 5 we explain the implementation of the theory. This includes descriptions of class structures and information flow, numerical integration, stress analysis, and visualization of results. In Chapter 6 we validate the implementation by comparing results to well-known solutions, as well as results retrieved from CrossX. We also investigate the running time for both massive and thin-wall analysis. In Chapter 7 we summarize our thesis, discuss our findings, and make suggestions for future work.

Chapter 2

Theory - Massive Cross-Sections

This chapter presents the theory behind the implementation of massive cross-section analysis. This includes a deeper look into the finite element method, triangular and quadrilateral elements, numerical integration, and calculation of stress distributions due to bending moments, torsion, and shear loading. Towards the end we present an introduction to analysis of composite materials, which unifies torsion and shear loading.

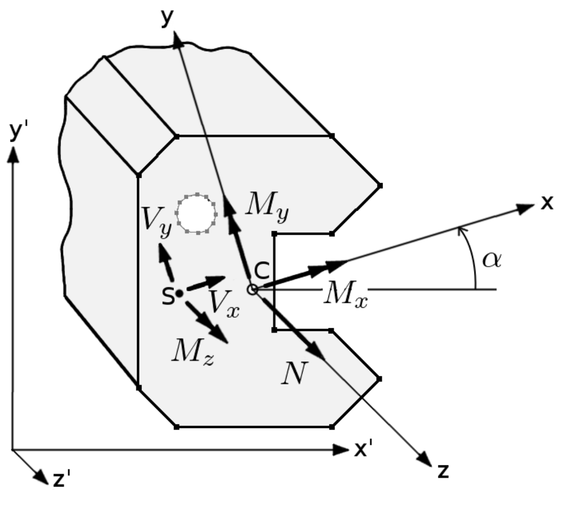


Figure 2.1: Example of a massive cross-section with coordinate system and applied section forces in the area center (C) and shear center (S)

2.1 The finite element method

The finite element method is a powerful technique used to solve structural problems with the use of numerical approximations to physical problems. The development of FEM has been closely related to the advancement of the digital computer. Moore's law, postulating that the number of transistors on a circuit doubles every 18 months, has been valid for the latter half of the 20th century, decreasing the cost of computational power immensely. As a consequence, today's computer programs are able to solve equations with several millions of unknowns, making FEM an effective tool suitable for most branches of engineering.

The general idea behind FEM is to divide a system with complex behaviour into a set of subsystems which can be analyzed more easily. A sub-system consists of elements, and each element has a set of nodes. The elements connect to each other through these nodes. For every node several loads and corresponding displacements are defined, which are directly related to the node's degrees of freedoms (dofs). The loads and displacements are represented by the vectors $\boldsymbol{S}$ and $\boldsymbol{v}$ respectively. For an element with n nodes and n dofs (out-of plane displacements), the relation between $\boldsymbol{S}$ and $\boldsymbol{v}$ is described through the element stiffness matrix $\boldsymbol{k_e}$ with n rows and n columns.

$$\boldsymbol{S}^T = \begin{bmatrix} S_1 & S_2 & \cdots & S_n \end{bmatrix}$$

$$\boldsymbol{v}^T = \begin{bmatrix} v_1 & v_2 & \cdots & v_n \end{bmatrix}$$

Establishing the stiffness matrix is a large part of solving any problem using FEM, and results in the following key equation

$$\boldsymbol{S} = \boldsymbol{k_e} \boldsymbol{v} \tag{2.1}$$

However, there is no simple way of establishing this relation. By assuming the displacement field of an element to be described as a function of the nodal displacements, we allow the displacements to be described at any point in the element. Eq. (2.2) states this relationship. The *shape-* (or *interpolation*) functions $\boldsymbol{N}$ are specific to each element type, and is further explained in Section 2.4.2. It is important to recognise that this assumption makes FEM an approximate solution. The correct displacement field cannot be assumed even for simple cross-section analysis.

$$\begin{bmatrix} u \\ v \end{bmatrix} = \boldsymbol{u} = \boldsymbol{N}\boldsymbol{v} \tag{2.2}$$

We introduce **B** as the strain-displacement matrix and define the strains as

$$\boldsymbol{\epsilon} \;=\; \Delta\boldsymbol{u} \;=\; \Delta\boldsymbol{N}\boldsymbol{v} \;=\; \boldsymbol{B}\boldsymbol{v} \tag{2.3}$$

2.2 Principle of virtual displacements

The principle of virtual displacements (PVD) is the basis for the displacement-based finite element solution with respect to the warping function $\Psi(x,y)$. We refer to Bell[2] for a precise interpretation of this principle:

> If a system of *real* external and internal *forces* (stresses) that are in *static equilibrium* are subjected to *any* set of *virtual*, but *kinematically compatible*, displacements and deformations, then the *virtual work* performed by the real external forces over the virtual displacements is *equal* to the *virtual work* performed by the real internal forces (stresses) over the virtual deformations (strains).

By subjecting an element to a set of virtual displacements $\tilde{\boldsymbol{v}}$, and choosing the virtual displacements inside an element $\tilde{\boldsymbol{u}}$ according to Eq. (2.2) we get

$$\tilde{\boldsymbol{u}} \;=\; \boldsymbol{N}\tilde{\boldsymbol{v}} \tag{2.4}$$

with the corresponding virtual strains

$$\tilde{\boldsymbol{\epsilon}} \;=\; \boldsymbol{B}\tilde{\boldsymbol{v}} \tag{2.5}$$

PVD states that the virtual work done over the virtual displacements by the real external forces, is equal to the virtual work done over the virtual strains by the real internal forces. In other words $\tilde{\boldsymbol{W}}_{\boldsymbol{e}} = \tilde{\boldsymbol{W}}_{\boldsymbol{i}}$. For an arbitrary element this equation can be written as

$$\tilde{\boldsymbol{v}}^T\boldsymbol{S} + \int_{V_e} \tilde{\boldsymbol{u}}^T\boldsymbol{F}dV + \int_{S_T} \tilde{\boldsymbol{u}}^T\boldsymbol{\phi}dS = \int_{V_e} \tilde{\boldsymbol{\epsilon}}^T\boldsymbol{\sigma}dV \tag{2.6}$$

Virtual denotes that the displacements are not *real* displacements of the body. Rather, the virtual displacements are independent of the actual displacements, and is used as a thought experiment to establish the equilibrium equation in (2.6). For linear elastic, isotropic materials, $\boldsymbol{G}$ (often referred to as $\boldsymbol{C}$) is simply a diagonal matrix with the shear modulus G on the diagonal. By substituting $\tilde{\boldsymbol{u}}$, $\tilde{\boldsymbol{\epsilon}}$, $\boldsymbol{G}$ and $\boldsymbol{\sigma}$ in the above equation and rearranging we get

$$\tilde{\boldsymbol{v}}^T\boldsymbol{S} = \tilde{\boldsymbol{v}}^T\int_{V_e} \boldsymbol{B}^T\boldsymbol{G}(\boldsymbol{B}\boldsymbol{v}+\boldsymbol{\epsilon}_0)dV - \tilde{\boldsymbol{v}}^T\int_{V_e} \boldsymbol{N}^T\boldsymbol{F}dV - \tilde{\boldsymbol{v}}^T\int_{S_T} \boldsymbol{N}^T\boldsymbol{\phi}dS \tag{2.7}$$

Since this equation must be valid for any virtual displacement $\tilde{\boldsymbol{v}}$ we end up with

$$\boldsymbol{S} = \int_{V_e} \boldsymbol{B}^T\boldsymbol{G}\boldsymbol{B}dV\boldsymbol{v} + \int_{V_e} \boldsymbol{B}^T\boldsymbol{G}\boldsymbol{\epsilon}_0dV - \int_{V_e} \boldsymbol{N}^T\boldsymbol{F}dV - \int_{S_T} \boldsymbol{N}^T\boldsymbol{\phi}dS = \boldsymbol{k}_e\boldsymbol{v} + \boldsymbol{S}_e^0 \tag{2.8}$$

where the element stiffness matrix is

$$\boldsymbol{k}_e = \int_{V_e} \boldsymbol{B}^T\boldsymbol{G}\boldsymbol{B}dV \tag{2.9}$$

and the consistent load vector is

$$\boldsymbol{S}_e^0 = \boldsymbol{S}_{\epsilon_0}^0 + \boldsymbol{S}_F^0 + \boldsymbol{S}_\phi^0 \tag{2.10}$$

with the following contributions due to initial strain, volume forces and surface forces respectively

$$\boldsymbol{S}_{\epsilon_0}^0 = \int_{V_e} \boldsymbol{B}^T\boldsymbol{G}\boldsymbol{\epsilon}_0dV \tag{2.11a}$$

$$\boldsymbol{S}_F^0 = -\int_{V_e} \boldsymbol{N}^T\boldsymbol{F}dV \tag{2.11b}$$

$$\boldsymbol{S}_\phi^0 = -\int_{S_T} \boldsymbol{N}^T\boldsymbol{\phi}dS \tag{2.11c}$$

2.3 Element types

CCSC offers support for meshes with four different element types; three- and six-node triangular elements (**T3** and **T6**) and four- and nine-node quadrilateral elements (**Q4** and **Q9**). The four element types are illustrated in Fig. 2.2. T3 and Q4 elements only have nodes in their corners, and as a consequence their shape functions are linear and bilinear respectively. The T6 and Q9 elements have additional nodes on the mid-point of each edge (Q9 also has a node in the middle of the element), giving them quadratic and biquadratic shape functions respectively. All shape functions will be defined over the next few sections.

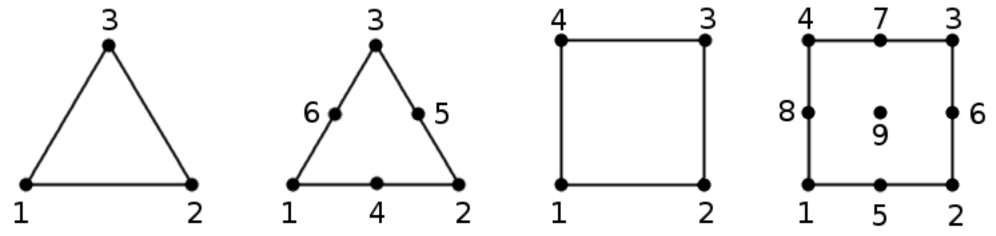


Figure 2.2: T3, T6, Q4, and Q9 elements respectively

2.4 Triangular elements

As previously mentioned, CCSC offers support for two triangular element types (T3 and T6). The T3 element only has a linear displacement field, which makes the strain field constant. Because of this, the T3 element is often referred to as the *constant strain triangle* (CST). The T6 element on the other hand, has a quadratic displacement field, which gives a linear strain field across the element. Hence it is often called the *linear strain triangle* (LST). The linear strain elements should in general provide more accurate results, particularly for coarse meshes, as their solutions converge faster. This will be examined further in Section 6.1.2.

It is important to mention that the triangular elements have been implemented with straight edges. Therefore there is no need for mapping, nor the Jacobian matrix, unlike the isoparametric quadrilaterals. Still, we consider the workload needed to implement support for curved-side T6 elements to be small.

2.4.1 Interpolation using area coordinates

For the implementation of triangular elements in CCSC we make use of area coordinates, which is a normalized (dimensionless) system. This has clear advantages over using the global coordinate system on an element level. Area coordinates are defined by splitting a triangle into three smaller triangles as illustrated in Fig. 2.3. The area coordinates are given as

$$\zeta_i = \frac{A_i}{A} = \frac{\frac{1}{2} z_i L_i}{\frac{1}{2} H_i L_i} = \frac{z_i}{H_i} \qquad i = 1,2,3 \tag{2.12}$$

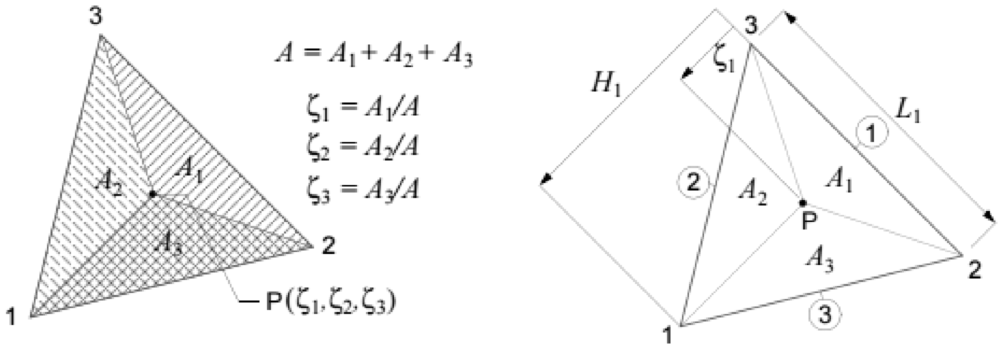


Figure 2.3: Area coordinates $(\zeta_1, \zeta_2, \zeta_3)$ for a plane triangle, from Bell[2] p. 144

From this we interpret ζ_i to be the normalized distance from edge i to an arbitrary point P within the triangle. The three area coordinates are constrained by each other as

$$\zeta_1 + \zeta_2 + \zeta_3 = 1 \tag{2.13}$$

The relationship between the Cartesian reference system and area coordinates are given by the following equations

$$x = \zeta_1 x_1 + \zeta_2 x_2 + \zeta_3 x_3 \tag{2.14a}$$

$$y = \zeta_1 y_1 + \zeta_2 y_2 + \zeta_3 y_3 \tag{2.14b}$$

The area coordinates can be expressed by Cartesian coordinates using $x_{ij} = x_i - x_j$ and $y_{ij} = y_i - y_j$, and numbering the corners 1,2, and 3 counter clockwise (with A being the area of the element)

$$\begin{bmatrix} \zeta_1 \\ \zeta_2 \\ \zeta_3 \end{bmatrix} = \frac{1}{2A} \begin{bmatrix} y_{23} & x_{32} & (x_2 y_3 - x_3 y_2) \\ y_{31} & x_{13} & (x_3 y_1 - x_1 y_3) \\ y_{12} & x_{21} & (x_1 y_2 - x_2 y_1) \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \tag{2.15}$$

Referring to Eq. (2.3), we establish the strain-displacement matrix as

$$\boldsymbol{B} = \Delta \boldsymbol{N} = \begin{bmatrix} \frac{\partial \boldsymbol{N}}{\partial x} \\ \frac{\partial \boldsymbol{N}}{\partial y} \end{bmatrix} \tag{2.16}$$

By differentiating Eq. (2.15) we establish the following relationships

$$\frac{\partial \zeta_i}{\partial x} = \frac{y_{jk}}{2A} \qquad \frac{\partial \zeta_i}{\partial y} = \frac{x_{kj}}{2A} \tag{2.17}$$

which in turn help us determine the terms in **B** as

$$\boldsymbol{B} = \begin{bmatrix} \frac{\partial \boldsymbol{N}}{\partial x} \\ \frac{\partial \boldsymbol{N}}{\partial y} \end{bmatrix} = \frac{1}{2A} \begin{bmatrix} y_{23} & y_{31} & y_{12} \\ x_{32} & x_{13} & x_{21} \end{bmatrix} \begin{bmatrix} \frac{\partial \boldsymbol{N}}{\partial \zeta_1} \\ \frac{\partial \boldsymbol{N}}{\partial \zeta_2} \\ \frac{\partial \boldsymbol{N}}{\partial \zeta_3} \end{bmatrix} \tag{2.18}$$

2.4.2 Three-node triangular element

The T3 element is the simplest of the four element types, and it was the only element implemented in CCSC when we started working on the software. We will now look to determine its shape functions by making use of the *"0-lines" method* described in Appendix A.3. Fig. 2.4 illustrates this method used on a T3 element.

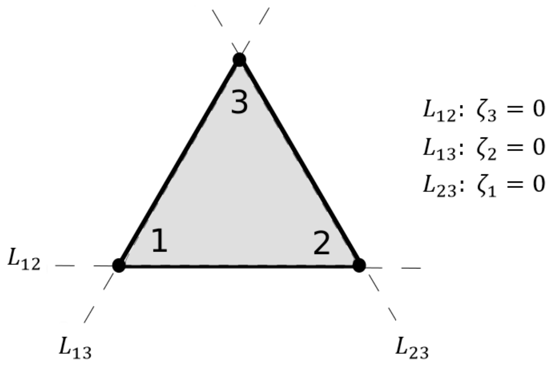


Figure 2.4: Construction of shape functions for T3 element by the "0-lines" method

We look at node number **1**, in an attempt to establish its shape function:

1. Assume the shape function $N_1 = c_1\,L_{23}$, as it is zero at every node except **1**
2. $\implies N_1 = c_1\,\zeta_1$
3. Fulfill unity requirement: $N_1(\zeta_1 = 1, \zeta_2 = 0, \zeta_3 = 0) = c_1 \cdot 1 = 1 \implies c_1 = 1$
4. $\implies N_1 = \zeta_1$

The procedure is identical for the two other nodes. We conclude that for the three-node triangular element, the shape functions are simply the area coordinates themselves as given in Eq. (2.19).

$$\mathbf{N} = \begin{bmatrix} \zeta_1 & \zeta_2 & \zeta_3 \end{bmatrix} \tag{2.19}$$

Fig. 2.5 illustrates these three shape funtions.

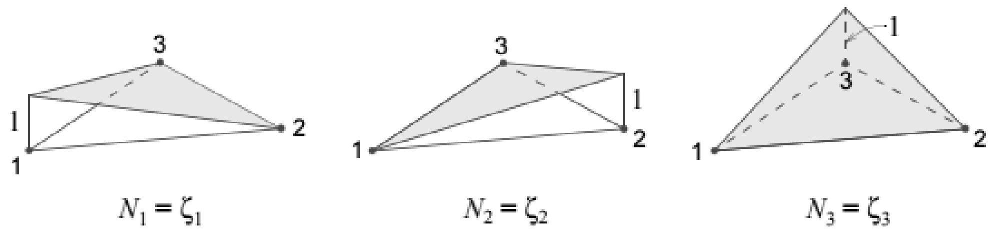


Figure 2.5: Linear shape functions N_1, N_2, and N_3 for the three-node triangular element, from Bell[2] p. 174

Eq. (2.18) established the strain-displacement matrix for triangular elements. Inserting the shape functions derived above we get

$$\boldsymbol{B} = \begin{bmatrix} \dfrac{\partial \boldsymbol{N}}{\partial x} \\ \dfrac{\partial \boldsymbol{N}}{\partial y} \end{bmatrix} = \frac{1}{2A} \begin{bmatrix} y_{23} & y_{31} & y_{12} \\ x_{32} & x_{13} & x_{21} \end{bmatrix} \tag{2.20}$$

We observe that $\boldsymbol{B}$ is constant over the element, with no dependence on $\boldsymbol{\zeta}$, making it a CST.

2.4.3 Six-node triangular element

The six-node triangular element has both corner nodes and mid-point nodes along the edges. As a consequence the derivation of its shape functions (and their derivatives) are more complex compared to its T3 counterpart. Again we will apply the "0-lines" method to obtain the shape functions. Fig. 2.6 illustrates this method used on a T6 element.

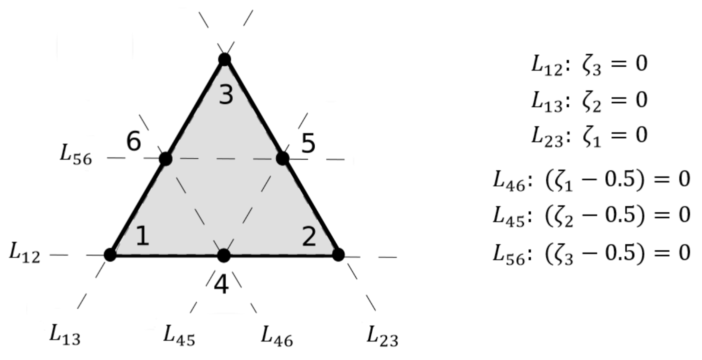


Figure 2.6: Construction of shape functions for T6 element by the "0-lines" method

We look at node number **1**:

1. Assume the shape function $N_1 = c_1\, L_{23}\, L_{46}$, as it is zero at every node except **1**
2. $\implies N_1 = c_1\, \zeta_1\, (\zeta_1 - 0.5)$
3. Fulfill unity requirement: $N_1(\zeta_1 = 1, \zeta_2 = 0, \zeta_3 = 0) = c_1 \cdot 1 \cdot 0.5 = 1 \implies c_1 = 2$
4. $\implies N_1 = 2\, \zeta_1\, (\zeta_1 - 0.5) \implies N_1 = \zeta_1(2\zeta_1 - 1)$

The procedure is identical for the two other corner nodes. We now look at edge node number **4**:

1. Assume the shape function $N_4 = c_4\, L_{13}\, L_{23}$, as it is zero at every node except **4**
2. $\implies N_4 = c_4\, \zeta_1\, \zeta_2$
3. Fulfill unity requirement: $N_4(\zeta_1 = 0.5,\, \zeta_2 = 0.5,\, \zeta_3 = 0) = c_4 \cdot 0.5 \cdot 0.5 = 1 \implies c_4 = 4$
4. $\implies N_4 = 4\, \zeta_1\, \zeta_2$

The procedure is identical for the two other mid-point nodes. Both N_1 and N_4 are illustrated in Fig. 2.7. By finishing the procedures for the remaining four nodes we arrive at the following shape functions for the T6 element

$$\boldsymbol{N} = \begin{bmatrix} \zeta_1(2\zeta_1 - 1) & \zeta_2(2\zeta_2 - 1) & \zeta_3(2\zeta_3 - 1) & 4\zeta_1\zeta_2 & 4\zeta_2\zeta_3 & 4\zeta_1\zeta_3 \end{bmatrix} \tag{2.21}$$

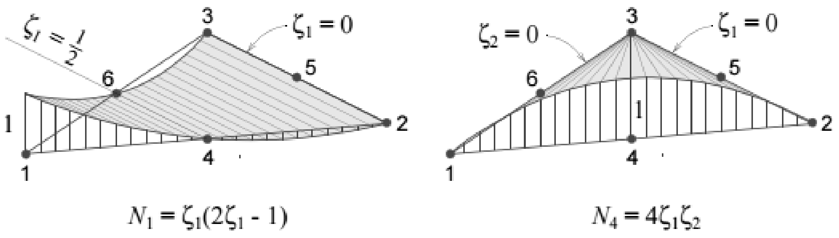


Figure 2.7: Quadratic shape functions N_1 and N_4 for the six-node triangular element, from Bell[2] p. 176

To determine the partial derivatives of **N** (i.e. $\boldsymbol{N},_x$ and $\boldsymbol{N},_y$) we make use of Eq. (2.18) to establish the strain-displacement matrix in Eq. (2.22). We observe that **B** varies linearly with respect to $\boldsymbol{\zeta}$ over the element, which makes it an LST.

$$\boldsymbol{B} = \begin{bmatrix} \dfrac{\partial \boldsymbol{N}}{\partial x} \\ \dfrac{\partial \boldsymbol{N}}{\partial y} \end{bmatrix} = \frac{1}{2A} \begin{bmatrix} y_{23} & y_{31} & y_{12} \\ x_{32} & x_{13} & x_{21} \end{bmatrix} \begin{bmatrix} (2\zeta_1 - 1) & 0 & 0 & 4\zeta_2 & 0 & 4\zeta_3 \\ 0 & (2\zeta_2 - 1) & 0 & 4\zeta_1 & 4\zeta_3 & 0 \\ 0 & 0 & (2\zeta_3 - 1) & 0 & 4\zeta_2 & 4\zeta_1 \end{bmatrix} \tag{2.22}$$

2.5 Quadrilateral elements

CCSC offers support for two quadrilateral elements. The Q4 element has a bilinear displacement field, making the strain field non-constant. The Q9 element on the other hand, has a quadratic displacement field, giving a fully linear strain field.

2.5.1 Natural coordinates interpolation

For the implementation of quadrilateral elements, we make use of the natural coordinate system. By normalizing curvilinear coordinates to the simplest possible geometry, it is much easier to perform mathematical operations. This is done by establishing relations between the physical coordinates (x, y) and the corresponding natural coordinates (ξ, η) in the normalized system. With this in mind, we can express the shape functions and their derivatives in terms of natural coordinates. The resulting element properties can then be transformed back to the physical coordinates. Fig. 2.8 illustrates the transformation back and forth between physical and natural coordinates.

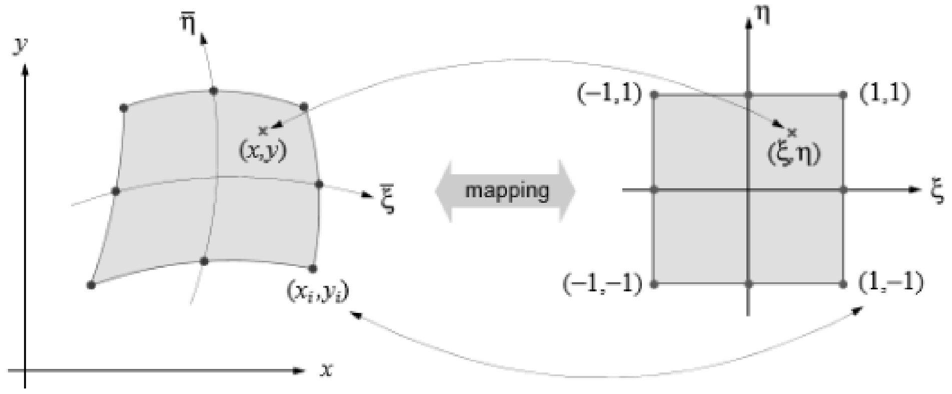


Figure 2.8: Mapping from physical coordinates (x, y) to natural coordinates (ξ, η) and vice versa, from Bell[2] p. 190

We will not establish a relationship between (ξ, η) and (x, y) in closed form, like we did for triangular elements. This is not possible for general geometries. The position of the nodal points define the geometric form of the element, thus it makes sense to express this geometry as an interpolation of the nodal points. That is

$$x(\xi,\eta) = \sum_i N_i x_i = \boldsymbol{N x} \tag{2.23a}$$

$$y(\xi,\eta) = \sum_i N_i y_i = \boldsymbol{N y} \tag{2.23b}$$

where **N** contains shape functions in the natural coordinate system (ξ,η), and **x** and **y** contains nodal coordinates in the physical reference system (x, y). The transformation between gradients of the two coordinate systems is given by the Jacobian matrix, which is defined as

$$\boldsymbol{J} = \frac{\partial(x,y)}{\partial(\xi,\eta)} = \begin{bmatrix} \dfrac{\partial x}{\partial \xi} & \dfrac{\partial y}{\partial \xi} \\ \dfrac{\partial x}{\partial \eta} & \dfrac{\partial y}{\partial \eta} \end{bmatrix} \tag{2.24}$$

Using Eq. (2.23) we can extend the previous equation to

$$\boldsymbol{J} = \begin{bmatrix} \dfrac{\partial \boldsymbol{N x}}{\partial \xi} & \dfrac{\partial \boldsymbol{N y}}{\partial \xi} \\ \dfrac{\partial \boldsymbol{N x}}{\partial \eta} & \dfrac{\partial \boldsymbol{N y}}{\partial \eta} \end{bmatrix} = \begin{bmatrix} J_{11} & J_{12} \\ J_{21} & J_{22} \end{bmatrix} \tag{2.25}$$

The determinant of the Jacobian matrix is then defined as

$$J = |\boldsymbol{J}| = \det(\boldsymbol{J}) = J_{11}J_{22} - J_{12}J_{21} \tag{2.26}$$

The inverse is given as

$$\boldsymbol{J}^{-1} = \frac{1}{J} \begin{bmatrix} J_{22} & -J_{12} \\ -J_{21} & J_{11} \end{bmatrix} \tag{2.27}$$

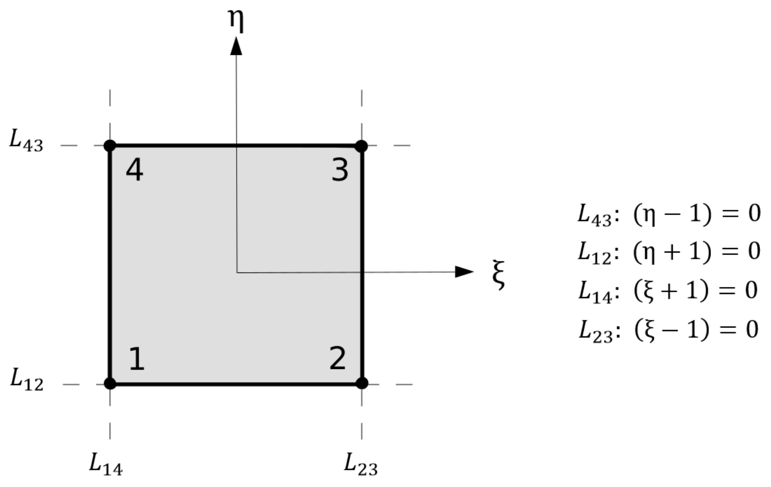
which is the transformation from (ξ,η) to (x, y). As with triangular elements, we must determine $\boldsymbol{B}$. By applying the *chain rule* we can easily establish its terms as

$$\boldsymbol{B} = \begin{bmatrix} \dfrac{\partial \boldsymbol{N}}{\partial x} \\ \dfrac{\partial \boldsymbol{N}}{\partial y} \end{bmatrix} = \begin{bmatrix} \dfrac{\partial \xi}{\partial x} & \dfrac{\partial \eta}{\partial x} \\ \dfrac{\partial \xi}{\partial y} & \dfrac{\partial \eta}{\partial y} \end{bmatrix} \begin{bmatrix} \dfrac{\partial \boldsymbol{N}}{\partial \xi} \\ \dfrac{\partial \boldsymbol{N}}{\partial \eta} \end{bmatrix} = \boldsymbol{J}^{-1} \begin{bmatrix} \dfrac{\partial \boldsymbol{N}}{\partial \xi} \\ \dfrac{\partial \boldsymbol{N}}{\partial \eta} \end{bmatrix} \tag{2.28}$$

We are now able to transform element gradients from the normalized system (ξ,η) back to the physical coordinate system (x, y). If the same shape functions are used to represent both the element geometry and the unknown field variable (in this case warping displacements), the element is defined as *isoparametric*. This is the case for the two quadrilateral elements.

2.5.2 Four-node quadrilateral element

The Q4 element is the simplest of the two quadrilateral elements supported in CCSC, containing only corner nodes. We make use of the now familiar "0-lines" method to establish the shape functions. Fig. 2.9 illustrates the use of this method for a Q4 element.



η
ξ
L_{43}
L_{12}
L_{14}
L_{23}
4
3
1
2
L_{43}: $(\eta - 1) = 0$
L_{12}: $(\eta + 1) = 0$
L_{14}: $(\xi + 1) = 0$
L_{23}: $(\xi - 1) = 0$

Figure 2.9: Construction of shape functions for Q4 element by the "0-lines" method

We now look to node number **1**:

1. Assume the shape function $N_1 = c_1\, L_{23}\, L_{43}$, as its function is zero at every node except **1**
2. $\implies N_1 = c_1\, (\xi - 1)(\eta - 1)$
3. Fulfill unity requirement: $N_1(\xi = -1, \eta = -1) = c_1 \cdot (-2)(-2) = 1 \implies c_1 = \frac{1}{4}$
4. $\implies N_1 = \frac{1}{4}(1 - \xi)(1 - \eta)$

The procedure is identical for the three other corner nodes, and their shape functions need no further explanation. The four shape functions are given in Eq. (2.29).

$$\boldsymbol{N}^T = \frac{1}{4}\begin{bmatrix}(1-\xi)(1-\eta)\\(1+\xi)(1-\eta)\\(1+\xi)(1+\eta)\\(1-\xi)(1+\eta)\end{bmatrix} \tag{2.29}$$

Fig. 2.10 illustrates the bilinear shape functions of $\boldsymbol{N}$, while Eq. (2.30) gives the partial derivatives ($\boldsymbol{N},_\xi$ and $\boldsymbol{N},_\eta$) of the four-node quadrilateral element.

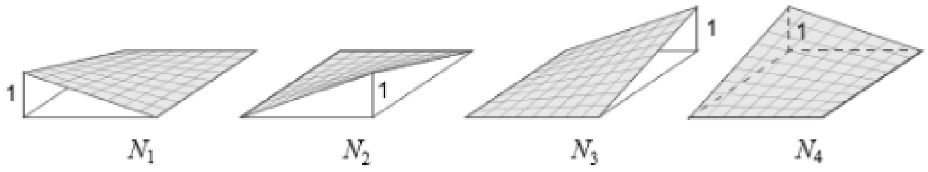


Figure 2.10: Shape functions N_1 through N_4 for the four-node quadrilateral element, from Bell[2] p. 170

$$\frac{\partial \boldsymbol{N^T}}{\partial \xi} = \boldsymbol{N},_\xi^T = \frac{1}{4}\begin{bmatrix}(-1+\eta)\\(1-\eta)\\(1+\eta)\\(-1-\eta)\end{bmatrix} \qquad \frac{\partial \boldsymbol{N^T}}{\partial \eta} = \boldsymbol{N},_\eta^T = \frac{1}{4}\begin{bmatrix}(-1+\xi)\\(-1-\xi)\\(1+\xi)\\(1-\xi)\end{bmatrix} \tag{2.30}$$

By using Eq. (2.28) we establish $\boldsymbol{B}$ as

$$\boldsymbol{B} = \frac{1}{4}\boldsymbol{J}^{-1}\begin{bmatrix}-(1-\eta) & (1-\eta) & (1+\eta) & -(1+\eta)\\-(1-\xi) & -(1+\xi) & (1+\xi) & (1-\xi)\end{bmatrix} \tag{2.31}$$

We observe $\boldsymbol{B}$ to be non-constant unlike the linear triangular element. Additionally we observe $\boldsymbol{N},_\xi^T$ to be linearly dependent on η but independent of ξ. This means that γ_{xz} (and therefore τ_{xz}) is independent of ξ. Applying the same idea for $\boldsymbol{N},_\eta^T$ concludes that γ_{yz} (and therefore τ_{yz}) is independent of η. This is investigated further in Sec. 6.3.4.

2.5.3 Nine-node quadrilateral element

The Q9 element contains three different kinds of nodes: corner nodes, mid-point nodes along the edges, and a node at the center of the element. Each of these three cases will be investigated individually, using the "0-lines" method. Fig. 2.11 illustrates the method used on a Q9 element.

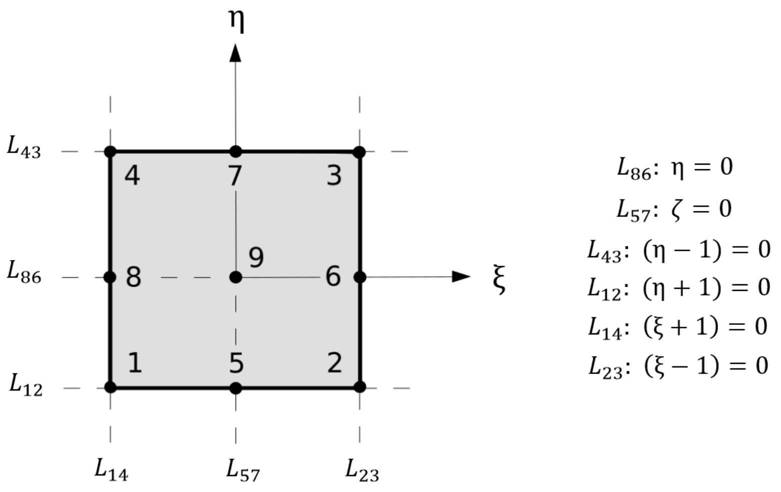


Figure 2.11: Construction of shape functions for Q9 element by the "0-lines" method

First, we look at corner node number **1**:

1. Assume the shape function $N_1 = c_1\, L_{23}\, L_{43}\, L_{57}\, L_{86}$, as it is zero at every node except **1**
2. $\implies N_1 = c_1\, \xi\eta(\xi-1)(\eta-1)$
3. Fulfill unity requirement: $N_1(\xi=-1, \eta=-1) = c_1 \cdot (-1)(-1)(-2)(-2) = 1 \implies c_1 = \frac{1}{4}$.
4. $\implies N_1 = \frac{1}{4}\xi\eta(\xi-1)\eta-1)$.

The procedure is identical for the three other corner nodes. Now, we investigate edge node number **5**:

1. Assume the shape function $N_5 = c_5\, L_{14}\, L_{23}\, L_{43}\, L_{86}$, as it is zero at every node except **5**
2. $\implies N_5 = c_5\, (\xi+1)(\xi-1)(\eta-1)\eta$
3. Fulfill unity requirement: $N_5(\xi=0, \eta=-1) = c_5 \cdot (1)(-1)(-2)(-1) = 1 \implies c_5 = -\frac{1}{2}$

4. $\implies N_5 = -\frac{1}{2}\eta(\eta-1)(\xi^2-1)$

The procedure is identical for the three other edge nodes. Finally, we investigate center node number **9**:

1. Assume the shape function $N_9 = c_9\, L_{14}\, L_{23}\, L_{43}\, L_{12}$, as it is zero at every node except **9**
2. $\implies N_9 = c_9(\xi+1)(\xi-1)(\eta-1)(\eta+1)$
3. Fulfill unity requirement: $N_9(\xi=0,\, \eta=0) = c_9 \cdot (1)(-1)(-1)(1) = 1 \implies c_9 = 1$
4. $\implies N_9 = (\xi^2-1)(\eta^2-1)$

By finishing this procedure for the remaining six nodes we establish $\boldsymbol{N}$ for the Q9 element as given in Eq. (2.32). Fig. 2.12 illustrates the examined shape functions (N_1,N_5, and N_9), while Eq. (2.33) gives the partial derivatives ($\boldsymbol{N}_{,\xi}$ and $\boldsymbol{N}_{,\eta}$) of the nine-node quadrilateral element.

$$\boldsymbol{N}^T = \frac{1}{4}\begin{bmatrix} \xi\eta(\xi-1)(\eta-1) \\ \xi\eta(\xi+1)(\eta-1) \\ \xi\eta(\xi+1)(\eta+1) \\ \xi\eta(\xi-1)(\eta+1) \\ -2\eta(\eta-1)(\xi^2-1) \\ -2\xi(\xi+1)(\eta^2-1) \\ -2\eta(\eta+1)(\xi^2-1) \\ -2\xi(\xi-1)(\eta^2-1) \\ 4(\xi^2-1)(\eta^2-1) \end{bmatrix} \tag{2.32}$$

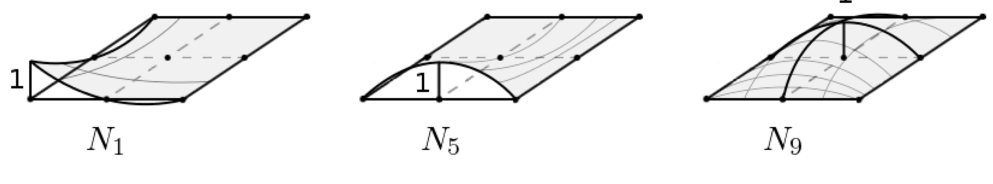


Figure 2.12: Biquadratic shape functions N_1, N_5, and N_9 for the nine-node quadrilateral element, from Bell[2] p. 170

$$\frac{\partial \boldsymbol{N^T}}{\partial \xi} = \boldsymbol{N},_{\xi}^{T} = \frac{1}{4}\begin{bmatrix} \eta(\eta-1)(2\xi-1) \\ \eta(\eta-1)(2\xi+1) \\ \eta(\eta+1)(2\xi+1) \\ \eta(\eta+1)(2\xi-1) \\ -4\xi\eta(\eta-1) \\ -2(\eta^2-1)(2\xi+1) \\ -4\xi\eta(\eta+1) \\ -2(\eta^2-1)(2\xi-1) \\ 8\xi(\eta^2-1) \end{bmatrix} \qquad \frac{\partial \boldsymbol{N^T}}{\partial \eta} = \boldsymbol{N},_{\eta}^{T} = \frac{1}{4}\begin{bmatrix} \xi(\xi-1)(2\eta-1) \\ \xi(\xi+1)(2\eta-1) \\ \xi(\xi+1)(2\eta+1) \\ \xi(\xi+1)(2\eta+1) \\ -2(\xi^2-1)(2\eta-1) \\ -4\xi\eta(\xi+1) \\ -2(\xi^2-1)(2\eta+1) \\ -4\xi\eta(\xi-1) \\ 8\eta(\xi^2-1) \end{bmatrix} \tag{2.33}$$

We observe $\boldsymbol{N},_{\xi}^{T}$ to be quadratically dependent on η and linearly dependent on ξ, while $\boldsymbol{N},_{\eta}^{T}$ is quadratically dependent on ξ and linearly dependent on η. In other words the element has a linear strain-displacement field. By using Eq. (2.28) we establish $\boldsymbol{B}$ as

$$\boldsymbol{B} = \frac{1}{4}\boldsymbol{J}^{-1}\begin{bmatrix} \boldsymbol{N},_{\xi} \\ \boldsymbol{N},_{\eta} \end{bmatrix} \tag{2.34}$$

2.6 Numerical integration

In order to determine the element stiffness matrix $\boldsymbol{k_e}$ and the load vector $\boldsymbol{S}_0$, we must evaluate integrals on the form

$$I_t = \int_A f(x,y)dA = \int_A f(\zeta_1,\zeta_2,\zeta_3)dA \tag{2.35}$$

for triangular elements, and

$$I_q = \int_A f(x,y)dA = \int_{-1}^{1}\int_{-1}^{1} f(\xi,\eta)J(\xi,\eta)d\xi d\eta \tag{2.36}$$

for quadrilateral elements. In many cases these integrals are very difficult, or even impossible, to evaluate analytically. Therefore we make use of numerical integration in the form of Gaussian quadrature. Instead of evaluating the integral, Gaussian quadrature takes the sum of the function value at a number of integration points, where each integration point has an assigned weight value. The next two sections elaborate on this. The evaluation of the stiffness matrix is used as an example as the integrand contains the highest polynomial order present in our massive analyses.

2.6.1 Triangular elements

For triangular elements the element stiffness matrix is of the form

$$\boldsymbol{k_e} = \int_V \boldsymbol{B}^T(\zeta_1,\zeta_2,\zeta_3)\, \boldsymbol{GB}\,(\zeta_1,\zeta_2,\zeta_3) dV = \int_A t(\zeta_1,\zeta_2,\zeta_3) f(\zeta_1,\zeta_2,\zeta_3)\, dA \tag{2.37}$$

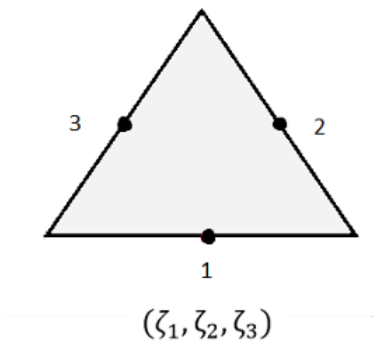
which implies an integration over the element, with both **B** and t (thickness) varying with respect to $\boldsymbol{\zeta}$. However, we define t to be equal to unity over the element. By using Gaussian quadrature for this integral we get

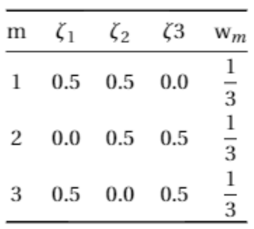
$$I = \int_A f(\zeta_1,\zeta_2,\zeta_3)\, dA \approx A \sum_{i=1}^{n} w_i f(\zeta_{1i},\zeta_{2i},\zeta_{3i}) \tag{2.38}$$

If we were not using straight edge triangles, we would multiply the expression with the Jacobian determinant rather than the area. We establish the stiffness matrix as

$$\boldsymbol{k_e} \approx A \sum_{i=1}^{n} w_i \boldsymbol{B}^T(\zeta_{1i},\zeta_{2i},\zeta_{3i}) \boldsymbol{GB}(\zeta_{1i},\zeta_{2i},\zeta_{3i}) \tag{2.39}$$

With n integration points we are able to evaluate the exact value of an integral of order *2n-1*. The integrand for T3 elements is constant while the integrand for T6 elements contains the terms ζ_1^2, ζ_2^2, and ζ_3^2. With one integration point we can only evaluate 1st order polynomials. By using three integration points we can evaluate the exact integral up to the 5th order. This exceeds the polynomial degree for both T3 and T6 elements. CCSC therefore makes use of three integration points for triangular elements. The ability to evaluate integrals exactly is called *full integration*[2]. The three integration points are located at the mid-point of each element edge. An illustration of the integration points and their assigned weights is given in Fig. 2.13.





m	ζ_1	ζ_2	$\zeta 3$	w_m
1	0.5	0.5	0.0	$\frac{1}{3}$
2	0.0	0.5	0.5	$\frac{1}{3}$
3	0.5	0.0	0.5	$\frac{1}{3}$

Figure 2.13: The three integration points used for triangular elements

2.6.2 Quadrilateral elements

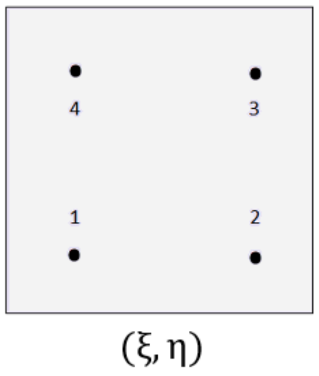
For the quadrilateral element types, the element stiffness matrix is on the form

$$\boldsymbol{k}_e = \int_V \boldsymbol{B}^T(\xi,\eta)\, \boldsymbol{GB}\,(\xi,\eta) dV = \int_A t(\xi,\eta) f(\xi,\eta)\, dA \tag{2.40}$$

We define t to be equal to unity over the element. Using Gaussian quadrature for this integral gives

$$\begin{aligned} \boldsymbol{k}_e &= \int_{-1}^{1}\int_{-1}^{1} \boldsymbol{B}^T(\xi,\eta)\, \boldsymbol{GB}\,(\xi,\eta) J(\xi,\eta) d\xi d\eta \\ &\approx \sum_{i=1}^{1}\sum_{j=1}^{1} \boldsymbol{B}^T(\xi_i,\eta_j)\, \boldsymbol{GB}\,(\xi_i,\eta_j) J(\xi_i,\eta_j) w_{ij} d\xi d\eta \end{aligned} \tag{2.41}$$

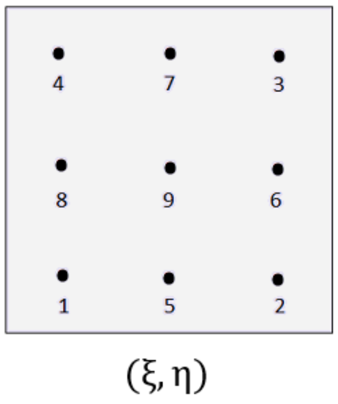
For Q4 elements, the stiffness matrix $\boldsymbol{B}$ contains terms of ξ and η, leaving the integrand with terms of ξ^2 and η^2. A 2x2 integration scheme evaluates integrals of up to 3rd order exactly, thus we achieve full integration with this rule. Fig. 2.14 illustrates these integration points and their assigned weights.



m	ξ_m	η_m	w_m
1	$-\frac{1}{\sqrt{3}}$	$-\frac{1}{\sqrt{3}}$	1.0
2	$\frac{1}{\sqrt{3}}$	$-\frac{1}{\sqrt{3}}$	1.0
3	$\frac{1}{\sqrt{3}}$	$-\frac{1}{\sqrt{3}}$	1.0
4	$-\frac{1}{\sqrt{3}}$	$-\frac{1}{\sqrt{3}}$	1.0

Figure 2.14: 2x2 Gaussian quadrature integration scheme. Evaluates integrals of up to order three exactly

For Q9 elements, $\boldsymbol{B}$ contains the term $\xi^2\eta^2$. The highest order polynomial in the integrand is then $\xi^4\eta^4$. The 2x2 integration scheme no longer provides full integration, and we have to use a 3x3 integration scheme instead. Fig. 2.15 illustrates the integration points and their weights for the 3x3 integration scheme.



m	ξ_m	η_m	w_m
1	$-\sqrt{0.6}$	$-\sqrt{0.6}$	$\frac{25}{81}$
2	$\sqrt{0.6}$	$-\sqrt{0.6}$	$\frac{25}{81}$
3	$\sqrt{0.6}$	$\sqrt{0.6}$	$\frac{25}{81}$
4	$-\sqrt{0.6}$	$\sqrt{0.6}$	$\frac{25}{81}$
5	0	$-\sqrt{0.6}$	$\frac{40}{81}$
6	$\sqrt{0.6}$	0	$\frac{40}{81}$
7	0	$\sqrt{0.6}$	$\frac{40}{81}$
8	$-\sqrt{0.6}$	0	$\frac{40}{81}$
9	0	0	$\frac{64}{81}$

Figure 2.15: 3x3 Gaussian quadrature integration scheme. Evaluates integrals of up to 5th order exactly

For both Q4 and Q9 elements we achieve a higher integration than actually needed to obtain exact results. This means that the thickness does not necessarily have to be constant over the cross-section. The integrals would be evaluated exactly even with a bilinear variation of thickness.

2.6.3 Reduced integration

As mentioned in the previous sections we have chosen a three-point Gaussian scheme for triangular elements, and a 2x2 and 3x3 Gaussian quadrature scheme for Q4 and Q9 elements respectively. This is enough to provide full integration, which can help prevent certain instabilities such as spurious modes, kinematic modes, singular modes, hourglass modes, and/or zero energy modes[2]. These modes can often occur when utilizing *reduced integration*. Reduced integration is when the integration uses fewer integration points than what is needed in order to obtain exact results.

Although reduced integration gives a reduction in accuracy, for some cases it is recommended over full integration. The main advantage of using reduced integration is a significantly reduced workload, especially for large, time-consuming analyses. Additionally, reduced integration leads to less stiff elements, which may lead to a closer approximation to real world problems. In other words the loss in accuracy is counteracted by a closer approximation to the real world[10]. We have as of yet not implemented options for reduced integration in CCSC, and it is left for future development.

2.7 The equilibrium equation

With the stiffness matrix $\boldsymbol{k}_e$ and load vector $\boldsymbol{S}_{\boldsymbol{e}}^{\boldsymbol{0}}$ established at the element level in Eqs. (2.9 and 2.10), we sum them up into a global stiffness matrix $\boldsymbol{K}$ and a global load vector $\boldsymbol{R}$ as

$$\boldsymbol{K} = \sum_{e=1}^{n} \boldsymbol{k}_e \tag{2.42a}$$

$$\boldsymbol{R} = \pm \sum_{e=1}^{n} \boldsymbol{S}_e^0 \tag{2.42b}$$

The sign of the element load vector is dependent on which analysis is performed. It is negative for torsional loading and positive for shear loading. This is clarified in Sections 2.9 and 2.10. Now we are able to solve the problem for the nodal displacements $\boldsymbol{r}$ using

$$\boldsymbol{K}\boldsymbol{r} = \boldsymbol{R} \tag{2.43}$$

For the boundary conditions it is sufficient to constrain one arbitrary node i (Section 4.3 describes the implementation). When the global displacements are found, we also have the element displacements. They are stored in $\boldsymbol{v}$, which contains the nodal values of the warping function $\Psi(x, y)$ for an element with n nodes.

$$\boldsymbol{v}^T = \begin{bmatrix} \Psi_1 & \Psi_2 & \cdots & \Psi_n \end{bmatrix} \tag{2.44}$$

Using the element displacements, the stresses due to torsional and shear loading can be found as described in Sections 2.9 and 2.10.

2.8 Bending moment analysis

The calculations of axial stresses for a cross-section subjected to bending moment loading are only dependent on the second moment of area (found in Appendix A.2). They are given as

$$\sigma_x = \frac{M_x}{I_x} \cdot y \tag{2.45a}$$

$$\sigma_y = -\frac{M_y}{I_y} \cdot x \tag{2.45b}$$

where x and y are the distances between the node and the area center.

2.9 Elastic torsion analysis

In Fig. 2.16 a prismatic beam with a massive cross-section and length L is illustrated. It is subjected to a constant torsional moment M_z causing the cross-section to rotate ϕ_L at $z = L$. The following derivation of torsion analysis is a brief summary of the theory explained in the documentation for CrossX[3], and it is based on an orientation of the cross-section about the principal axis.

The torsion analysis is based on the following assumptions:

- the material is linearly elastic, continuous, isotropic, and constant within the element. The properties can, however, change from one element to the next

- the cross-section is assumed to rotate as a rigid body about the shear center as described in Eq. (2.46)
- warping w is proportional to the *rate of twist* θ, and independent of z.

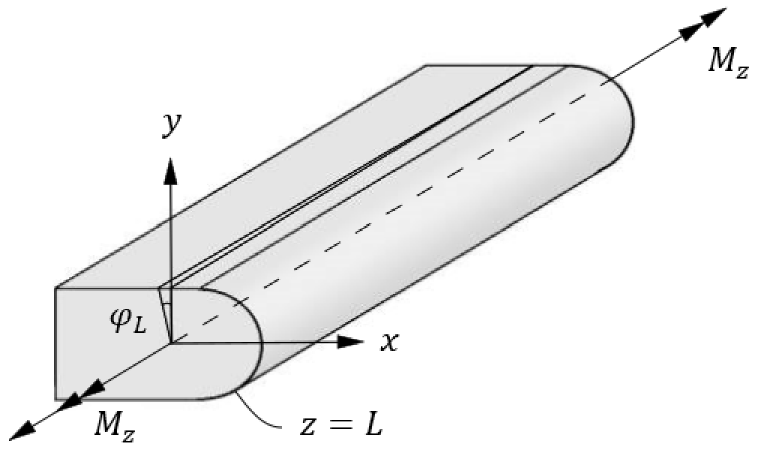


Figure 2.16: Uniform torsion of a prismatic bar with an arbitrary massive cross-section

2.9.1 St. Venant torsion

Displacements due to *St. Venant torsion*[11] are given as

$$\begin{aligned} u &= -\theta yz \\ v &= \theta xz \\ w &= \theta \Psi(x, y) \end{aligned} \tag{2.46}$$

where $\theta = \frac{d\phi}{dz}$ is the rate of twist (or twist per length) and $\phi(z)$ is the angle of twist. $\Psi(x, y)$ is the (unknown) warping function and w is the warping displacement. It is straight-forward using Hooke's law and Eq. (2.46) to show that $\epsilon_x = \epsilon_y = \epsilon_z = \gamma_{xy} = 0$ and

$$\begin{aligned} \gamma_{xz} &= w_{,x} + u_{,z} = \Psi_{,x} - \theta y = \frac{1}{G}\tau_{xz} \\ \gamma_{yz} &= w_{,y} + u_{,z} = \Psi_{,y} - \theta x = \frac{1}{G}\tau_{yz} \end{aligned} \tag{2.47}$$

We now repeat the fundamental assumption that Ψ can be expressed in terms of nodal point values at any point in the element. That is

$$\Psi = \boldsymbol{N}\boldsymbol{v} \tag{2.48}$$

with $\boldsymbol{N} = \left[N_1\ N_2\ ...\ N_n\right]$ as the interpolation functions for an element with n nodes. Eq. (2.47) can then be written on matrix form as

$$\boldsymbol{\epsilon} = \begin{bmatrix} \gamma_{xz} \\ \gamma_{yz} \end{bmatrix} = \left(\begin{bmatrix} \Psi_{,x} \\ \Psi_{,y} \end{bmatrix} + \begin{bmatrix} -\theta y \\ \theta x \end{bmatrix} \right) \tag{2.49}$$

The coordinates x and y can be expressed in terms of the element coordinates as

$$x = \boldsymbol{N}\boldsymbol{x} \quad \text{and} \quad y = \boldsymbol{N}\boldsymbol{y} \tag{2.50}$$

For an element with a displacement vector $\boldsymbol{v}$, strain-displacement matrix $\boldsymbol{B}$, and initial strain vector $\boldsymbol{\epsilon}_0$, we can rewrite Eq. (2.49) as

$$\boldsymbol{\epsilon} = \boldsymbol{B}\boldsymbol{v} + \boldsymbol{\epsilon}_0 \tag{2.51}$$

where

$$\boldsymbol{v}^T = \begin{bmatrix} \Psi_1 & \Psi_2 & \cdots & \Psi_n \end{bmatrix} \tag{2.52a}$$

$$\boldsymbol{B} = \begin{bmatrix} \boldsymbol{N}_{,x} \\ \boldsymbol{N}_{,y} \end{bmatrix} \tag{2.52b}$$

$$\boldsymbol{\epsilon}_0 = \begin{bmatrix} \boldsymbol{0} & -\theta\boldsymbol{N} \\ \theta\boldsymbol{N} & \boldsymbol{0} \end{bmatrix} \begin{bmatrix} \boldsymbol{x} \\ \boldsymbol{y} \end{bmatrix} \tag{2.52c}$$

Stresses are simply given as

$$\boldsymbol{\tau} = \boldsymbol{G}\boldsymbol{\epsilon} = \boldsymbol{G}\left(\boldsymbol{B}\boldsymbol{v} + \boldsymbol{\epsilon}_0\right) \tag{2.53}$$

The contributions to the nodal forces for a whole element are given as

$$\boldsymbol{S} = \int_{A_e} \boldsymbol{B}^T \boldsymbol{\tau}\, dA = \int_{A_e} \boldsymbol{B}^T \boldsymbol{G}\boldsymbol{B}\, dA\, \boldsymbol{v} + \int_{A_e} \boldsymbol{B}^T \boldsymbol{G}\boldsymbol{\epsilon}_0\, dA = \boldsymbol{k}_e \boldsymbol{v} + \boldsymbol{S}_e^0 \tag{2.54}$$

In the previous equation we recognize both the stiffness matrix $\boldsymbol{k}_e$ and the element load $\boldsymbol{S}_e^0$ to be defined identical to what was established in Eqs. (2.9 and 2.11a). The warping displacement w is proportional to the rate of twist θ at any point in the the cross-section (see Eq. (2.46)). For this reason will use the following procedure to calculate the shear stresses due to torsion:

1. compute the nodal displacements $\boldsymbol{v}$ for θ equal to unity (with $\boldsymbol{v}$ as the only unknown in Eq. (2.54))
2. compute θ based on the computed nodal displacements $\boldsymbol{v}$ and applied moment M_z
3. compute stresses according to the warping displacements $w = \theta\Psi(x, y)$

The first step involves producing the global stiffness matrix **K** and load vector **R** and solve the equilibrium equation of (2.43). This was described in Section 2.7 and is not repeated here. We only clarify that Eq. (2.54) requires the global load vector for torsional analysis on the form

$$\boldsymbol{R} = -\sum_{e=1}^{n} \boldsymbol{S}_e^0 \tag{2.55}$$

In order to determine θ (step two) we begin by expressing the torsional moment as

$$M_z = \int_A (\tau_{yz}x - \tau_{xz}y)dA = \int_A \begin{bmatrix} y & x \end{bmatrix} \begin{bmatrix} -\tau_{xz} \\ \tau_{yz} \end{bmatrix} dA \tag{2.56}$$

Referring to Eq. (2.52), the contribution to M_z from one element is

$$\Delta M_z = G\int_A \begin{bmatrix} \boldsymbol{x^T} \\ \boldsymbol{y^T} \end{bmatrix} \begin{bmatrix} \boldsymbol{0} & \boldsymbol{N}^T \\ \boldsymbol{N}^T & \boldsymbol{0} \end{bmatrix} \left(\begin{bmatrix} -\boldsymbol{N}_{,x} \\ \boldsymbol{N}_{,y} \end{bmatrix} \theta\boldsymbol{v} + \begin{bmatrix} \boldsymbol{0} & \boldsymbol{N} \\ \boldsymbol{N} & \boldsymbol{0} \end{bmatrix} \begin{bmatrix} \boldsymbol{x} \\ \boldsymbol{y} \end{bmatrix} \theta \right) \tag{2.57}$$

We introduce torsional stiffness GI_t for one element as

$$\Delta M_z = \theta(\Delta GI_t) \tag{2.58}$$

which means the torsional stiffness increment is

$$\Delta GI_t = G\int_A \begin{bmatrix} \boldsymbol{x^T} \\ \boldsymbol{y^T} \end{bmatrix} \begin{bmatrix} \boldsymbol{0} & \boldsymbol{N}^T \\ \boldsymbol{N}^T & \boldsymbol{0} \end{bmatrix} \left(\begin{bmatrix} -\boldsymbol{N}_{,x} \\ \boldsymbol{N}_{,y} \end{bmatrix} \boldsymbol{v} + \begin{bmatrix} \boldsymbol{0} & \boldsymbol{N} \\ \boldsymbol{N} & \boldsymbol{0} \end{bmatrix} \begin{bmatrix} \boldsymbol{x} \\ \boldsymbol{y} \end{bmatrix} \right) \tag{2.59}$$

$$GI_t = \sum \Delta GI_t = \frac{M_z}{\theta} \tag{2.60}$$

The analysis proceeds by computing the torsional stiffness GI_t. Knowing the applied load M_z we can determine θ. Finally, according to step three, the shear stresses are computed for each element by Eq. (2.61)

$$\boldsymbol{\sigma} = \begin{bmatrix} \tau_{xz} \\ \tau_{yz} \end{bmatrix} = \boldsymbol{G}(\boldsymbol{B}\theta\boldsymbol{v} + \boldsymbol{\epsilon}_0) = \theta\boldsymbol{G}\left(\begin{bmatrix} \boldsymbol{N}_{,x} \\ \boldsymbol{N}_{,y} \end{bmatrix} \boldsymbol{v} + \begin{bmatrix} \boldsymbol{0} & -\boldsymbol{N} \\ \boldsymbol{N} & \boldsymbol{0} \end{bmatrix} \begin{bmatrix} \boldsymbol{x} \\ \boldsymbol{y} \end{bmatrix} \right) \tag{2.61}$$

2.9.2 Shear center from torsion

We refer to Meek[12] for the formulas for shear center due to torsion (CrossX[3] also provides a similar analysis). The shear center is given as

$$x_s = -\frac{\int_A E(x,y)\, y\Psi(x,y)\, dA}{EI_x} \tag{2.62a}$$

$$y_s = \frac{\int_A E(x,y)\, x\Psi(x,y)\, dA}{EI_y} \tag{2.62b}$$

Using Eqs. (2.48 and 2.50) we can rewrite Eq. (2.62) to finite element terms:

$$x_s = -\frac{\sum E_e \boldsymbol{y}^T \int_{A_e} \boldsymbol{N}^T \boldsymbol{N} dA\, \boldsymbol{v}}{EI_x} \tag{2.63a}$$

$$y_s = \frac{\sum E_e \boldsymbol{x}^T \int_{A_e} \boldsymbol{N}^T \boldsymbol{N} dA\, \boldsymbol{v}}{EI_y} \tag{2.63b}$$

2.10 Elastic shear analysis

Determining the shear stress distribution, shear center, and the transverse deflection due to transverse shear is not a straight-forward task for beams with complex cross-sections. The theory we present follows Bell[3], which in turn is based on the work of Mason and Herrmann[13]. The main assumption for this theory is that the relationship between stress and stress resultants and between curvature changes and stress resultants obtained for simple loading cases, approximately hold for all loading systems.

For shear analysis we will simply present the formulas needed for an implementation. The theory is clearly presented in Bell[3], and we found it unnecessary to reproduce it here. The case we consider is a prismatic cantilever with an arbitrary massive cross-section subjected to transverse shear loading, as illustrated in Fig. 2.17. The x- and y-axes are assumed to be principal to simplify expressions. We do not lose generality from doing this, because the orientation can always be determined from complementary analysis.

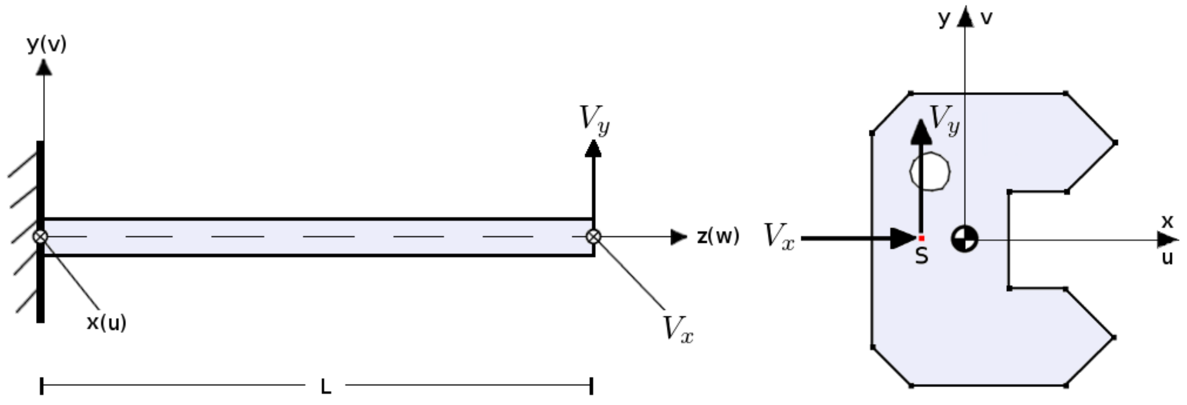


Figure 2.17: Cross-section with applied shear forces in the shear center (S)

Mason and Herrmann made the following assumptions:

- the cross-section is assumed to be uniform over the entire length of the beam
- the z-axis coincides with the cross-section centroid
- the material is linearly elastic, continuous, and isotropic
- body forces are small compared to the stresses and can be neglected
- for the constant shear case we have $\sigma_y = \sigma_z = \tau_{xy} = \tau_{yx} = 0$
- the normal stress σ_z is distributed as in the case of pure bending and the end loads V_x and V_y are applied in the shear center to eliminate torsion

2.10.1 Computing the stresses

Axial stress is independent of the nodal displacements from warping and is simply computed as

$$\sigma_z = E(L - z)(C_x x + C_y y) \tag{2.64}$$

for $(L-z)=1$. For the calculation of shear stress we establish the element load vector as

$$\boldsymbol{S}^0 = \boldsymbol{S}_x^0 + \boldsymbol{S}_y^0 \tag{2.65}$$

with components in x- and y-directions

$$\boldsymbol{S}_x^0 = \frac{E_e V_x}{EI_y}\left[\int_{A_e} \boldsymbol{N}^T x dA + \frac{\nu_e}{4(1+\nu_e)}\int_{A_e}(\boldsymbol{N},_x^T(x^2-y^2) + 2\boldsymbol{N},_y^T xy)dA\right] \tag{2.66a}$$

$$\boldsymbol{S}_y^0 = \frac{E_e V_y}{EI_x}\left[\int_{A_e} \boldsymbol{N}^T y dA + \frac{\nu_e}{4(1+\nu_e)}\int_{A_e}(\boldsymbol{N},_y^T(y^2-x^2) + 2\boldsymbol{N},_x^T xy)dA\right] \tag{2.66b}$$

The nodal displacements of the cross-section can now be determined for the two load cases as described in Section 2.7. For this case we will determine the nodal displacements due to the two shear loads separately. This is done by establishing

$$\boldsymbol{R}_x = \sum_{e=1}^{n} \boldsymbol{S}_x^0 \qquad \boldsymbol{R}_y = \sum_{e=1}^{n} \boldsymbol{S}_y^0 \tag{2.67}$$

and solving the equilibrium equation (2.43) for the two loads separately. Notice the positive sign for the summation of element load vectors, which differs from the torsional analysis. The stiffness matrix is the same as for torsional analysis and is defined by Eq. (2.11a). With the displacements found, we compute the shear stresses as

$$\begin{bmatrix}\tau_{xz}\\ \tau_{yz}\end{bmatrix} = \boldsymbol{G}(\boldsymbol{B}\boldsymbol{v} + \boldsymbol{\epsilon}_0) \tag{2.68}$$

where

$$\boldsymbol{\epsilon}_0 = \begin{bmatrix}-\nu\left(\frac{1}{2}C_x(x^2-y^2) + C_y xy\right)\\ -\nu\left(C_x xy + \frac{1}{2}C_y(y^2-x^2)\right)\end{bmatrix} \tag{2.69}$$

and

$$C_x = \frac{V_x}{EI_y} \qquad C_y = \frac{V_y}{EI_x} \tag{2.70}$$

2.10.2 Shear deformation factors

After determining the element displacements $\boldsymbol{v}$ due to shear loading in both the x- and y-direction, we can compute the shear deformation factor contributions κ_x^e and κ_y^e from each element as

$$\kappa_x^e = \frac{AG}{V_x EI_y}\left(E_e \boldsymbol{x}^T \boldsymbol{a}\boldsymbol{v}_x\right) - \frac{\nu_e}{4(1+\nu)}\frac{E_e A_e}{EA}\frac{(EI_y - EI_x)}{EI_y} \tag{2.71a}$$

$$\kappa_y^e = \frac{AG}{V_y EI_x}\left(E_e \boldsymbol{y}^T \boldsymbol{a}\boldsymbol{v}_y\right) - \frac{\nu_e}{4(1+\nu)}\frac{E_e A_e}{EA}\frac{(EI_x - EI_y)}{EI_x} \tag{2.71b}$$

where

$$\boldsymbol{a} = \int_{A_e} \boldsymbol{N}^T \boldsymbol{N} dA \tag{2.72}$$

Once the shear deformation factors are computed for each element, they are summed up to the global shear deformation factors

$$\kappa_x = \sum_{e=1}^{n} \kappa_x^e \qquad \kappa_y = \sum_{e=1}^{n} \kappa_y^e \tag{2.73}$$

2.11 Stress analysis for composite materials

We now present a general procedure for analysis by unifying torsion and shear loading. This is the first step of developing functionality for composite materials. The theory is not complete, and will need further work before CCSC can offer this functionality. We also recognize that many of the steps are familiar to the derivation of the torsional analysis from Section 2.9, and it is therefore somewhat shortened.

2.11.1 Governing equations

The total potential energy of a system is given by

$$\Pi = U - W \tag{2.74}$$

The strain energy U for the beam is given as

$$U = \frac{1}{2}\int_V \boldsymbol{\epsilon}^T \boldsymbol{C}\boldsymbol{\epsilon} dV \tag{2.75}$$

Strain is defined as

$$\boldsymbol{\epsilon} = \begin{bmatrix} \epsilon_z \\ \gamma_{xz} \\ \gamma_{yz} \end{bmatrix} \tag{2.76}$$

which can be expressed as

$$\boldsymbol{\epsilon} = \boldsymbol{B}\boldsymbol{r} + \boldsymbol{\epsilon}_0 \tag{2.77}$$

where the strain-displacement matrix and the warping displacements are defined as

$$\boldsymbol{B} = \begin{bmatrix} \boldsymbol{0} \\ \boldsymbol{N}_{,x} \\ \boldsymbol{N}_{,y} \end{bmatrix} \qquad \boldsymbol{r} = \begin{bmatrix} \Psi_1 \\ \vdots \\ \Psi_n \end{bmatrix} \tag{2.78}$$

for a mesh with n nodes. The initial strain $\boldsymbol{\epsilon}_0$ is dependent on which analysis is performed and for torsional loading it was established in Eq. (2.52c) as

$$\boldsymbol{\epsilon}_{0,t} = \begin{bmatrix} \boldsymbol{0} & \boldsymbol{0} \\ \boldsymbol{0} & -\boldsymbol{N} \\ \boldsymbol{N} & \boldsymbol{0} \end{bmatrix} \begin{bmatrix} \boldsymbol{x} \\ \boldsymbol{y} \end{bmatrix} \theta \qquad \text{with} \qquad \theta = \frac{M_z}{GI_t} \tag{2.79}$$

This expression includes the work from the external force M_z. For shear loading the initial strain is

$$\boldsymbol{\epsilon}_{0,s} = \begin{bmatrix} (L-z)(C_x x + C_y y) \\ -\nu\left(\frac{1}{2}C_x(x^2-y^2) + C_y xy\right) \\ -\nu\left(\frac{1}{2}C_y(y^2-x^2) + C_x xy\right) \end{bmatrix} \tag{2.80}$$

with the two constants C_x and C_y defined as

$$C_x = \frac{V_x}{EI_y} \qquad\qquad C_y = \frac{V_y}{EI_x} \tag{2.81}$$

We now apply Eq. (2.77) to the strain energy equation and get

$$U = \frac{1}{2}\int_V (\boldsymbol{\epsilon}_0^T + (\boldsymbol{Br})^T)\boldsymbol{C}(\boldsymbol{\epsilon}_0 + \boldsymbol{Br})dV \tag{2.82}$$

which in its fully extended form is written as

$$U = \frac{1}{2}\int_V \left(\boldsymbol{\epsilon}_0^T\boldsymbol{C}\boldsymbol{\epsilon}_0 + \boldsymbol{\epsilon}_0^T\boldsymbol{CBr} + \boldsymbol{r}^T\boldsymbol{B}^T\boldsymbol{C}\boldsymbol{\epsilon}_0 + \boldsymbol{r}^T\boldsymbol{B}^T\boldsymbol{CBr}\right)dV \tag{2.83}$$

The potential energy associated with the applied forces in the case of shear loading is given as

$$W = V_x \cdot u|_{z=L} + V_y \cdot v|_{z=L} \tag{2.84}$$

which can be written on matrix form as

$$W = \begin{bmatrix} V_x & V_y \end{bmatrix} \begin{bmatrix} u|_{z=L} \\ v|_{z=L} \end{bmatrix} \tag{2.85}$$

Mason and Herrmann[13] states the following solution for the equations of three-dimensional linear elasticity theory for the cantilever problem:

$$u = \nu(L-z)\left[\frac{1}{2}C_x(x^2-y^2) + C_y xy\right] + \left(\frac{1}{2}Lz^2 - \frac{1}{6}z^3\right)C_x + a_3 z + a_5 \tag{2.86a}$$

$$v = \nu(L-z)\left[C_x xy + \frac{1}{2}C_y(y^2-x^2)\right] + \left(\frac{1}{2}Lz^2 - \frac{1}{6}z^3\right)C_y + a_2 z + a_4 \tag{2.86b}$$

$$w = -\left(Lz - \frac{1}{2}z^2\right)(C_x x + C_y y) + \Psi(x,y) - a_2 y - a_3 x \tag{2.86c}$$

where the constants $a_2 - a_5$ are defined by satisfying the averages of the displacement boundary conditions at the *fixed* end of the beam. They are given as

$$a_2 = \frac{\int\int E\Psi(x,y)y dx dy}{EI_x} \tag{2.87a}$$

$$a_3 = \frac{\int\int E\Psi(x,y)x dx dy}{EI_y} \tag{2.87b}$$

$$a_4 = -\frac{\nu L}{2AE}\big(EI_x - EI_y\big)C_y \tag{2.87c}$$

$$a_5 = -\frac{\nu L}{2AE}\big(EI_y - EI_x\big)C_x \tag{2.87d}$$

with A as the area of the cross-section. We rewrite the potential from external forces as

$$W = \begin{bmatrix} V_x & V_y \end{bmatrix} \begin{bmatrix} \frac{1}{3}L^3 C_x + La_3 + a_5 \\ \frac{1}{3}L^3 C_y + La_2 + a_4 \end{bmatrix} \tag{2.88}$$

Eqs. (2.87a and 2.87b) can be rewritten as (by multiplying with V_x and V_y respectively)

$$V_x a_3 = C_x \int\int E\Psi(x,y)x dx dy \tag{2.89a}$$

$$V_y a_2 = C_y \int\int E\Psi(x,y)y dx dy \tag{2.89b}$$

Now we rewrite the potential from external forces in Eq. (2.88) as

$$W = \begin{bmatrix} C_x & C_y \end{bmatrix} \Bigg(\begin{bmatrix} L\int_A E(\boldsymbol{N}\boldsymbol{r})x dA \\ L\int_A E(\boldsymbol{N}\boldsymbol{r})y dA \end{bmatrix} + \begin{bmatrix} S_4 \\ S_5 \end{bmatrix} \Bigg) \tag{2.90}$$

where the constants S_4 and S_5 are defined as

$$S_4 = \big(\frac{1}{3}V_x L^3 + a_5 EI_y\big) \qquad S_5 = \big(\frac{1}{3}V_y L^3 + a_4 EI_x\big) \tag{2.91}$$

Eq. (2.90) can be rewritten as

$$W = L\boldsymbol{c}^T\big(\int_A \boldsymbol{d}\boldsymbol{N}\boldsymbol{r} dA + \boldsymbol{s}\big) \tag{2.92}$$

where

$$\boldsymbol{c} = \begin{bmatrix} C_x \\ C_y \end{bmatrix} \qquad \boldsymbol{d} = E\begin{bmatrix} x \\ y \end{bmatrix} \qquad \boldsymbol{s} = \begin{bmatrix} S_4 \\ S_5 \end{bmatrix}$$

Now we can rewrite the total potential energy of the system as

$$\Pi = U - W = \frac{1}{2}\int_V \left(\boldsymbol{\epsilon}_0^T \boldsymbol{C}\boldsymbol{\epsilon_0} + \boldsymbol{\epsilon}_0^T \boldsymbol{C}\boldsymbol{B}\boldsymbol{r} + \boldsymbol{r}^T\boldsymbol{B}^T\boldsymbol{C}\boldsymbol{\epsilon}_0 + \boldsymbol{r}^T\boldsymbol{B}^T\boldsymbol{C}\boldsymbol{B}\boldsymbol{r}\right) dV - L\boldsymbol{c}^T\left(\int_A \boldsymbol{d}\boldsymbol{N}\boldsymbol{r}\, dA + \boldsymbol{s}\right) \tag{2.93}$$

Principle of minimum potential energy states that

$$\delta\Pi = \frac{\delta\Pi}{\delta\boldsymbol{r}}\delta\boldsymbol{r} = \frac{1}{2}\int_V \left(\boldsymbol{\epsilon}_0^T\boldsymbol{C}\boldsymbol{B}\delta\boldsymbol{r} + \delta\boldsymbol{r}^T\boldsymbol{B}^T\boldsymbol{C}\boldsymbol{\epsilon}_0 + \delta\boldsymbol{r}^T\boldsymbol{B}^T\boldsymbol{C}\boldsymbol{B}\boldsymbol{r} + \boldsymbol{r}^T\boldsymbol{B}^T\boldsymbol{C}\boldsymbol{B}\delta\boldsymbol{r}\right) dV \\ - L\boldsymbol{c}^T\int_A \boldsymbol{d}\boldsymbol{N}\delta\boldsymbol{r}\, dA = 0 \tag{2.94}$$

$$\delta\Pi = \int_V \left(\delta\boldsymbol{r}^T\boldsymbol{B}^T\boldsymbol{C}\boldsymbol{\epsilon}_0 + \delta\boldsymbol{r}^T\boldsymbol{B}^T\boldsymbol{C}\boldsymbol{B}\boldsymbol{r}\right) dV - L\delta\boldsymbol{r}^T\int_A \boldsymbol{N}^T\boldsymbol{d}^T\, dA\cdot\boldsymbol{c} = 0 \tag{2.95}$$

$$\Longrightarrow \cancel{\delta\boldsymbol{r}^T}\int_V \boldsymbol{B}^T\boldsymbol{C}\boldsymbol{B}\, dV\boldsymbol{r} = -\cancel{\delta\boldsymbol{r}^T}\int_V \boldsymbol{B}^T\boldsymbol{C}\boldsymbol{\epsilon}_0 dV + \cancel{\delta\boldsymbol{r}^T} L\int_A \boldsymbol{N}^T\boldsymbol{d}^T\, dA\cdot\boldsymbol{c} \tag{2.96}$$

$\boldsymbol{B}$, $\boldsymbol{C}$, $\boldsymbol{c}$, and $\boldsymbol{d}$ are independent of z. $\boldsymbol{\epsilon}_0$ is not however, and we take the mean ans rewrite the previous equation as

$$\Longrightarrow \cancel{L}\int_A \boldsymbol{B}^T\boldsymbol{C}\boldsymbol{B}\boldsymbol{r}\, dA = \cancel{L}\int_A \boldsymbol{N}^T\boldsymbol{d}^T\, dA\cdot\boldsymbol{c} - \cancel{L}\int_A \boldsymbol{B}^T\boldsymbol{C}\,\overline{\boldsymbol{\epsilon}_0}\, dA \tag{2.97}$$

which we recognize as the equilibrium equation,

$$\boldsymbol{K}\boldsymbol{r} = \boldsymbol{R} \tag{2.98}$$

with the global stiffness matrix (which was defined identically in Section 2.9.1), load vector, and mean initial strain from both torsion and shear loading defined as

$$\boldsymbol{K} = \int_A \boldsymbol{B}^T\boldsymbol{C}\boldsymbol{B}\, dA \qquad \text{and} \qquad \boldsymbol{R} = \int_A \boldsymbol{N}^T\boldsymbol{d}^T\, dA\cdot\boldsymbol{c} - \int_A \boldsymbol{B}^T\boldsymbol{C}\,\overline{\boldsymbol{\epsilon}_0}\, dA \tag{2.99}$$

$$\overline{\boldsymbol{\epsilon}}_0 = \overline{\boldsymbol{\epsilon}}_{0_s} + \overline{\boldsymbol{\epsilon}}_{0_t} = \begin{bmatrix} \frac{1}{2}L(C_x x + C_y y) \\ -\nu\left(\frac{1}{2}C_x(x^2-y^2) + C_y xy\right) - \boldsymbol{N}\boldsymbol{y}\theta \\ -\nu\left(\frac{1}{2}C_y(y^2-x^2) + C_x xy\right) + \boldsymbol{N}\boldsymbol{x}\theta \end{bmatrix} \tag{2.100}$$

2.11.2 Element formulation

The nodal displacement $\boldsymbol{v}$ of an element is related through the global displacements by the following kinematic relationship

$$\boldsymbol{v} = \boldsymbol{A}_e \boldsymbol{r} \tag{2.101}$$

We now rewrite Eq. (2.96) with regards to elements

$$\delta\boldsymbol{r}^T \int_A \boldsymbol{B}^T \boldsymbol{C}\boldsymbol{B}\boldsymbol{r}\, dA = \sum_{e=1}^{n} \delta\boldsymbol{r}^T \boldsymbol{A}_e^T \int_{A_e} \boldsymbol{B}_e^T \boldsymbol{C}_e \boldsymbol{B}_e\, dA\, \boldsymbol{A}_e \boldsymbol{r} \tag{2.102}$$

From this we can define the global and element stiffness matrices (respectively) as

$$\boldsymbol{K} = \sum_{e=1}^{n} \boldsymbol{A}_e^T \boldsymbol{k}_e \boldsymbol{A}_e \tag{2.103a}$$

$$\boldsymbol{k}_e = \int_{A_e} \boldsymbol{B}_e^T \boldsymbol{C}_e \boldsymbol{B}_e\, dA \tag{2.103b}$$

Using the same approach for loading gives the global and element load vectors (respectively)

$$\boldsymbol{R} = \sum_{e=1}^{n} \boldsymbol{A}_e^T \boldsymbol{S}_e^0 \tag{2.104a}$$

$$\boldsymbol{S}_e^0 = \int_{A_e} \left(\boldsymbol{d}_e^T \boldsymbol{c} - \boldsymbol{B}_e^T \boldsymbol{C}_e\, \overline{\boldsymbol{\epsilon}}_{0,e}\right) dA \tag{2.104b}$$

The global equilibrium equation is then solved as described in Section 2.7. Finally, stresses for each element are computed as

$$\boldsymbol{\tau}_e = \boldsymbol{C}_e\left(\boldsymbol{B}_e \boldsymbol{v} + \boldsymbol{\epsilon}_{0,e}\right) \tag{2.105}$$

where the constitutive matrix for an element is given by

$$\boldsymbol{C}_e = \begin{bmatrix} C_{11} & C_{12} & C_{13} \\ C_{21} & C_{22} & C_{23} \\ C_{31} & C_{32} & C_{33} \end{bmatrix} \tag{2.106}$$

For linear elastic isotropic materials the constitutive matrix is simplified to

$$\boldsymbol{C}_e = \begin{bmatrix} E_e & 0 & 0 \\ 0 & G_e & 0 \\ 0 & 0 & G_e \end{bmatrix} \tag{2.107}$$

with the Young's modulus and shear modulus related through

$$G_e = \frac{E_e}{2(1+\nu)} \tag{2.108}$$

2.11.3 Composite materials

A composite material is comprised of two or more materials. When combined, the composite will get significantly different properties than the individual components. Composite materials are developed because they are stronger and lighter compared to traditional materials. One of the most widely used composites is the fiber-reinforced crossplied laminate. An example with three plies is illustrated in Fig. 2.18. The fibers of each ply have different orientations, hence the material properties differ in each direction for the plies.

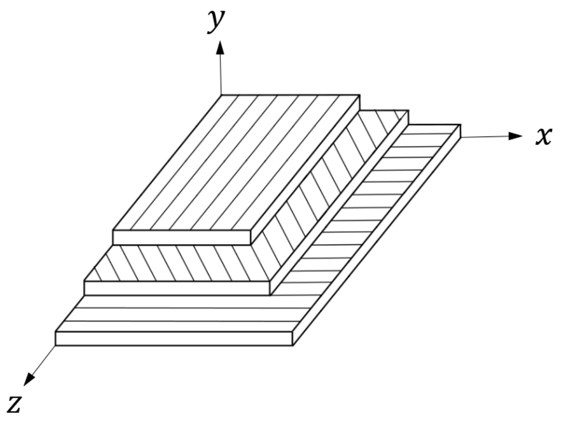


Figure 2.18: Arbitrary laminate with three plies, each having their own fiber orientation

The constitutive matrix defined in Eq. (2.107) only holds for isotropic materials. The reason for this is that the stresses are more strongly coupled for anisotropic materials. In Fig. 2.19 an axial tensile force (N) is applied to a material with fibers oriented in a different direction. This material will not only deform in the direction of the force (w) (unlike an isotropic material). The fibers in the material forces a deformation in the transverse direction as well (u). Similarly, a shear force would lead to both a transverse and an axial deformation. This means that the normal force is coupled to the shear strains, and the shear force is coupled to the normal strain. Consequently the components C_{12} and C_{13} of the constitutive matrix can no longer be assumed to be zero.

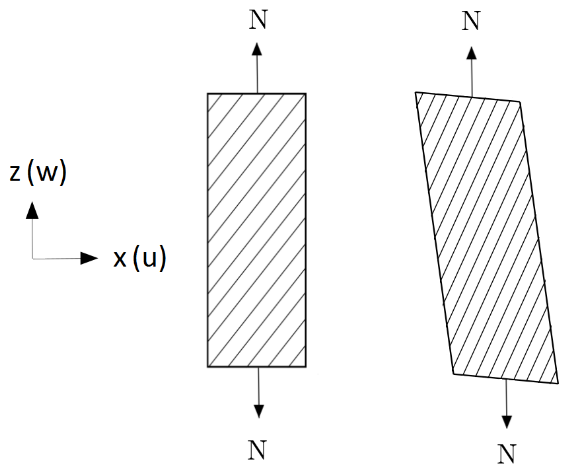


Figure 2.19: Coupling between axial and transverse displacements for composite material subjected to tensile loading

Fig. 2.20 illustrates another composite design with a continuous spiral surface along the length of the rod. When torsion (M_z) is applied to the cross-section, an axial deformation (w) will occur in addition to the regular rotation of the cross-section (ϕ). Similarly, the cross-section will get a rotation in addition to the regular axial deformation when a tensile force (N) is applied.

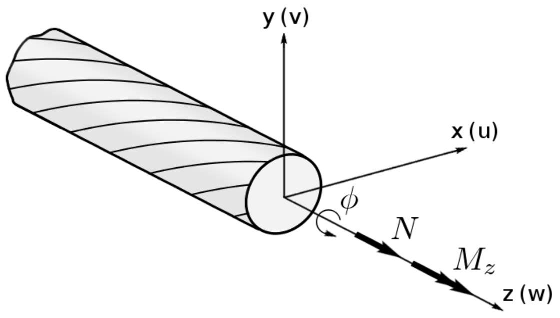


Figure 2.20: Rod with spiral surface subjected to axial force and torsion

The consequence of the two deformation patterns just observed is that the axial force N must be included in the expression for the potential of the external forces in Eq. (2.84). In addition, the potential of the torsion moment should be included in the same expression, rather than directly in the expression for initial strain due to torsion in Eq. (2.79). It is now clear that the final expression for element loading and stress calculations at the end of Section 2.11.1 are not complete. However, we consider it a good starting point for a continued development for analysis of composite materials. We leave it to another thesis to complete the theory, which also includes determining the coefficients of the constitutive matrix.

2.12 Stress recovery

Because the stresses are derived from the nodal displacements, they are often referred to as *derived quantities*, while the displacements are the *primary quantities*. In general the accuracy of primary quantities are greater than the accuracy of the derived quantities. Because of this, stress recovery techniques (with a very low overhead of computational effort) are often used in order to improve the accuracy of the nodal stresses. In CCSC we have made use of two approaches for the calculation of the stresses:

1. evaluate stresses directly at the nodes
2. evaluate stresses at the integration points and then extrapolating the stress values to the nodes

For triangular elements the difference between the two approaches is insignificant, and approach number one is used. For quadrilateral elements however, approach number two is used. The reason for this is that the stress values found inside these elements are often more accurate than the values found on the edges. Thus we can increase the accuracy of our solution by computing the stress values for each Gauss point, and extrapolating the values to the corner nodes using bilinear extrapolation. The stress recovery method used in CCSC is based on the approach described in *Introduction to Finite Element Methods* by Felippa[14].

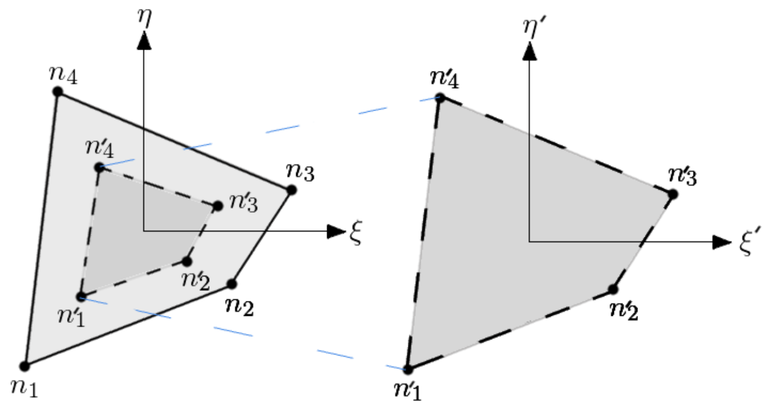


Figure 2.21: Quadrilateral element and inner Gauss element

As seen in Fig. 2.21 each element has four corners n_1, n_2, n_3, and n_4. The element also contains a sub-element defined by its Gauss points, called the *Gauss element*. The Gauss element has the corner nodes $n'_1 n'_2, n'_3$, and n'_4. If we want to extrapolate the stress of n'_1 to the corner n_1, we simply replace ξ and η by $-\sqrt{3}$, and insert it into the shape function (the extrapolation values for ξ and η can be seen in Table 2.1 below). This gives us

$$\boldsymbol{E_1} = \begin{bmatrix} 1+\frac{1}{2}\sqrt{3} & -\frac{1}{2} & 1-\frac{1}{2}\sqrt{3} & -\frac{1}{2} \end{bmatrix}$$

Table 2.1: Natural coordinates used for extrapolating stress values

Node nr.	ξ	η	ξ'	η'
1	-1	-1	-$\sqrt{3}$	-$\sqrt{3}$
2	+1	-1	+$\sqrt{3}$	-$\sqrt{3}$
3	+1	+1	+$\sqrt{3}$	+$\sqrt{3}$
4	-1	+1	-$\sqrt{3}$	+$\sqrt{3}$

Repeating the procedure for the three other nodes, gives us the following extrapolation matrix

$$\boldsymbol{E} = \begin{bmatrix} 1+\frac{1}{2}\sqrt{3} & -\frac{1}{2} & 1-\frac{1}{2}\sqrt{3} & -\frac{1}{2} \\ -\frac{1}{2} & 1+\frac{1}{2}\sqrt{3} & -\frac{1}{2} & 1-\frac{1}{2}\sqrt{3} \\ 1-\frac{1}{2}\sqrt{3} & -\frac{1}{2} & 1+\frac{1}{2}\sqrt{3} & -\frac{1}{2} \\ -\frac{1}{2} & 1-\frac{1}{2}\sqrt{3} & -\frac{1}{2} & 1+\frac{1}{2}\sqrt{3} \end{bmatrix} \tag{2.109}$$

Once **E** is found we can simply extrapolate the values to the element corner nodes by

$$\boldsymbol{\tau} = \mathbf{E}\boldsymbol{\tau}' \tag{2.110}$$

2.13 Mesh convergence

In FEA a model is discretized into a set of finite elements. The number of elements in the mesh greatly affects the accuracy of the solution. In most cases increasing the number of elements makes the solution converge towards an exact value. In cases with high stress concentrations (*singularities*), however, the stresses will not converge as the mesh is refined. In fact the stresses in the singularity will keep approaching infinity.

A stress concentration occurs when the stress gradient in a small localized area is large. This typically happens when rapid changes in geometry are present. The sudden change in geometry disrupts the stresses from flowing smoothly through the structure. Fig. 2.22 shows an example of this. Other causes for singularities involve point loads being applied, as well as changes in material properties[15]. Even though stress singularities pollute the results in its vicinity, results further away can still be accurate. This is a consequence of the St.Venant's principal, which states that "local stresses in one region of a structure do not affect the stresses elsewhere." [16] In general it is the task of the analyst to identify these singularities, and judge whether or not they are of importance for the analysis. Ways to avoid singularities includes smoothing sharp edges using fillets.

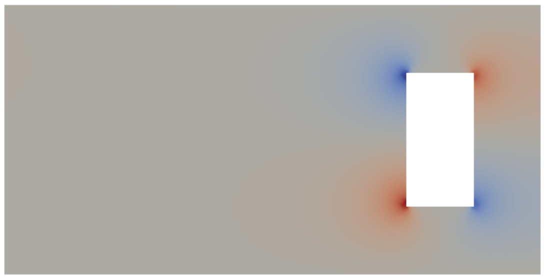


Figure 2.22: Examples of singularities in analysis using CCSC

Chapter 3

Theory - Thin-Walled Cross-Sections

This chapter presents the theory for analysis of thin-walled cross-sections. The main goal is to find cross-sectional parameters and stress distributions over thin-walled cross-sections due to applied section forces. By simplifying the model as a thin-walled structure, huge time-savings can be had, while still retaining sufficient accuracy. The chapter is mainly a reprised version of the theory presented in the CrossX documentation by Bell et al.[3], which in turn is based on the work of Holthe[17].

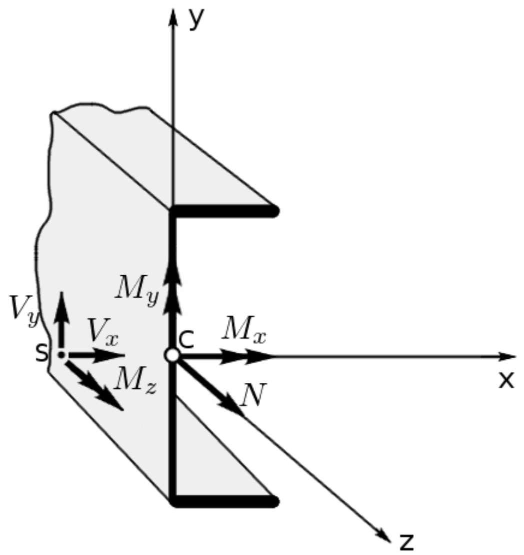


Figure 3.1: Example of a thin-walled cross-section with applied section forces in the area center (C) and shear center (S)

3.1 Applied loads

Fig. 3.1 shows how the different loads are applied to a thin-walled cross-section oriented to coincide with the principal axis. The moments (M_x and M_y), as well as the normal force N is applied in the area center (C). The shear forces (V_x and V_y) and total torsional moment (M_z) are applied in the shear center (S). The total torsional moment is given as

$$M_z = T_{stv} + T_{wrp} \tag{3.1}$$

where T_{stv} is the St. Venant torsion and T_{wrp} is the warping torsion. In addition to the illustrated loads, the bimoment (Bi) is applied at the fixed end of the beam.

3.2 Assumptions

For the thin-walled cross-sections the following assumptions are made

- the beam is straight with a constant cross-section over its length
- all deformations in the local plane of the cross-section is neglected
- bending deformations obey Navier's hypothesis[18] (plane sections remain plane)
- the span to depth ratio should be ten at the very least
- local transverse moments and shear forces are not included in our analysis
- open line elements do not include shear stresses from torsional loading

In addition the cross-section is assumed to be made up of a series of straight line segments. It should also be noted that it is the user's responsibility to create a valid mesh, as there is no mesh generation for thin-walled cross-sections offered by CCSC. This subject is explained in detail in Section 5.6.

3.3 Basic equations and cross-sectional parameters

The relationship between stresses, section forces, and moments are given as

$$N = \int_A \sigma_z \, dA \quad , \quad M_x = \int_A \sigma_z \, y dA \quad , \quad M_y = -\int_A \sigma_z \, x dA \quad , \quad Bi = -\int_A \sigma_z \, \omega dA \tag{3.2}$$

where *Bi* is the bimoment (often referred to as *warping moment*), and ω is the sectorial area (see Section 3.4). The cross-sectional stiffnesses are simply found using

$$EA = \int_A E dA \quad , \quad EI_x = \int_A E y^2 dA \quad , \quad EI_y = \int_A E x^2 dA \quad , \quad EI_\omega = \int_A E \omega^2 dA \tag{3.3}$$

The cross-sectional parameters are then found by dividing the stiffnesses by the Young's modulus of the base material. The St. Venant stiffness (GI_t) is defined in Section 3.5.

3.4 Cross-section geometry conventions

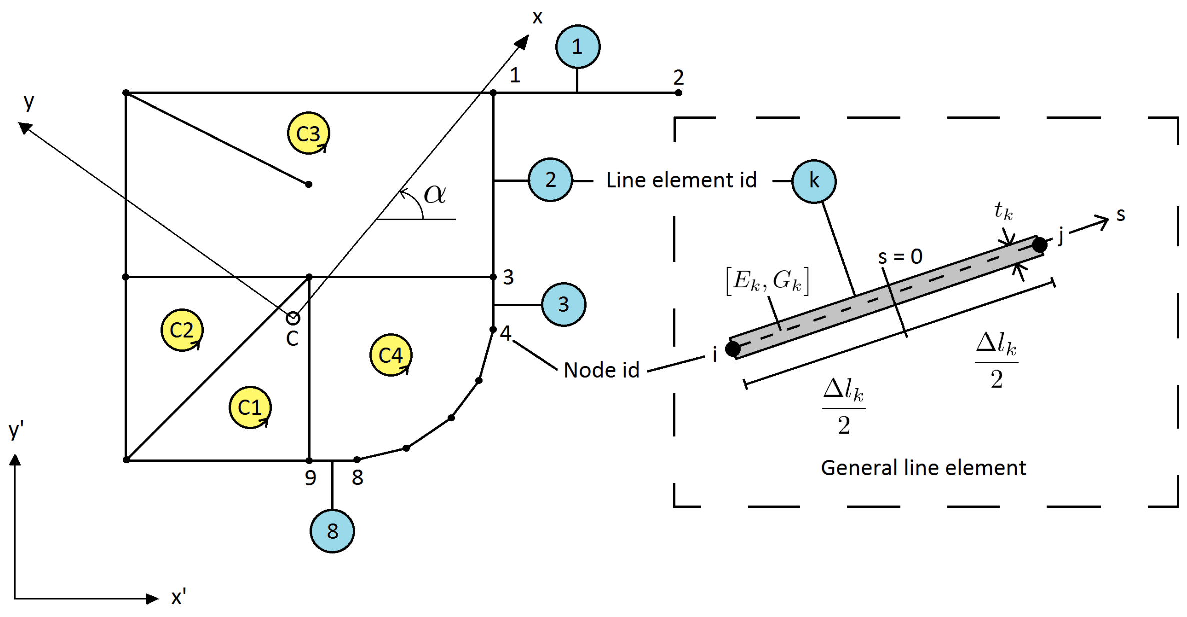


Figure 3.2: Geometric conventions used for thin-walled cross-sections

Conventions used in CCSC can be seen in Fig. 3.2. Each cross-section is discretized as a number of straight line segments (elements) made up of two nodes. Each element has a unique id, a constant thickness t_k, and a local coordinate system s. This coordinate system is directed from node i to node j, and ranges from $\pm\frac{\Delta l_k}{2}$ (with zero at the mid point). Elements are interconnected through the nodes, and each node has its own unique id.

A *cell* is defined as a cycle not containing any other cycles. The cross-section depicted in Fig. 3.2 has four of them, C1, C2, C3, and C4. The cross-section uses a global reference system (x', y', z'), where z' is parallel to the beam length. The material weighted area center (C) and the principal angle (α) are found in a straight forward manner. It should be noted that dividing a straight line into several elements will not give any advantages, and will only result in an increase in computational effort. For the remaining computations the nodal coordinates are transformed to coincide with the principal axis.

3.5 Shear flow in cells

If the cross-section has at least one cell, the torsional stiffness due to St. Venant torsion is found by solving a system of linear equations where the unknowns $\boldsymbol{p}$ are modified shear flows around the cells. This makes the number of unknowns equal to the number of cells. For CCSC we define shear flow to be positive if it is counterclockwise. The equations are written as

$$\boldsymbol{B}\boldsymbol{p} = \boldsymbol{a} \tag{3.4}$$

where $\boldsymbol{a}$ is a vector containing the cell areas, and $\boldsymbol{B}$ is a coefficient matrix. The diagonal of $\boldsymbol{B}$ is given as

$$B_{rr} = \sum_{k=1}^{n_{rr}} \frac{\Delta l_k}{G_k t_k} \tag{3.5}$$

where n_{rr} is the number of elements in cell r. The off-diagonals of $\boldsymbol{B}$ are obtained by

$$B_{rs} = B_{sr} = \sum_{k=1}^{n_{rs}} - \frac{\Delta l_k}{G_k t_k} \tag{3.6}$$

where n_{rs} is the number of elements that are common to both cells r and s. If no common elements exist between cells r and s, the corresponding coefficient is zero ($B_{rs} = 0$). Once the modified shear flows are found, the total St. Venant stiffness can be expressed as

$$GI_t = 4\mathbf{p}^T\mathbf{a} + \frac{1}{3}\sum_{k=1}^{n_f} G_k {t_k}^3 \Delta l_k \tag{3.7}$$

where n_f is the number of *open elements* (elements not part of a cell). To find the torsion constant (I_t), the St. Venant stiffness is simply divided by the shear modulus of the base material.

Three different cases may occur when determining the resulting modified shear flow in an element:

- the element is part of two cells
- the element is part of one cell
- the element is an open element

In case one, where the element is part of two cells (r and s), we can express the resulting modified shear flow as

$$\tilde{q}_k = \eta_k(p_r - p_s) \tag{3.8}$$

In case two, where the element is only part of one cell (r), the shear flow is given as

$$\tilde{q}_k = \eta_k p_r \tag{3.9}$$

where η_k is a factor that can be ± 1. If the flow p_r has the same direction as the line element, then η_k is +1 (and -1 otherwise). For the first case (element is part of two cells), the factor ensures that the the shear flows in the cell wall cancel each other out. In case three (open elements) the modified shear flow is equal to zero. The real shear flow of an element can now be expressed as

$$q_{k,stv} = \frac{2\tilde{q}_k}{GI_t} T_{stv} \tag{3.10}$$

The shear stress is then found by dividing the shear flow by the thickness of the element, as follows

$$\tau_{stv} = \frac{q_{k,stv}}{t_k} \tag{3.11}$$

For open line elements, the shear stress varies linearly over the thickness, and the maximum value (which is always at the edges of the element) is obtained by

$$\tau_{max} = \frac{G_k t_k}{GI_t} T_{stv} \tag{3.12}$$

These stresses are not included in the resulting shear stress nor the effective stress.

3.6 The ω - diagram

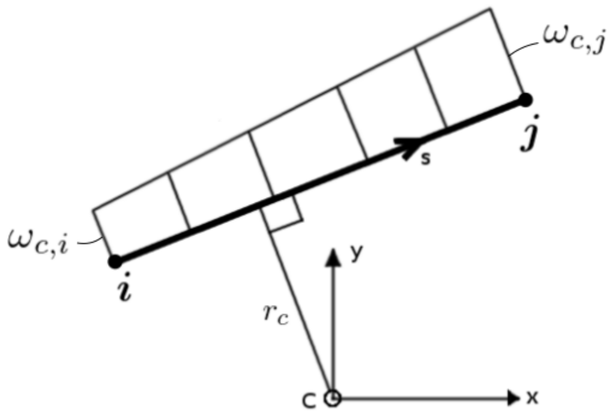


Figure 3.3: ω_c - diagram for an arbitrary line element

Once the resulting shear flows are found, we can determine the ω_c - diagram of the cross-section. This diagram is referred to the area center C of the cross-section. Fig. 3.3 illustrates an arbitrary element with nodal coordinates referred to the principal axis (origin in area center C), and where r_c is the distance from the origin to the element. The differential of ω_c is expressed as

$$d\omega_c = \left(r_c - \frac{2\tilde{q}}{G_k t_k} \right) ds \tag{3.13}$$

From this we can show that the slope of ω_c from node i to j is given by

$$\Delta\omega_c = \omega_{cj} - \omega_{ci} = (x_i y_j - x_j y_i) - \frac{2\tilde{q}\Delta l}{G_k t_k} \tag{3.14}$$

Around each cell the sum of $\Delta\omega_c$ is zero. This means that we can determine the ω_c - diagram through the following three steps:

- start at an arbitrary node *i* and assign it a value of zero
- use Eq. (3.14) to find the ω_c - value of *i's* neighboring nodes
- repeat the process until all nodes have been assigned their ω_c - values

Once the ω_c - diagram has been determined, the ω - diagram (referred to the shear center S) can be found by using the following equation

$$\omega = \omega_c - (x_s y - y_s x) + C \tag{3.15}$$

where the shear center S (x_s, y_s), and shear constant C is found by

$$x_s = \frac{\int_A E\omega_c y dA}{EI_x} \tag{3.16}$$

$$y_s = -\frac{\int_A E\omega_c x dA}{EI_y} \tag{3.17}$$

$$C = -\frac{\int_A E\omega_c dA}{EA} \tag{3.18}$$

3.7 Axial stress distribution

Having determined the cross-sectional stiffnesses as well as the ω - diagram, the axial stress distribution over the cross-section is simply computed as

$$\sigma_z = E_k\left(\frac{N}{EA} + \frac{M_x}{EI_x}y - \frac{M_y}{EI_y}x - \frac{Bi}{EI_\omega}\omega\right) \tag{3.19}$$

where E_k is the Young's modulus of line element k.

3.8 Shear stress distribution

The shear stress distribution over the cross-section is found as the sum of the contribution from pure St. Venant torsion (using Eq. (3.11)) and a change in axial stress. We will now only consider the latter contribution. It is found by solving a system of equations where the unknowns are the mid-point shear flows of each line element, as illustrated in Fig. 3.4.

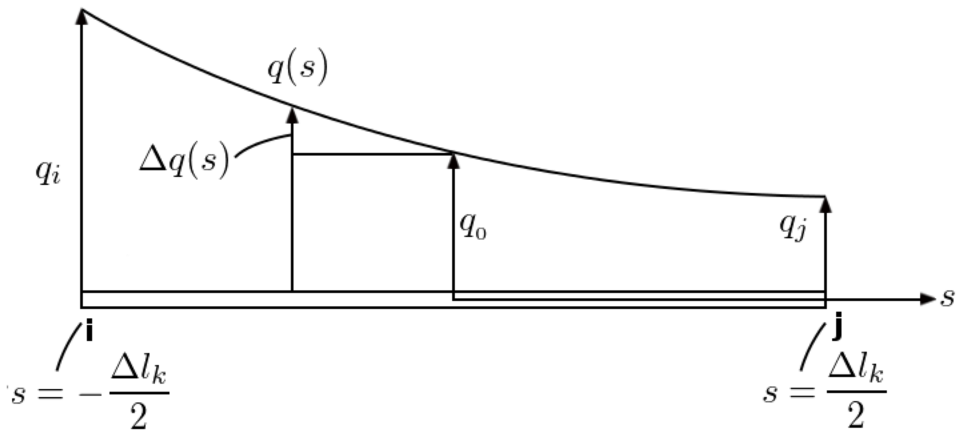


Figure 3.4: Shear flow distribution for line element k

At any arbitrary point along the element, the shear flow is given as

$$q(s) = q_0 + \Delta q(s) \tag{3.20}$$

The increment in shear stress over a distance Δs is given as

$$\Delta\tau = \frac{E_k}{t_k}\left(\frac{N^{'}}{EA}\Delta A + \frac{M_x^{'}}{EI_x}\Delta S_x - \frac{M_y^{'}}{EI_y}\Delta S_y - \frac{Bi^{'}}{EI_\omega}\Delta S_\omega\right) + \frac{1}{t_k}\int_{\Delta s} p_z ds \tag{3.21}$$

where

$$\Delta A = \int_{\Delta s} ds \quad \Delta S_x = \int_{\Delta s} yds \quad \Delta S_y = \int_{\Delta s} xds \quad \Delta S_\omega = \int_{\Delta s} \omega ds \tag{3.22}$$

and p_z is an external loading in the beam direction (along the z-axis). Apostrophes indicate differentiation with respect to z. Using Eq. (3.21) the increment in shear flow can be expressed as

$$\Delta q(s) = E_k t_k \left(\frac{N'}{EA} s + \frac{M'_x}{EI_x} \int_0^s y ds - \frac{M'_y}{EI_y} \int_0^s x ds - \frac{Bi'}{EI_\omega} \int_0^s \omega ds \right) \tag{3.23}$$

In matrix notation this is simplified to

$$q(s) = q_0 + E_k t_k \left(\boldsymbol{c}_0^T s + \Delta \boldsymbol{c}^T \left(\frac{s^2}{2\Delta l_k} \right) \right) \boldsymbol{D} \boldsymbol{f}' \tag{3.24}$$

where

$$\boldsymbol{c}_o^T = \begin{bmatrix} 1 & y_0 & x_0 & \omega_0 \end{bmatrix} \tag{3.25a}$$

$$\Delta \boldsymbol{c}^T = \begin{bmatrix} 0 & \Delta y & \Delta x & \Delta \omega \end{bmatrix} \tag{3.25b}$$

$$\boldsymbol{f}^T = \begin{bmatrix} N & M_x & M_y & Bi \end{bmatrix} \tag{3.25c}$$

$$\boldsymbol{f}'^T = \begin{bmatrix} N' & V_y & V_x & T_{wrp} \end{bmatrix} \tag{3.25d}$$

$$\boldsymbol{D} = \begin{bmatrix} \frac{1}{EA} & 0 & 0 & 0 \\ 0 & \frac{1}{EI_x} & 0 & 0 \\ 0 & 0 & \frac{1}{EI_y} & 0 \\ 0 & 0 & 0 & \frac{1}{EI_\omega} \end{bmatrix} \tag{3.25e}$$

There are two conditions that restricts Eq. (3.24). First, for each point in the cross-section there must be an equilibrium in the z-direction. Secondly, there must be no resulting displacement in the z-direction around each cell of the cross-section.

3.8.1 Equilibrium condition

For a given point i the first condition can be stated as

$$\sum_{n_i} \eta_k q_k(\alpha) = p_{zi} \qquad \text{where} \qquad \alpha = -\eta_k \frac{\Delta l_k}{2} \tag{3.26}$$

Where n_i is the number of line elements containing point i. The factor η_k is used to account for the direction of line element k. It is +1 if the direction of the line element is away from the point, and -1 if the direction is towards the point. We now assume an external distributed load p_{zi} to be applied in the z-direction at point i. Combining Eqs. (3.24 and 3.26) we establish

$$\sum_{n_i} \eta_k q_{0,k} + \sum_{n_i} \eta_k E_k t_k \left(\boldsymbol{C}_{0,k}^T \left(-\eta_k \frac{\Delta l_k}{2} \right) + \Delta \boldsymbol{C}^T \left(\frac{\left(-\eta_k \frac{\Delta l_k}{2} \right)^2}{2 \Delta l_k} \right) \right) \boldsymbol{D} \boldsymbol{f'} \; = \; p_{zi} \tag{3.27}$$

which is easily simplified to

$$\sum_{n_i} \eta_k q_{0,k} \; = \; p_{zi} \; + \; \sum_{n_i} E_k t_k \left(\boldsymbol{C}_{0,k}^T \left(\frac{\Delta l_k}{2} \right) - \Delta \boldsymbol{C}^T \left(\frac{\eta_k \Delta l_k}{8} \right) \right) \boldsymbol{D} \boldsymbol{f'} \tag{3.28}$$

The assumed line load p_{zi} may be expressed as a fraction of the axial load differentiated

$$p_{zi} \; = \; -g_{zi} N' \tag{3.29}$$

with an equilibrium requirement over all m_p points in the cross-section

$$\sum_{i=1}^{m_p} g_{zi} \; = \; 1 \tag{3.30}$$

When expressing the equilibrium equations for all m_p points in matrix form, we get

$$\boldsymbol{A}_1 \boldsymbol{q}_0 \; = \; -\boldsymbol{g}_z N' \; + \; \begin{bmatrix} \boldsymbol{g}_1^1 & \boldsymbol{g}_1^2 & \boldsymbol{g}_1^3 & \boldsymbol{g}_1^4 \end{bmatrix} \boldsymbol{f}' = \boldsymbol{G}_1 \boldsymbol{f}' \tag{3.31}$$

where $\boldsymbol{A}_1$ is a coefficient matrix with the corresponding η_k values.

3.8.2 No out-of-plane displacement condition

The condition of no resulting displacement in the z-direction can be expressed as

$$\sum_{n_r} \frac{\eta_k}{G_k t_k} \int_{\Delta l_k} q(s) ds = 0 \tag{3.32}$$

where n_r is the number of elements in cell r. In this case, the factor η_k is +1 if the direction of the line element coincides with a positive counter-clockwise direction around cell r, otherwise η_k is -1. Combining Eqs. (3.24 and 3.32) yields

$$\sum_{n_r} \eta_k \frac{\Delta l_k}{G_k t_k} \left(q_{0_k} + E_k t_k \Delta \boldsymbol{C}_k^T \left(\frac{\Delta l_k}{24} \right) \boldsymbol{D f}' \right) \; = \; 0 \tag{3.33}$$

The $\boldsymbol{C}_0$ term from Eq. (3.24) vanishes as the integral is taken over the length of the element. The previous equation then simplifies to

$$\sum_{n_r} \eta_k \frac{\Delta l_k}{G_k t_k} q_{0_k} \; = - \left(\sum_{n_r} \eta_k \; \Delta \boldsymbol{C}_k^T \frac{E_k}{G_k} \frac{(\Delta l_k)^2}{24} \right) \boldsymbol{D f}' \tag{3.34}$$

The equations meeting the no out-of-plane displacement condition for all m_c cells may be summarized in matrix form as

$$\boldsymbol{A}_2 \boldsymbol{q}_0 \; = \; \begin{bmatrix} \boldsymbol{g}_2^1 & \boldsymbol{g}_2^2 & \boldsymbol{g}_2^3 & \boldsymbol{g}_2^4 \end{bmatrix} \boldsymbol{f}' \; = \; \boldsymbol{G}_2 \boldsymbol{f}' \tag{3.35}$$

where $\boldsymbol{A}_2$ is a coefficient matrix with the corresponding η_k values.

3.8.3 Shear stress system of equations

By combining Eq. (3.31) and Eq. (3.35) we end up with the total system of equations that the shear stresses (originating from the change in axial stress) must satisfy

$$\begin{bmatrix} \boldsymbol{A}_1 \\ \boldsymbol{A}_2 \end{bmatrix} \boldsymbol{q}_0 \; = \; \begin{bmatrix} \boldsymbol{g}_1^1 - \boldsymbol{g}_z & \boldsymbol{g}_1^2 & \boldsymbol{g}_1^3 & \boldsymbol{g}_1^4 \\ \boldsymbol{g}_2^1 & \boldsymbol{g}_2^2 & \boldsymbol{g}_2^3 & \boldsymbol{g}_2^4 \end{bmatrix} \boldsymbol{f}' \; = \; \begin{bmatrix} \boldsymbol{G}_1 \\ \boldsymbol{G}_2 \end{bmatrix} \boldsymbol{f}' \tag{3.36}$$

The number of unknowns is equal to the number of line elements m_l. The number of equations is equal to the number of nodes m_p and the number of cells m_c combined. Generally, this system is overdetermined. This means that we have one more equation than unknowns, because $m_p + m_c = m_l + 1$. Solving this system yields the shear flow at the mid point of every line element due to changes in the axial stress.

3.8.4 Resulting shear stress distribution

Having determined $\boldsymbol{q}_0$, Eq. (3.20) gives us the parabolic shear flow function for all elements due to changes in axial stress. If we assume that the stresses are uniformly distributed over the element thickness, we can obtain them through dividing by said thickness. For non-open elements, the shear stress distribution is found by adding the shear stress from pure St. Venant torsion defined by Eq. (3.11).

$$\tau_{cell} = \frac{q(s) + q_{k,stv}}{t_k} \tag{3.37}$$

For open elements however, the resulting shear stress does not include the stresses due to pure St. Venant torsion obtained by Eq. (3.12). Instead they are simply given as

$$\tau_{open} = \frac{q(s)}{t_k} \tag{3.38}$$

3.9 Shear deformation factors

From elementary beam theory (Euler-Bernoulli) we have that for pure bending about the principal x-axis

$$\frac{d^2\bar{v}}{dz^2} = -\frac{M_x}{EI_x} \tag{3.39}$$

where $\bar{v}$ is the transverse displacement of the beam axis. Timoshenko modified this expression to also include a contribution from shear deformation, based on a mean shear deformation γ_{yz}^m that corresponds to a mean shear stress τ_{yz}^m

$$\tau_{yz}^m = \frac{V_y}{A_{sy}} = \kappa_y \frac{V_y}{A} \implies A_{sy} = \frac{A}{\kappa_y} \tag{3.40}$$

$$\gamma_{yz}^m = \frac{\tau_{yz}^m}{G} = \frac{V_y}{A_{sy}} = \frac{\kappa_y}{AG} V_y \tag{3.41}$$

Where κ_y is the shear deformation factor and A_{sy} is the shear area. When the shear force V_y is applied, the contribution to $\bar{v}$ becomes

$$d\bar{v} = \gamma_{yz}^m dz = \frac{\kappa_y}{AG} V_y dz \implies \frac{d\bar{v}}{dz} = \frac{\kappa_y}{AG} V_y \tag{3.42}$$

The modified expression for curvature is now

$$\frac{d^2\bar{v}}{dz^2} = -\frac{M_x}{EI_x} + \frac{\kappa_y}{AG}\frac{dV_y}{dz} \tag{3.43}$$

The exact same procedure for the other in-plane principal axis yields

$$\frac{d^2\bar{u}}{dz^2} = \frac{M_y}{EI_y} + \frac{\kappa_x}{AG}\frac{dV_x}{dz} \tag{3.44}$$

The shear deformation factors in principal x- and y-directions for applied shear force are then given as

$$\kappa_x = (AG)\sum_k \left(\left(\frac{\Delta x_k}{\Delta L_k}\right)^2 \cdot \frac{t_k}{G_k}\int_{s_k} \left(\tau^2|_{V_x=1}\right) ds\right) \tag{3.45a}$$

$$\kappa_y = (AG)\sum_k \left(\left(\frac{\Delta y_k}{\Delta L_k}\right)^2 \cdot \frac{t_k}{G_k}\int_{s_k} \left(\tau^2|_{V_y=1}\right) ds\right) \tag{3.45b}$$

3.10 Numerical integration

Through this chapter we have presented numerous integrals that need to be evaluated over the line element. For these integrals we have again used Gaussian quadrature. With two integration points along the length of the line element, we can evaluate 3rd order polynomials exactly. This scheme is used for most integrations as it produces exact results. The integrand when determining the the shear deformation factor, however, is of 4th order, and a three point scheme is used for this calculation. The 2 point integration scheme for line elements is illustrated in Fig. 3.5. Each integration point has the assigned weight value of unity.

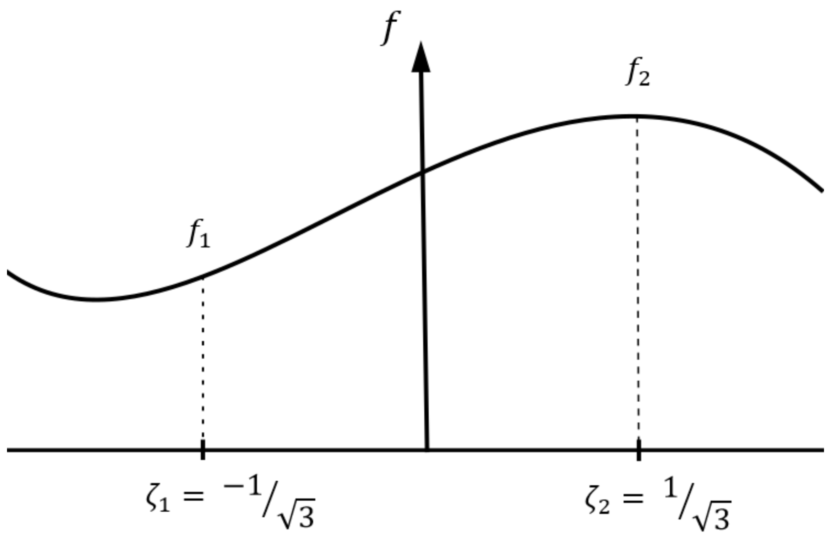

f
f_2
f_1
$\zeta_1 = {}^{-1}/_{\sqrt{3}}$
$\zeta_2 = {}^{1}/_{\sqrt{3}}$

Figure 3.5: Gaussian quadrature for line elements integrating 3rd order polynomials exactly

Chapter 4

Implementation - Massive Cross-Sections

This chapter takes a closer look at the implementation details for analysis of massive cross-sections. We begin the chapter by discussing the class structure and informational flow of CCSC. Next we present how numerical integration is performed for both triangular and quadrilateral elements, followed by an in-depth description of the implementation of torsion, shear, and composite analysis. Lastly, we present the equation solving, visualization, and mesh creation. To further supplement this chapter we have written an extensive documentation containing all functions and their underlying theory. This is available as a part of the program package under the *Documentation* folder.

4.1 Classes

The massive cross-sectional implementation of CCSC currently consists of 14 classes: **Solver**, **Mesh**, **MeshSTL**, **MSHParser**, **Element**, **ResultWriter**, **Material**, **Node**, including the six sub classes **TriangularElement**, **QuadrilateralElement**, **T3Element**, **T6Element**, **Q4Element**, and **Q9Element**. Fig. 4.1 illustrates how these classes are related. **MeshSTL** is left out of the illustration, as it is only used for validation of results and bares no real analytical significance.

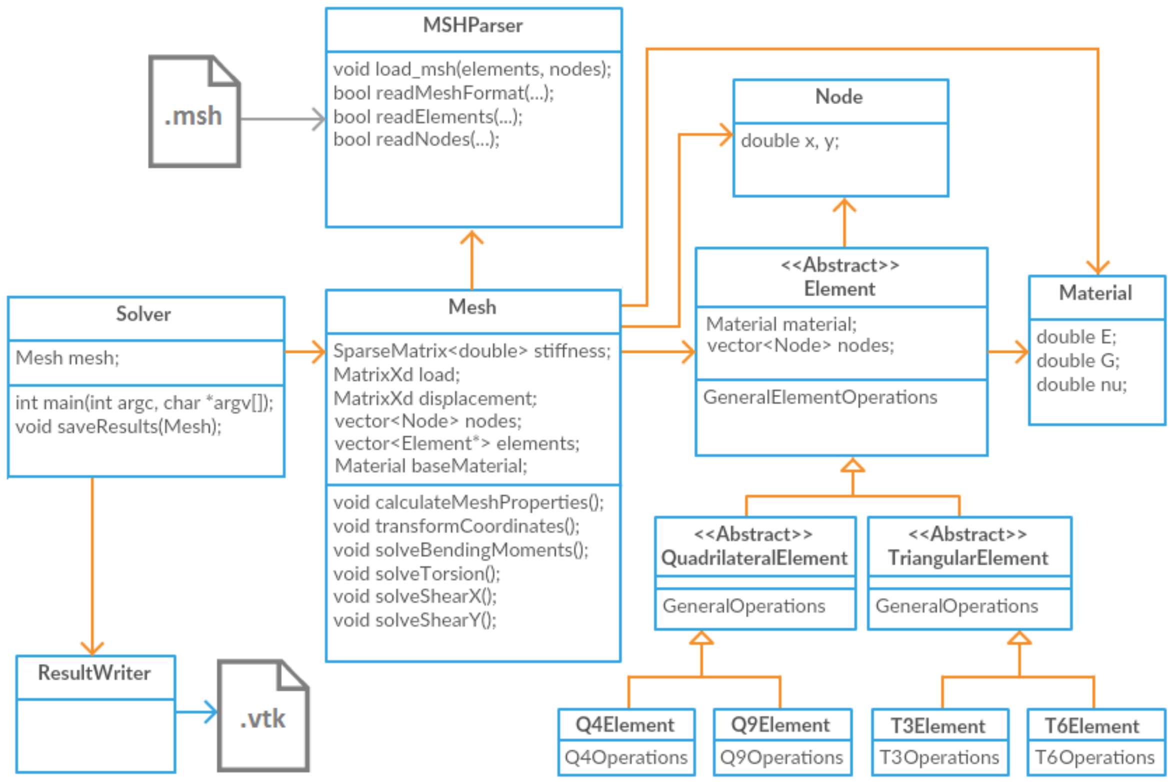


Figure 4.1: UML class diagram showing the relationship between classes for massive implementation

Solver

Solver acts as the entry point for CCSC. It reads input passed to it by the user through the program arguments. A complete list of the available arguments that can be passed to CCSC is given below.

- **-f nameOfMeshFile** decides which mesh file to analyze. If the mesh file lies within a subfolder, the folder is passed as a part of the file name: (-f nameOfFolder/nameOfMeshFile)
- **-material youngsModulusValue poissonRatio** defines an isotropic material and sets it as the base material for the cross-section. In the current massive implementation, all elements are automatically assigned the same material properties as the base material. For a thin-walled mesh, however, individual material properties can be hard-coded for each element through the *.msh* file. Both the Young's modulus and the Poisson's ratio parameters are mandatory when using the command

- **-massive** sets the program to analyze massive cross-sections (this is true by default)
- **-thin extraLineSegments** sets the program to analyze thin-walled cross-sections (defaults to massive). The extraLineSegments parameter is optional to include. It selects the number of elements used in order to approximate the quadratic stress distributions in VTK (it defaults to 100, and can be set as low as 7)
- **-help** prints out a full list of commands for CCSC

The main responsibilities of **Solver** is to instantiate a **Mesh** object (or **LineMesh**, depending on the user input). The object then calculates the cross-sectional parameters and stress distributions, before the results are written to file by the **ResultWriter** (or **LineResultWriter**).

Mesh

Mesh is responsible for performing the massive cross-sectional analysis. It represents the mesh by storing a set of vectors for the nodes and elements. These vectors are passed as pointers to the **MSHParser** where the individual nodes and elements from the mesh file are added. Once the mesh file is parsed, **Mesh** can start the analysis. The total area, centroid, stiffnesses, axial stresses, and second area of moments are all computed by iterating over all the elements and summing up the individual element contributions. Once this is done the cross-section is transformed to coincide with the principal axis for the rest of the computations. This is done through the *transformCoordinates()* function.

The cross-section in CCSC is often rotated the opposite way of CrossX. This is because the principal angle is forced to lie between $-45°$ and $45°$, as presented in Algorithm 1. CrossX on the other hand, follows the more standard procedure of forcing the angle to lie between $-90°$ and $90°$. The convention used in CCSC was introduced by Strømstad during his master thesis, and was never changed as it bares no real significance (other than making it harder to compare results with CrossX).

Algorithm 1 Algorithm for calculating angle of principal axis

```
double angle = atan $\left(\frac{I_x - I_{max}}{I_{xy}}\right) \cdot \frac{180}{\pi}$
anglePrincipalAxis = 0
if (angle < 45 and angle > -45) then
    anglePrincipalAxis = angle
else if (angle > 45) then
    anglePrincipalAxis = angle - 90
else if (angle < -45) then
    anglePrincipalAxis = angle + 90
end if
```

After the cross-section is transformed, **Mesh** computes the axial stress distribution ($\boldsymbol{\sigma_z}$) due to bending moments (using Eq. (2.45)). This is followed by the torsion and shear analyses respectively. The shear stress distributions ($\boldsymbol{\tau_{xz}}$ and $\boldsymbol{\tau_{yz}}$) due to torsional loading M_z (using Eqs. (2.61 and 2.63)), as well as the shear center, are found through the *solveTorsion()* function. The shear stress analyses due to V_x and V_y are performed by *solveShearX()* and *solveShearY()* respectively (using Eqs. (2.68 and 2.71)). This includes the determination of the shear deformation factors. Once all the stress distributions are found, the effective stress following the von Mises criterion can be calculated. The results are all stored in their respective matrices within the class.

MSHParser

The main responsibility of **MSHParser** is to parse the *.msh* file received from the **Mesh**. The constructor of **MSHParser** takes a file name as well as pointers to element and node vectors as arguments. First the parser attempts to open the *.msh* file with the given file name. If the file is not found (or is wrongly formatted) an exception is thrown. If the file is found the class proceeds to parse the file and fill up the node and element vectors according to the *.msh* file. A massive mesh can contain triangular elements, quadrilateral elements, or both combined.

The second integer in each element entry of the *.msh* file defines what element type is instantiated. The integers with their corresponding element types are defined as follows (using Gmsh standard)

- the integer 2 represents T3 elements
- the integer 3 represents Q4 elements

- the integer 9 represents T6 elements
- the integer 10 represents Q9 elements

Appendix C.1 gives the specifics of the *.msh* file format, while Section 4.8 explains how the user can produce mesh files for massive cross-sections.

Element

Element is an abstract class that contains functions common to all elements used in the massive analysis. It specifies which functions need to be implemented in its sub-classes by defining them as pure virtual functions. By making **Element** abstract we achieve polymorphism, which means that we can treat T3, T6, Q4, and Q9 elements as if they were the same. This simplifies the code a a lot, making it easy to both implement new element types, and to make changes in existing ones. All the variables needed in order to calculate the element contributions to the global cross-sectional parameters and stress distributions are stored inside **Element**. This includes the material, area, and element nodes among others.

TriangularElement and QuadrilateralElement

These two classes contain all the functions that need different implementations for the triangular and quadrilateral elements. This is done by implementing function bodies for the desired pure virtual functions defined in **Element**. Some of the pure virtual functions however, are even more specific and they need to be deferred even further to the concrete sub-classes **T3Element**, **T6Element**, **Q4Element**, and **Q9Element**. One of the main advantages of this solution (other than being able to easily add new element types) is that functions can be optimized for each element type. One example of this is the implementation of T3 elements. This element type has a constant strain-displacement matrix $\boldsymbol{B}$, which means it only needs to be computed once. It also means that calculations involving $\boldsymbol{B}$ can be moved outside the integration loops. This can be achieved by having separate function implementations between the T3 and T6 elements, thus saving computational effort.

For quadrilateral elements, computing the $\boldsymbol{N}$, $\boldsymbol{B}$, $\boldsymbol{J}$ (Jacobian matrix), and $\boldsymbol{N1}$ matrices for each Gauss point (as part of the numerical integration) can be computationally heavy. This is certainly the case for cross-sections containing many elements. To avoid having to recompute the same matrices several times, we opted to store them the first time they are computed, and reuse them for later computations. Each matrix has several sub-matrices, one for each Gauss point. These sub-matrices are stored in two-dimensional vectors, in the same order that the Gauss points are iterated through (illustrated in Fig. 4.2). The x and y values of the integration points (called **xIntegration** and **yIntegration** in the code) are also stored in the same manner. Doing this reduced the running time of our quadrilateral implementation by roughly 10-40 % depending on the element type and size of the mesh (see Fig. B.1 in the Appendix). This is only done for quadrilateral elements because the time savings for triangular elements were not as significant.

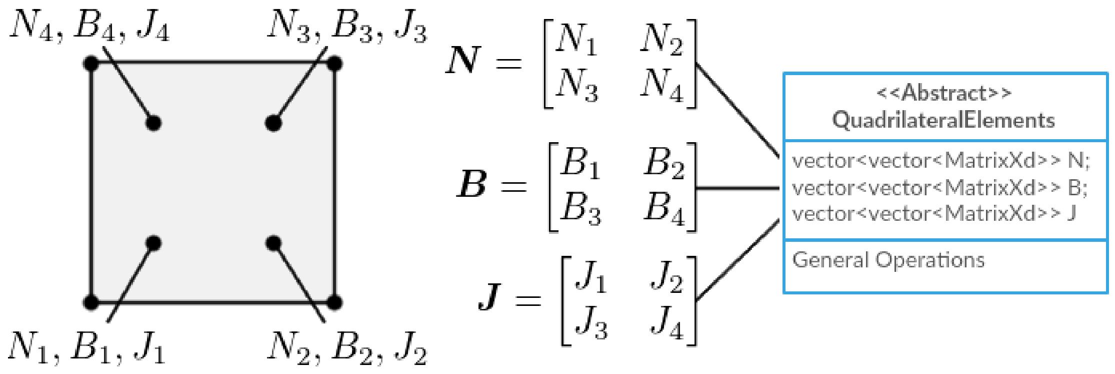

N_4, B_4, J_4
N_3, B_3, J_3
N_1, B_1, J_1
N_2, B_2, J_2
$\boldsymbol{N} = \begin{bmatrix} N_1 & N_2 \\ N_3 & N_4 \end{bmatrix}$
$\boldsymbol{B} = \begin{bmatrix} B_1 & B_2 \\ B_3 & B_4 \end{bmatrix}$
$\boldsymbol{J} = \begin{bmatrix} J_1 & J_2 \\ J_3 & J_4 \end{bmatrix}$
<<Abstract>>
QuadrilateralElements
vector<vector<MatrixXd>> N;
vector<vector<MatrixXd>> B;
vector<vector<MatrixXd>> J
General Operations

Figure 4.2: $\boldsymbol{N}$, $\boldsymbol{B}$, and $\boldsymbol{J}$ are stored in each quadrilateral element ($\boldsymbol{N1}$, **xIntegration**, and **yIntegration** are left out for illustration purposes)

T3, T6, Q4, and Q9 elements

These four classes hold functions for all calculations that are dependent on the element type. This includes *getNMatrix()* which calculates the shape functions $\boldsymbol{N}$, *getBMatrix()* which calculates the strain-displacement matrix $\boldsymbol{B}$, as well as *calculateStiffnessMatrix()* which calculates the element stiffness matrix $\boldsymbol{k_e}$.

ResultWriter

The main responsibility of **ResultWriter** is to output the results that are stored in the mesh. By default the results are outputted in the *.vtk* file format (see Appendix C.3), but a custom format (similar to the *.stl* file format) can also be outputted. This format does not make use of the *nodal point averaging* technique, and it is mainly used for analyzing the performance of different element types. In addition to the VTK output, CCSC writes all material properties, applied loads, cross-sectional parameters, and stiffnesses as a *.txt* file. An example of the *.txt* file is given in Fig. B.7 in the Appendix.

MeshSTL

MeshSTL is almost identical **Mesh**, with only a few key exceptions which relates to how the results are stored. **MeshSTL** stores the result on an element basis, rather than a nodal basis (as is the case for **Mesh**). This is done in order to facilitate a *.stl* like output, without nodal point averaging. This class was purely implemented in order to analyze the effects of nodal point averaging and for comparing linear and quadratic elements (which is demonstrated in Section 6.3.4).

Because it is purely made for validation purposes, we have not spent any time to seamlessly integrate it into the rest of CCSC. As a consequence, in order to use this class the user has to make changes in the source code manually. The boolean *writeSTL* in the **ResultWriter** constructor must be set to *true*, and all references to **Mesh** must be changed to **MeshSTL** in both the header files and the *.cpp* files of **Solver** and **ResultWriter**.

Material

Material is used to define an isotropic material, and acts as both a base material for the entire cross-section, as well as unique materials for each individual element. Each element has a Young's modulus (E), a shear modulus (G), and a Poisson's ratio (ν). The constructor takes E and ν, and calculates G through the relationship

$$G = \frac{E}{2(1+\nu)} \tag{4.1}$$

For the massive implementation all elements are initiated with the same material properties as the base material. This can, however, easily be modified in the future to allow for unique properties for individual elements. For the thin-walled implementation the user can assign individual properties simply by changing the *element type*-tag and adding the desired properties to the element (see Section 5.1). This was not done for massive cross-sections, as the number of elements is usually much larger compared to thin-walled cross-sections. Finding a way to assign material properties to a group of elements through the graphical user interface (GUI) could be an interesting aspect to investigate at a future point in time.

Node

The node class stores an *id* integer, the x- and y-positions of each node, as well as the number of elements that are connected to the node. For massive analysis the computed stresses are stored in **Mesh**. For thin-wall analysis, **Node** stores stresses as well as the ω - value.

4.2 Numerical integration

CCSC performs all integrations using Gaussian quadrature (see Section 2.6 for a detailed description). Arrays containing the Gaussian points and corresponding weights for the different schemes are stored in **TriangularElement** and **QuadrilateralElement**. Algorithm 2 presents pseudocode for the Gaussian quadrature scheme used for triangular elements.

Algorithm 2 Algorithm for numerical integration of triangular elements

```
// f(zeta1, zeta2, zeta3) is the function to be integrated
double I = 0
for (all Gauss points i) do
    I += weight[i]*f(zeta[0][i], zeta[1][i], zeta[2][i])
end for
I *= A
```

The integration for quadrilateral elements is quite similar, but it involves a double for-loop. This is because the Gauss points are defined in two directions (ξ, η), rather than the single direction (ζ). Additionally, the transformation from natural to physical coordinates requires the Jacobian determinant for each integration point. Algorithm 3 presents pseudocode for Gaussian quadrature for quadrilateral elements. For most integrations a 2x2 scheme is used regardless of the element type. A 3x3 scheme is only used when calculating the stiffness matrix for Q9 elements. This integration contains fourth order terms in the integrand and can not be integrated to the exact value by a 2x2 scheme.

Algorithm 3 Algorithm for numerical integration of quadrilateral elements

```
// f(x,y) is the function that is be integrated
double I = 0
for (all Gauss points i in $\xi$ direction) do
    for (all Gauss points j in $\eta$ direction) do
        Matrix N = getNMatrix(xi[i][j], eta[i][j])
        Matrix Jacobi = getJMatrix(xi[i][j], eta[i][j])
        x = N*xCoordinates
        y = N*yCoordinates
        I += weight[i][j]*f(x,y)*Jacobi.determinant
    end for
end for
```

4.3 Torsional analysis

To perform the torsional analysis, we first need to calculate the consistent torsion load vector. This is done by iterating over all elements in **Mesh** and computing the individual element loads by calling the function *calculateElementLoadTorsion()*. The loads are then added to their correct place in the global load vector $\boldsymbol{R}$. Next, the stiffness matrix $\boldsymbol{K}$ is computed. This is done in much the same way as the load vector. The element stiffnesses are found by invoking the *calculateStiffnessMatrix()* function for all elements, and are then added into to the corresponding place in the global stiffness matrix. This procedure is illustrated in Fig. 4.3.

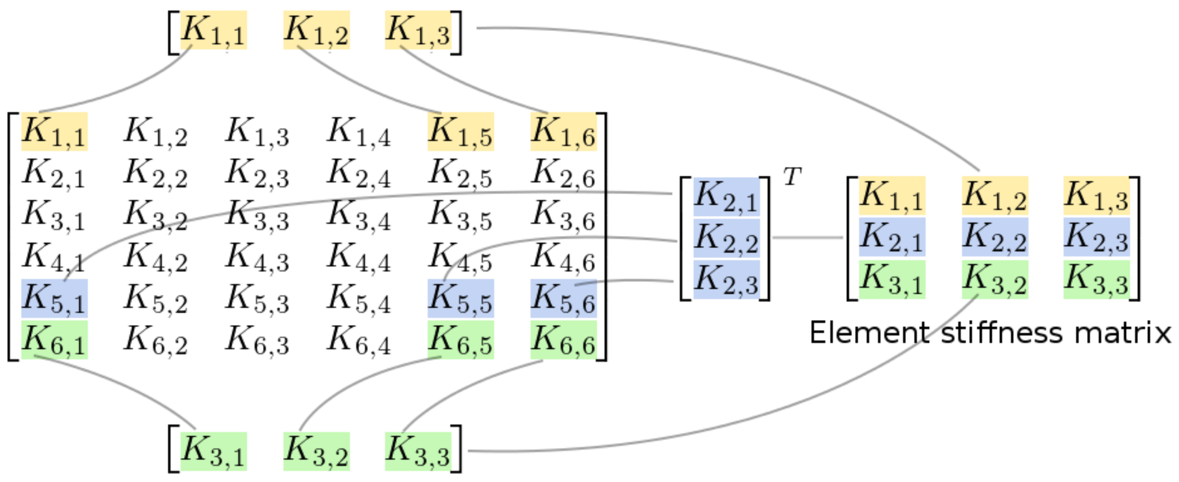

$K_{1,1}$ $K_{1,2}$ $K_{1,3}$
$K_{1,1}$ $K_{1,2}$ $K_{1,3}$ $K_{1,4}$ $K_{1,5}$ $K_{1,6}$
$K_{2,1}$ $K_{2,2}$ $K_{2,3}$ $K_{2,4}$ $K_{2,5}$ $K_{2,6}$
$K_{3,1}$ $K_{3,2}$ $K_{3,3}$ $K_{3,4}$ $K_{3,5}$ $K_{3,6}$
$K_{4,1}$ $K_{4,2}$ $K_{4,3}$ $K_{4,4}$ $K_{4,5}$ $K_{4,6}$
$K_{5,1}$ $K_{5,2}$ $K_{5,3}$ $K_{5,4}$ $K_{5,5}$ $K_{5,6}$
$K_{6,1}$ $K_{6,2}$ $K_{6,3}$ $K_{6,4}$ $K_{6,5}$ $K_{6,6}$
$[K_{2,1}\ K_{2,2}\ K_{2,3}]^T$
$K_{1,1}$ $K_{1,2}$ $K_{1,3}$
$K_{2,1}$ $K_{2,2}$ $K_{2,3}$
$K_{3,1}$ $K_{3,2}$ $K_{3,3}$
Element stiffness matrix
$K_{3,1}$ $K_{3,2}$ $K_{3,3}$

Figure 4.3: Arbitrary insertion of element stiffness matrix $\boldsymbol{k_e}$ from a T3 element (with node 1, 5, and 6) into a global stiffness matrix $\boldsymbol{K}$

Once $\boldsymbol{K}$ and $\boldsymbol{R}$ are determined, we need to apply boundary conditions to the system of equations. For our implementation we have not considered any symmetry, and one (arbitrary) node needs to be constrained. The function *constrainDOF()* does this by zeroing out the last row and column in the stiffness matrix, then setting the lower right entry of $\boldsymbol{K}$ and the last entry of $\boldsymbol{R}$ to one, leaving the system on the following form (for a mesh with $n+1$ nodes)

$$\mathbf{Kr} = \mathbf{R} \quad \Longrightarrow \quad \begin{bmatrix} K_{11} & K_{12} & \cdots & K_{1n} & 0 \\ K_{21} & K_{22} & \cdots & K_{2n} & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ K_{n1} & K_{n2} & \cdots & K_{nn} & 0 \\ 0 & 0 & \cdots & 0 & 1 \end{bmatrix} \begin{bmatrix} r_1 \\ r_2 \\ \vdots \\ r_n \\ r_{n+1} \end{bmatrix} = \begin{bmatrix} R_1 \\ R_2 \\ \vdots \\ R_n \\ 1 \end{bmatrix}$$

After the boundary condition is applied, the system is solved for the displacements $\boldsymbol{r}$ (equation solving is described in Section 4.6). With the global displacements computed, **Mesh** moves the corresponding displacements down to the element level. **Mesh** then computes the torsional stiffness and stress distributions by calling *calculateDeltaGIt()* and *computeTauTorsion()* for each element. For triangular elements the stress values are computed directly at the nodes. For quadrilateral elements on the other hand, the stresses are calculated at the integration points before being extrapolated to the nodes. The extrapolation is done through the extrapolation matrix $\boldsymbol{E}$ initiated by the function *setExtrapolationMatrix()*. Next, the shear center due to torsion is calculated by calling *calculateTorsionShearCenter()*.

Multiple elements can contain the same node. Because of this, the stress is calculated multiple times for each node. The stress values might also differ slightly in numerical values, depending on the element they are calculated from. To produce a smooth stress output we make use of nodal point averaging, which sets the nodal stress value to an average of all the calculated values. The **Mesh** function *setSmoothNodalTau()* implements this technique, thereby ensuring a continuous stress field (even for T3 elements). The averaged shear stresses are then stored in the matrix **tauTorsion**.

4.4 Shear analysis

As for torsion analysis, **Mesh** iterates over all elements and computes the individual element loads. For shear analysis this calculation is separated into two cases, one for each shear force (V_x and V_y). The respective element loads are computed by calling *calculateElementLoadShearX()* and *calculateElementLoadShearY()* (using Eq. (2.66)). The rest of the procedure is very similar to that of the torsion analysis. Again the element loads are summed up to the global load vector $\boldsymbol{R}$. Since CCSC currently does not offer support for symmetry, the boundary condition established in Section 4.3 is valid for both torsion and shear analysis. Because of this we do not have to recompute the stiffness matrix, and we can reuse $\boldsymbol{K}$ from the torsional analysis. The global system of equations ($\boldsymbol{Kr} = \boldsymbol{R}$) is solved twice for the displacement vectors $\boldsymbol{r}$, one time for each load case. The displacements are again moved to the element level, before the shear stresses are calculated by calling *computeTauShear()* for all the elements (using Eq. (2.68)).

The stresses are smoothed out through *setSmoothNodalTau()* (as done in the torsion analysis) and stored in the matrices **tauShearX** and **tauShearY**. At the end of the shear analysis **Mesh** determines the shear deformation factors. This is done by making calls to *calculateShearDeformationFactorX()* and *calculateShearDeformationFactorY()*, each of which loops through all elements and sums up their respective contributions. Supporting symmetry can lead to substantial savings of computational effort, but was not a focus point of our thesis. If we want to take advantage of it in the future the stiffness matrix would have to be computed four times, one time for the torsion analysis, one time for each of the shear load cases, and one time for composite analysis.

4.5 Analysis of composite materials

The implementation of composite materials unifies torsion and shear loading into one analysis. Again, **Mesh** is responsible for executing the analysis. It iterates over all elements and uses the function *calculateElementLoadComposite()* to calculate the element load vectors (using Eq. (2.104b)). The contribution to the initial strain due to torsion is dependent on the rate of twist θ, which was computed during regular torsion analysis. Rather than recomputing it, we simply reuse the stored value.

The analysis proceeds by establishing the global load vector $\boldsymbol{R}$. The pre-established global stiffness matrix $\boldsymbol{K}$ can be reused. We solve the equilibrium equation ($\boldsymbol{Kr} = \boldsymbol{R}$) which determines the nodal displacements. After moving the global displacements to the element level, **Mesh** makes a call to *computeTauComposite()* for the calculation of the stresses based on Eq. (2.105). The stresses are calculated at the integration points, before they are extrapolated out to the nodes (for quadrilateral elements). The results are then smoothed, before being stored in the matrix **tauComposite**. Unlike for the torsion and shear analyses, the axial stress $\boldsymbol{\sigma_z}$ is computed as a part of this analysis (in addition to the shear stresses $\boldsymbol{\tau_{xz}}$ and $\boldsymbol{\tau_{yz}}$). The axial stress is stored in the first column, while the shear stresses are stored in the second and third column.

4.6 Eigen

In order to solve the equilibrium equation described in Section 2.7, CCSC uses an open source linear algebra template library for C++ called Eigen. This library is used for matrix operations throughout the application, as well as for equation solving. For the analysis of massive cross-sections we have chosen to use Eigen's *SimplicalLDLT* solver, which performs an LDLT (Cholesky) decomposition in order to solve the equations. The solver was chosen based on the testing performed by Kristian Strømstad during his master thesis[1]. FEM has the advantage that it produces a positive definite and sparse stiffness matrix, which makes this solver ideal for our case as they are both requirements for the SimplicalLDLT solver. The equation solving is done as follows.

```
SimplicialLDLT<SparseMatrix<double>> LDLT(K);
r = LDLT.solve(R);
```

We have sometimes encountered stiffness matrices that are not positive definite. When this is the case, the system of equations is solved using Eigen's ColPivHouseholderQR solver. Having a non-positive definite stiffness matrix indicates that something is wrong, and the results obtained should be interpreted with great caution. This has been an issue since the original version of CCSC (developed by Strømstad), and unfortunately we have not had the time to investigate the issue further.

As the mesh size gets bigger, so does the memory usage of Eigen's solver. Before inserting non-zero elements into a SparseMatrix we need to reserve enough memory for a certain number of elements in order to prevent unnecessary memory reallocations. Consequently the maximum mesh size of CCSC depends on the memory of the users computer. Currently CCSC reserves space for a fixed 1% of the matrix elements. This means that the system theoretically needs a minimum free space of

$$memory = \frac{n \times n \times 8 \times 0.01}{1024^2} MB \tag{4.2}$$

where n is number of nodes, 8 is number of bytes for a double, and 1024 is the number of bytes in a kilobyte. As the mesh is further and further refined, the stiffness matrix becomes increasingly sparse. Because of this we could implement a dynamically sized allocation. However, for all intents and purposes the current solution should be able to run most meshes on any modern computer. Therefore we chose not to spend any time implementing this option.

4.7 ParaView visualization

The results can be visualized in any program supporting the VTK file format. For our project we used the open source program ParaView. To make validation of the massive cross-sections easier, we wrote a python script for ParaView that visualizes the principal axis (located in area center) and the shear center. The script is named *DisplayCoordinateSystem* and is found under the *Scripts* folder. By adding this to a ParaView macro, the original axes and shear center can be viewed by simply pressing a button. It is only used as a crude way of validating the shear center, therefore we did not put a lot of effort in to making the visualization more visually appealing.

4.8 Meshing cross-sections using Gmsh

CCSC currently only supports mesh files that are of the type *.msh*, which is generated by Gmsh 2.11. Gmsh is an open source finite element mesh generator distributed under the GNU General Public License (GPL). We used it to generate meshes with both three- and six-node triangular elements, as well as four- and nine-node quadrilateral elements. There is currently no implementation for meshing in CCSC, and the user is responsible for creating valid mesh files. When creating a mesh for CCSC, it is important to ensure that the nodes are arranged in a counter-clockwise fashion for each element. Failing to do so may result in inverted parameters and stress distributions.

Chapter 5

Implementation - Thin-Walled Cross-Sections

In this chapter we take a closer look at how the theory of thin-walled cross-sections presented in Chapter 3 is implemented in CCSC. This includes descriptions of the class structure, informational flow, numerical integration, equation solving, and stress analyses from applied forces. As with the massive implementation, this chapter is supplemented by an extensive documentation containing all functions and their underlying theory. This is available as a part of the program package, under the *Documentation* folder.

5.1 Classes

The thin-walled implementation of CCSC consists of 8 classes: **LineMesh**, **CycleDetection**, **LineElement**, **LineParser**, **LineResultWriter**, **Material**, **Solver**, and **Node**. The latter three classes are the same as used in the massive implementation described in Section 4.1, and will not be described any further. Thin-walled cross-sections differ a lot from massive cross-sections, both in terms of theory, and in terms of implementation. As a consequence we decided to separate the two implementations by creating entirely new classes, rather than using any inheritance. The relationship between classes for the thin-walled implementation can be seen in Fig. 5.1.

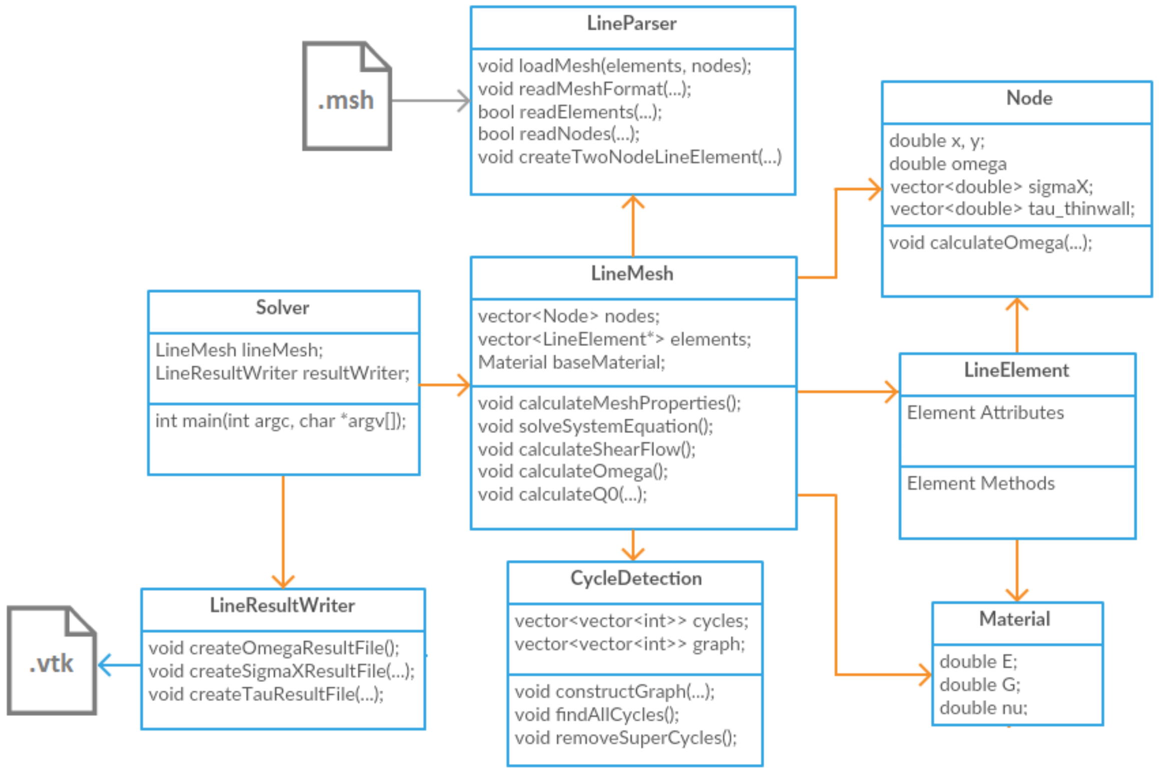

.msh
LineParser
void loadMesh(elements, nodes);
void readMeshFormat(...);
bool readElements(...);
bool readNodes(...);
void createTwoNodeLineElement(...)
Node
double x, y;
double omega
vector<double> sigmaX;
vector<double> tau_thinwall;
void calculateOmega(...);
Solver
LineMesh lineMesh;
LineResultWriter resultWriter;
int main(int argc, char *argv[]);
LineMesh
vector<Node> nodes;
vector<LineElement*> elements;
Material baseMaterial;
void calculateMeshProperties();
void solveSystemEquation();
void calculateShearFlow();
void calculateOmega();
void calculateQ0(...);
LineElement
Element Attributes
Element Methods
.vtk
LineResultWriter
void createOmegaResultFile();
void createSigmaXResultFile(...);
void createTauResultFile(...);
CycleDetection
vector<vector<int>> cycles;
vector<vector<int>> graph;
void constructGraph(...);
void findAllCycles();
void removeSuperCycles();
Material
double E;
double G;
double nu;

Figure 5.1: UML class diagram showing the relationship between classes for thin-walled implementation

CycleDetection

In order to perform analysis on a thin-walled cross-section, we need an algorithm to detect all *cells* (i.e. cycles not containing other cycles). The first step to detect these cells is to find all existing cycles in the cross-section. To solve this problem we used a recursive depth-first search algorithm (DFS). When a DFS loops back on an already visited node (and the length of the path taken is more than two) it is called a *back edge*. If a back edge is found, we know that the path taken is a cycle. By performing DFS from all the nodes in the cross-section sequentially, we can ensure that all cycles are detected. Our implementation of the DFS is a modified version of the one presented in Axel Kempers post on Stack Overflow[19]. The algorithm has been validated through extensive testing on multiple cross-sections. The following is a rough pseudocode of the algorithm:

Algorithm 4 Algorithm to find all cycles

```
/* cycles is an empty vector of cycles and findNewCycles(path) is called from all nodes
sequentially, where path is (originally) a vector ontaining only the start node id æ*/
function FINDNEWCYCLES(vector<int> path)
    startNode = path[0];
    for (all edges in cross-section) do
        nextNode = neighbor node of startNode
        if (nextNode not visited ) then
            add nextNode to path
            findNewCycles(path)
        else if (path > 2 and nextNode is last node in path) then
            // Cycle is detected
            if ( cycles does not already contain path or the inverse of path) then
                add path to cycle
            end if
        end if
    end for
end function
```

For the analyses we are only interested in the cells, and consequently we need to remove all *super-cycles* (i.e a cycle containing other cycles), from the list of cycles. In Fig. 5.2 a simple thin-walled cross-section is illustrated, with its two cells colored in blue, and the super-cycle colored in red. Algorithm 5 shows how we detect and remove all super-cycles.

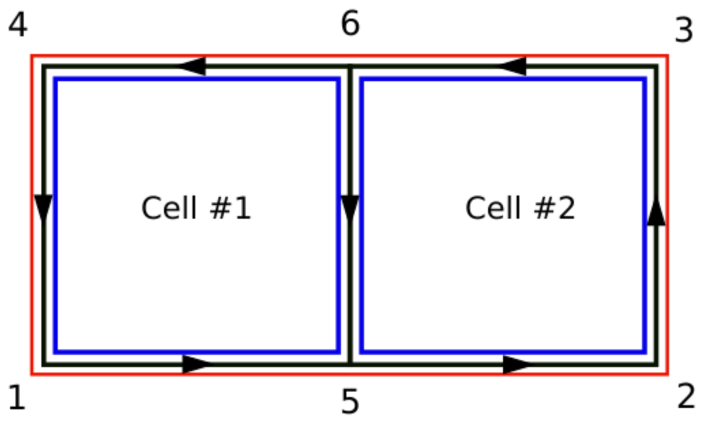


Figure 5.2: Simple thin-walled mesh, containing two cells (blue coloring) and one super-cycle (red coloring). Direction of the elements are included

Algorithm 5 Algorithm to remove super-cycles

```
Maintain a two dimensional vector of the cycles and their edges and call it cycles
while (there exists an unchecked cycle in cycles) do
    vector<Edge> current = edges of first unchecked cycle in cycles
    vector<Edge> other = edges of all cycles other than current
    if (all edges in current exist in other ) then
        //current is super-cycle
        remove current from cycles
    end if
end while
```

In order to solve the system of equations ($\boldsymbol{Bp} = \boldsymbol{a}$), we need to calculate the area of each cell. To simplify this calculation, we flip all elements of the cycle in question so that they are facing the same direction (either clockwise or counter-clockwise). Now the area below each element is either added or subtracted based on the x-direction of the element. This is all done inside the *calculateCellArea()* function. It should be noted that the area of a cell will be negative if the elements in a cycle are oriented counter-clockwise. Fig. 5.3 illustrates the area calculation procedure.

The same function ensures a positive (counter-clockwise) orientation of each cell. If, however, an element is part of two cycles it is impossible for it to have a positive direction for both cells. In these cases the element is given a positive direction according to the last cell for which the area is calculated. The pseudocode for the function is given in Algorithm 6.

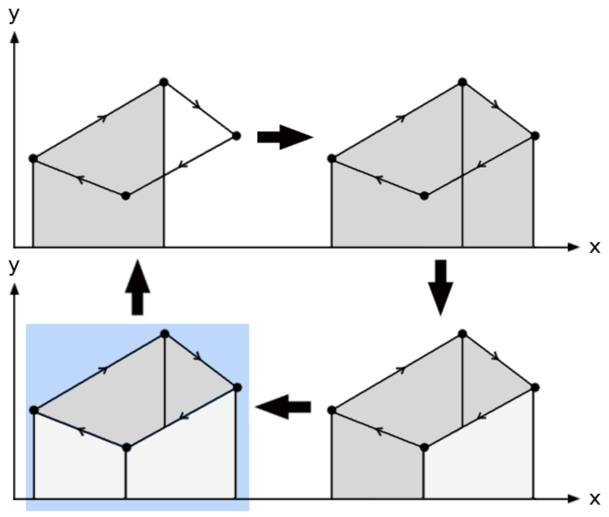


Figure 5.3: Algorithm to calculate the area of an irregular polygon

Algorithm 6 Algorithm to calculate area and ensuring counter-clockwise direction for a cell (done sequentially for all cells)

```
Maintain a two dimensional vector containing all cycles and their respective edges **cycles**
Maintain a one dimensional vector of doubles containing the areas of the cells and call it **areas**
for (every cycle c in cycles) do
    Create a one dimensional vector containing the edges of a single cycle and call it cycle
    cycle = cycles[c]
    assure uniform direction of all edges in cycle
    double area = 0
    for (every edge in cycle) do
        double $x_1$ = xCoordinate of first node
        double $y_1$ = yCoordinate of first node
        double $x_2$ = xCoordinate of second node
        double $y_2$ = yCoordinate of second node
        area += $\frac{(y_1+y_2)}{2} \cdot (x_2 - x_1)$
    end for
    if (area > 0) then
        //Direction of cell is clockwise
        reverse cycle and flip all elements in cycle
    else
        //Direction of cell is counter-clockwise, and area has a negative value
        area *= -1
    end if
    push area to areas
end for
```

LineMesh

LineMesh is the class where the thin-walled stress analysis takes place. It represents the entire cross-section by storing all the elements and nodes as vectors. These vectors are then passed as pointers to the **LineParser**, where they are filled up according to the mesh given by the user. By initializing a **CycleDetection** object and passing the vector of elements to it, the cells and their corresponding areas are found. The analysis can now start. First the basic cross-sectional stiffnesses are determined through the *calculateMeshProperties()* function. This includes the computation of bending stiffnesses and the angle of principal axis. For the rest of the calculations the cross-section is transformed to coincide with the principal axis. This is done inside the function *transformCoordinates()*.

The modified shear flows of each element can now be determined by solving the system of equations defined in Eq. (3.4). The area vector $\boldsymbol{a}$ has already been determined, and the coefficient matrix $\boldsymbol{B}$ is constructed by calling the function *constructBMatrix()*. The algorithm for constructing $\boldsymbol{B}$ is shown in Algorithm 7.

Algorithm 7 Algorithm to construct **B**-matrix

```
MatrixXd B
for (i in all cycles) do
    for (j in all cycles) do
        if (i == j) then
            for (all elements in cycle) do
                diagonalSum += length / (shearModulus * thickness)
            end for
            B(i,j) = diagonalSum
        else
            for (all common elements of cycle[i] and cycle[j]) do
                offDiagonalSum -= length / (shearModulus * thickness)
            end for
            B(i,j) = offDiagonalSum
        end if
    end for
end for
```

The system of equations is now solved for the shear flow $\boldsymbol{p}$. Once determined, the ω - diagram can be established. This is done through the function *calculateOmega()*, which is described in detail in Section 5.3. When the ω - diagram is determined, the axial stresses may be computed by *calculateAxialStress()*. Next, the shear stresses are found by invoking *calculateTau()* (further described in Section 5.4). All results are stored in **LineMesh** for later use by **LineResultWriter**.

LineElement

This class represents the line segments of the cross-section. All parameters needed for the thin-walled analyses are stored here. This includes material properties, nodes, area, shear center, and shear flow values. In order for **LineMesh** to calculate the global stiffnesses and parameters, the individual element contributions are computed and added up to their global values.

LineResultWriter

This class is responsible for outputting results of the thin-walled analysis. If the desired output is linear, *writeLinearMeshTopology()* is called. In these cases the graph of each element is visualized as two triangular VTK polygons. If the graph has an intersection point along the element, the two polygons each have a corner node in the intersection point (as illustrated in Fig. 5.4). In the cases with no intersection points the two triangles are simply stacked on top of each other (as seen in Fig. 5.5). For further description of how these polygons are defined in the VTK file format, see Appendix C.4.

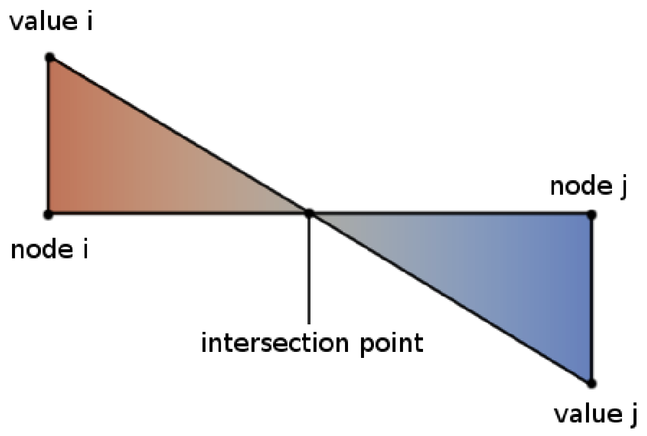


Figure 5.4: Linear diagram with an intersection point along the element. The two triangular polygons are colored based on the corresponding function value

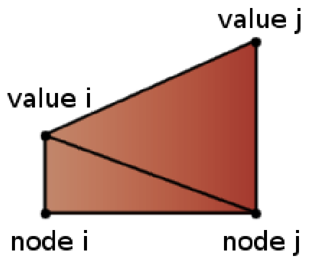


Figure 5.5: Linear diagram in CCSC without an intersection point along the element. The two triangular polygons are colored based on the corresponding function value

For quadratic functions the output is more comprehensive. In order to approximate the graphs, each element is divided into sets of equally large line segments. Each of these extra segments are treated in the same way as an element with linear variation, which means that the graph along the segment is approximated using two triangular polygons. The number of line segments is defined by the user (with a minimum of seven), and the more segments are used the better the approximation will be. An example of this method can be seen in Fig. 5.6. The quadratic output is created inside the function *writeQuadraticMeshTopology()*. For details on how the polygons are defined in the VTK format, see Appendix C.4

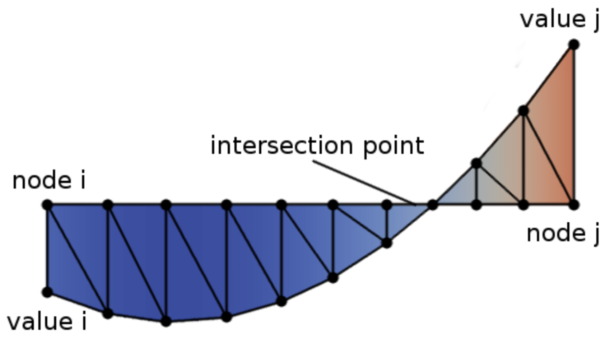


Figure 5.6: Quadratic diagram in CCSC made from numerous triangular polygons. Each polygon is colored based on the corresponding function value

The direction of each element is needed in order to interpret the results of our calculations. Therefore a *Glyph arrow* showing the element direction is drawn. These arrows are only visible after applying both a glyph filter and modifying several parameters inside of ParaView. For the convenience of the user, we have included a python script named *DisplayArrows*, which does this automatically. The user only has to add the script as a ParaView macro, and run the macro (all scripts are found under the *Scripts* folder of CCSC).

Because of the way polygons work in ParaView, we have not found a way of including all the distributions within one *.vtk* file. Therefore **LineResultWriter** creates a new sub-folder inside the *Results* folder, which is filled up with multiple result files (one for each diagram). In addition to the visualizations made in ParaView, **LineResultWriter** outputs a *.txt* file made by the *createTextFile()* function. This file contains the material properties, applied loads, and cross-sectional parameters. It is neatly organized, and makes it very simple to find the desired parameters. One example of such a file is given in Fig. B.8 in the Appendix.

LineParser

LineParser parses the thin-walled mesh file given as an input argument by the user. A thin-walled mesh only contains two-node segments (line elements) with an assigned thickness and an isotropic material. These properties are specific for each element, and can either be set to default values or user defined values. The parser differentiates between the two cases by the *element type*-tag found in the individual element records in the *.msh* file. This tag can be set as one of the following

- tag set as 1 - the element instantiated is given the default property values. The material properties are set to match the base material, while the thickness is set to $5mm$
- tag set as 4 - the element instantiated is given unique property values

If the element in the *.msh* file is given the tag 4, the Young's modulus (E), Poisson's ratio (v), and the element thickness (t) have to be added to the end of the record (in that order). If the element is given the tag 1 (as is standard in Gmsh for two-node elements), the record is left as is. By failing to obey these rules, the analysis will fail. Fig. C.2 in Appendix C shows the format of the *.msh* file.

Solver, Material and Node

These classes are common to both massive and thin-wall analysis, and are described in Section 4.1.

5.2 Numerical integration

For thin-walled analysis we make use of both two- and three point uni-directional integration schemes presented in Section 3.10. Algorithm 8 presents the simple procedure of numerical integration for line elements.

Algorithm 8 Algorithm for numerical integration of line elements

```
// f(zeta) is a function to be integrated over the area
double I = 0
for (all Gaussian points i) do
    I += weight[i]*f(zeta[i]) * (area /2)
end for
```

5.3 Axial stress analysis

The axial stress distribution is dependent on the basic cross-sectional parameters computed in **LineMesh**. After they are determined, the ω_c - diagram is computed for each element as described in Algorithm 9.

Algorithm 9 Algorithm to create ω_c - diagram

```
Sort the vector of elements in a depth-first manner (DFS), call it **elements**
The $\omega_c$ - value of the first node of **elements** is set to zero
**for** (all elements e in **elements**) **do**
    **if** (second node of e not visited) **then**
        secondNode.omega_c = firstNode.omega_c + delta_omega_c // (see Eq.(3.14))
        secondNode.visited = true
    **end if**
**end for**
// This will assign $\omega_c$ - values for all nodes
```

The axial analysis continues with three steps. First, the shear center is calculated by calling the function *calculateShearCenter()* (based on Eqs. (3.16 and 3.17)). Secondly, the ω_c - diagram is transformed around the shear center to produce the ω - diagram. The function *calculateOmega()* performs this operation using Eq. (3.15). Finally, the axial stresses are computed by *computeAxialStress()* (which uses Eq. (3.19). The axial stress values have a linear variation over the element, and is therefore outputted by **LineResultWriter** using *writeLinearMeshTopology()*.

5.4 Shear stress analysis

In order to compute the shear stress distribution over the cross-section, we must determine the shear flow distribution of each element. We find the shear flow at the mid-point of each element using Eq. (3.36). For this we need to develop the coefficient matrices $\boldsymbol{A}_1$ and $\boldsymbol{A}_2$ as well as the matrices $\boldsymbol{G}_1$ and $\boldsymbol{G}_2$. This is done in **LineMesh** by calling *calculateA1Matrix()*, *calculateA2Matrix()*, *calculateG1Matrix()*, and *calculateG2Matrix()* respectively. The mid-point shear flows can now be determined by invoking *calculateQ0()* for all the load cases. Once $\boldsymbol{q}_0$ is found, the shear flow is fully defined along the element through Eq. (3.24). It should be noted that the axial force is assumed to be constant over the cross-section, and consequently N' is zero.

The stresses at the nodes are now found by *calculateNodalTau()*, which is called one time for each loading case. Since the shear stresses have a quadratic variation along the element, computing only the nodal values is not sufficient. To approximate the quadratic distribution we have given each line element a set of extra points with equal spacing (through the function *addElementPoints()*). The stress values for each of these points are then found by passing its local element position (s) to the *calculateQs()* function. The nodal stress values are then stored inside **Node**, while the extra stress values are stored inside **LineElement**. Intersection points are also found by calling the *findElementIntersections()* function. Fig. 5.7 illustrates how the shear stress values are stored in CCSC. Lastly, the shear stress purely due to St. Venant torsion is determined through *calculateTauSTV()*.

During our validation of the thin-walled analysis we found that all shear stress values were inverted compared to CrossX. Because of this, we have forced the load vector ($\boldsymbol{f}$) inside *initializeMatrices()* to have a negative sign. By doing this, the values match with CrossX. A detailed validation of the thin-wall analysis is given in Section 6.4. After finishing the shear analysis, **LineMesh** computes the shear deformation factors by making calls to the functions *calculateKappaX()* and *calculateKappaY()*. Their implementation is based on Eq. (3.45).

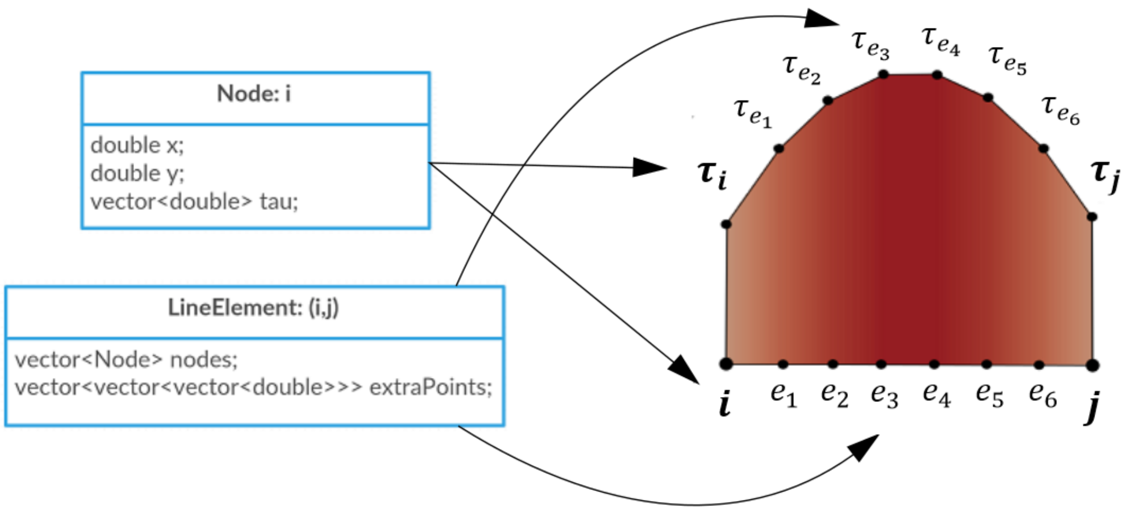


Figure 5.7: Relationship between results and output for an arbitrary line element with seven line segments (100 is set as default). Nodal shear stress values are stored in **Node**, while extra points and their shear stress values are stored in **LineElement**

5.5 Eigen

In the implementation of thin-walled cross-sections we decided to use Eigen's *HouseholderQR* to solve Eqs. (3.4 and 3.36). This solver uses a QR decomposition, and was chosen for robustness. Unlike SimplicalLDLT used for massive cross-sections, this solver has no requirements for the coefficient matrix (such as sparseness or positive definiteness). The solving is done as follows

```
HouseholderQR<MatrixXd> solver(B);
p = solver.solve(a);
```

5.6 Meshing cross-sections using Gmsh

For thin-walled analysis the meshes can be created using Gmsh. The meshes generated this way will only contain regular two-node line elements with default property values (see Section 5.1). If the user wants to define line elements with specific material properties, he/she must change the mesh file manually according to specifications of the *.msh* file format (see Appendix C.1). The *element type*-tag must be changed to 4, and Young's modulus, Poisson's ratio, and element thickness must be added to the end of the desired element record. If the element type tag is 4 (specific line element) and the three last properties are not defined, the analysis will fail. This is also the case if the *element type*-tag is set to 1 (regular line element), and the three element properties are added to the end of the record. The two element types can, however, be used interchangeably within the same mesh. As there is no meshing of thin-walled cross-sections inside of CCSC it is the users responsibility to create valid mesh files.

Chapter 6

Validation and Results

This chapter presents a validation of CCSC. We compare the outputted results with well-known solutions, as well as results retrieved from CrossX. This includes comparisons of cross-sectional parameters and stress distributions from massive as well as thin-walled analysi. The different element types supported by CCSC are tested, both for their accuracy and for their rate of convergence. Lastly, we investigate the running times of the application for numerous refinement levels.

6.1 Units and output

The cross-sections should be modeled in millimeters [mm] as the applied forces are given in newton [N]. This means that the stresses are computed in MPa, and all the cross-sectional parameters are given in terms of N and mm. The default applied loads have been chosen to match with CrossX:

- N = $V_x = V_y = 10^3\ N$
- $M_x = M_y = M_z = T_{stv} = T_{wrp} = 10^6\ Nmm$
- Bimoment = $10^9\ Nmm^2$

6.1.1 Stress component output for massive cross-sections

- Effective stress (von Mises)
- σ_z due to N, V_x, V_y, M_x and M_y
- σ_z resultant
- τ_{xz} and τ_{yz} due to V_x, V_y and M_z
- τ_{yz} and τ_{yz} resultant
- τ resultant
- σ_z, τ_{xz}, and τ_{yz} from composite analysis

6.1.2 Stress component output for thin-walled cross-sections

- Effective stress (von Mises)
- ω
- σ_z due to N, M_x, M_y and B
- σ_z resultant
- τ due to V_x, V_y, T_{stv} and T_{wrp}
- τ resultant

6.2 Validation of fundamental parameters

We have performed numerous analyses for different load cases in order to validate the results outputted by CCSC. The results for known (standard steel cross-sections) and unknown load cases (complex cross-sections) are considered, and both regular results as well as results from composite analysis are examined. The fundamental parameters area, second moments of area and the torsion constant have all been validated for CCSC. The torsion constant is especially important, as it is derived from the shape functions, the Jacobian, and the strain-displacement matrix, which makes it a very good indication for the validity of our solution.

First we analyzed the well-known steel beam profile IPE300. We designed a mesh using T6 elements (massive cross-section) and a mesh using line elements (thin-walled cross-section). Corresponding meshes were made in CrossX for comparison, and were matched on the number of nodes (as closely as we could manage). For the massive cross-sections we approximated the corner fillets with four straight line segments. In addition to CrossX, we also compared the results to the exact solutions listed in "Stålkonstruksjoner - profiler og formler"[20]. The massive meshes had 749 and 777 nodes for CCSC and CrossX respectively, whereas the thin-walled meshes were identical. Table 6.1 summarizes this analysis.

Table 6.1: Cross-sectional parameters of IPE300 (values gathered from CrossX and "Stålkonstruksjoner - profiler og formler"[20])

IPE300	**Massive**		**Thin-wall**		
Parameters	**CCSC**	**CrossX**	**CCSC**	**CrossX**	**Exact value**
$\boldsymbol{A} \cdot 10^{-3} \quad [mm^2]$	5.38574	5.3857	5.26403	5.264	5.38
$\boldsymbol{I}_x \cdot 10^{-6} \quad [mm^4]$	83.6411	83.641	81.4907	81.521	83.6
$\boldsymbol{I}_y \cdot 10^{-6} \quad [mm^4]$	6.03843	6.0384	6.01875	6.0274	6.04
$\boldsymbol{I}_t \cdot 10^{-6} \quad [mm^4]$	0.198414	0.19877	0.157019	0.15702	0.202
$\boldsymbol{\kappa}_x$	1.56981	1.56989	1.96786	1.96223	N/A
$\boldsymbol{\kappa}_y$	2.56851	2.56854	2.57862	2.57668	N/A

Next we computed the same parameters for RHS 100 × 50 × 4, which is another well-known steel beam profile. For the massive cross-section the outer boundary of the corners were approximated using four line segments, while the inner boundaries were made up of two segments. This cross-section is of interest because it contains a cell, which is an important feature for the thin-wall analysis (as explained in Chapter 3). For the massive meshes we used 303 nodes and 440 nodes for CCSC and CrossX respectively. Again, the thin-walled meshes were identical. Table 6.2 summarizes the analysis for the RHS cross-section.

Table 6.2: Cross-sectional parameters of RHS 100×50×4 (values gathered from CrossX and "Stålkonstruksjoner - profiler og formler")

RHS	Massive		Thin-wall		
Parameters	**CCSC**	**CrossX**	**CCSC**	**CrossX**	**Exact value**
$\boldsymbol{A} \cdot 10^{-3}\ \ [mm^2]$	1.13034	1.1303	1.10492	1.1049	1.13
$\boldsymbol{I}_x \cdot 10^{-6}\ \ [mm^4]$	1.42183	1.4218	1.36372	1.3642	1.42
$\boldsymbol{I}_y \cdot 10^{-6}\ \ [mm^4]$	0.467666	0.46767	0.453604	0.45461	0.467
$\boldsymbol{I}_t \cdot 10^{-6}\ \ [mm^4]$	1.15854	1.1543	1.12016	1.1202	1.13
$\boldsymbol{\kappa}_x$	3.91455	3.94419	2.92165	2.90876	N/A
$\boldsymbol{\kappa}_y$	1.51973	1.52334	1.48398	1.48296	N/A

It is clear from the two analyses that the results outputted by CCSC are very close to that of CrossX. This is what we expected, as our implementation is based on the same theory. Both programs calculate area and second moments of area exact to three digits for massive cross-sections. CCSC computes a torsional constant that differs from the exact solution by 1.78% for IPE300 and 2.53% for RHS. This is only slightly more than CrossX (1.60% for IPE300 and 2.15% for RHS).

For the thin-wall analysis the calculated results are further away from the exact solution. This is mainly because there is no mechanism to model the fillets of the cross-sections. Therefore the area (and the two second moments of area) is calculated to be smaller than reality. For the calculation of the torsion constant of IPE300, the thin-wall analysis is further away from the exact value than the massive analysis. The opposite is the case for the RHS cross-section, which is most likely incidental. We also recognize that the computation of κ_x seems to differ more (0.44%) between the two programs, than κ_y (0.06%). Fig. 6.1 illustrates the two meshes used for massive analysis in CCSC.

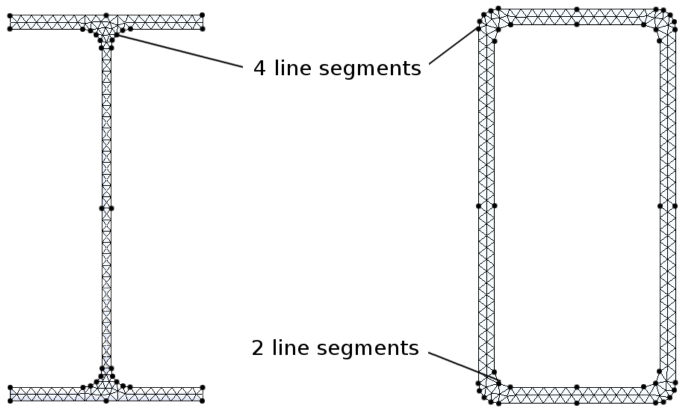


Figure 6.1: Mesh of the IPE300 and RHS 100×50×4 cross-sections used for massive analysis in CCSC

6.3 Validation of massive cross-sections

Further validation of the massive cross-section implementation in CCSC is needed. In this section we take a closer look at stress distributions, composite analysis, element performance, and mesh convergence, in an effort to further validate the results.

6.3.1 Stress distributions for massive cross-sections

To further validate our results, we compared the stress distributions for massive cross-sections obtained from CCSC to those of CrossX. We considered a rectangular cross-section with dimensions $200\times100\ mm^2$, which was chosen due to its rather simple cross-section, providing intuitive stress distributions. Both CCSC and CrossX were meshed identically with T6 elements and 2145 nodes.

The first thing we considered was shear stress due to applied shear loads. There are four general shear load cases, all of which seem to comply with CrossX (both in terms of stress distributions as well as numerical values). Figs. 6.2 and 6.3 show comparisons for two of these cases. Fig. 6.2 depicts the same distribution and virtually identical extreme values (a difference of only 0.09%) between the two programs. The case shown in Fig. 6.3 shows a reversed stress distribution. This is due to the cross-sections being rotated differently (as was discussed in Section 4.1). The numerical values are further off for this case, but CCSC is still within a 0.45% margin of CrossX.

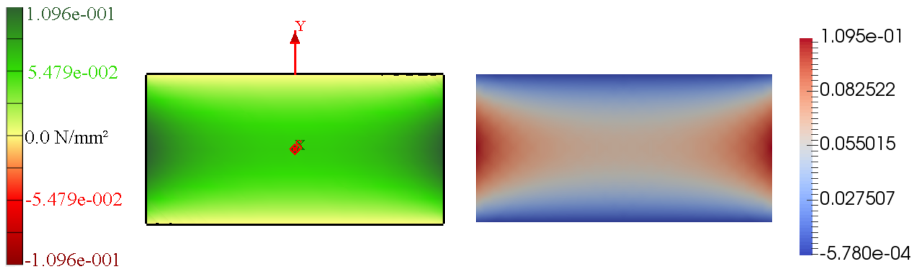


Figure 6.2: Comparing τ_{yz} due to V_y (vertical shear force). CrossX on the left, CCSC on the right

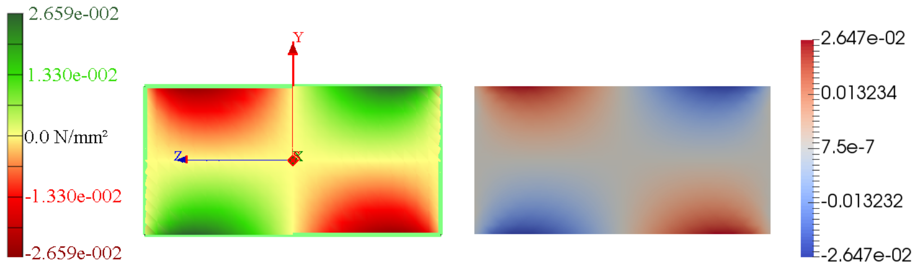


Figure 6.3: Comparing τ_{xz} due to V_y (vertical shear force). CrossX on the left, CCSC on the right

For the case of applied torsion, there are two distinct solutions. Both cases compute virtually identical stress distributions and numerical values between CCSC and CrossX. The case of τ_{yz} due to applied torsion M_z is shown in Fig. 6.4. For this case CrossX outputs an extreme value of 1.620 MPa, compared to the 1.617 MPa of CCSC, which is only a difference of 0.19%.

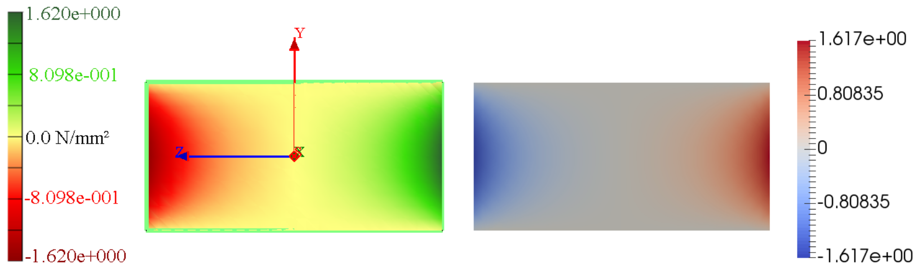


Figure 6.4: Comparing τ_{yz} due to torsion M_z. CrossX on the left, CCSC on the right

As expected, both the stress distribution as well as the numerical values are pretty close for the effective stress, as illustrated by Fig. 6.5. In fact the numerical extreme values from CrossX and CCSC are identical to each other. Again we observe that the distributions are reversed compared to each other. As mentioned previously this is because the cross-sections are rotated in opposite directions of each other (CrossX is rotated 90° while CCSC is rotated 0°).

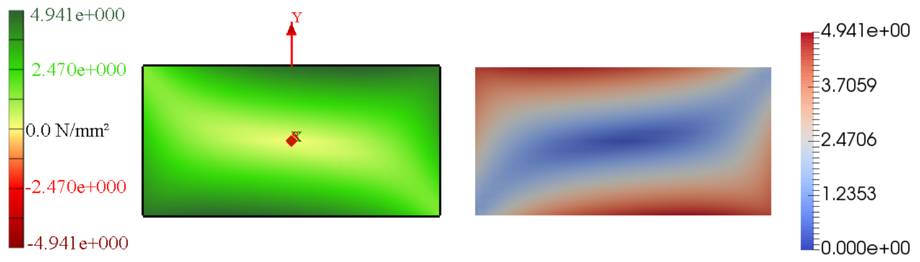


Figure 6.5: Comparing effective (von Mises) stress. CrossX on the left, CCSC on the right

In addition to the rectangle, we have compared a vast array of other cross-sections, all of which seem to comply with CrossX for both the stress distributions and parameters (the different cross-sections analyzed are given in Appendix D.2). This includes the calculation of shear center from torsion loading. It was mentioned in Section 2.13 that cross-sections with very large stress concentrations (singularities) can be produced. For these cases the discrepancies between the extreme numerical values might differ more. A common case would be when sharp re-entrant edges are introduced. A way to circumvent this problem is to provide fillets with a large radius.

6.3.2 Analysis of composite materials

It was stated in Section 2.11.3 that CCSC can not yet determine stress distributions for composite materials. But the implementation is still of interest because it can solve for torsion and shear loading in a unified analysis, rather than two separate analyses. For this reason the implementation will be the basis for the continued development of composite cross-section analysis (hence the name *composite analysis*) and therefore it needs validation.

If the constitutive matrix for isotropic materials defined in Eq. (2.107) is used for the composite analysis, we can make comparisons to the results provided by the torsion and shear analysis as they should be the same. We continue to use the rectangular cross-section from Section 6.3.1. We compare τ_{xz} obtained from composite analysis to the τ_{xz} resultant from regular analysis (and likewise for τ_{yz}). The comparisons are presented in Table 6.3.

Table 6.3: Comparing shear stress resultants for regular and composite analysis. Values are in MPa

Stress resultants	$\tau_{xz,max}$	$\tau_{xz,min}$	$\tau_{yz,max}$	$\tau_{yz,min}$
Regular analysis	2.112	−1.956	1.724	−1.506
Composite analysis	2.112	−1.956	1.724	−1.506

The comparison indicates that the composite analyses is indeed solved correctly. We can not verify that σ_z is identical, as the resulting σ_z outputted by regular analysis includes axial force and applied bending moments. However, comparing the σ_z distribution due to only shear loading gives identical distributions. We therefore conclude that the composite analysis is correct for isotropic cases.

6.3.3 Nodal point averaging

Each element calculates a stress value for every node that it contains. The different elements calculates different stress values for the same node. For linear elements such as T3 and Q4 elements, the stress is not continuous between elements. Therefore, in order to obtain the most accurate results possible, we make use of nodal point averaging. The stress value for any arbitrary node, is given as the average of all nodal stress values computed by every adjacent element. This technique makes the stress field derived from linear elements look much smoother than the field that is directly produced by the element. The nodal point averaging can also be modified by assigning different weights for different elements, however, in our implementation we have simply assigned the same weight to every element. For the cases of uniform material properties over the cross-section, this should work well. To demonstrate the effect of nodal point averaging, we make use of **MeshSTL** (described in Section 4.1) to output results in a file format similar to *.stl* (see Appendix C.5). This output file has no averaging of nodal stress values. The differences between nodal averaging and no nodal averaging for a mesh consisting of T3 elements are illustrated in Fig. 6.6.

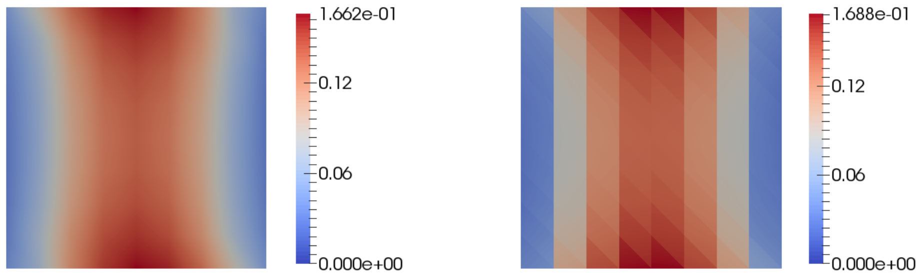


Figure 6.6: Comparing τ_{xz} for $F_{s,x}$ applied in the horizontal direction, with nodal stress averaging to the left and not (*.stl* file format) to the right. T3XLarge (128 elements) is used

6.3.4 Constant and linear strain elements

For elements with linear shape functions defining the displacement field, we expect a constant strain field (and stress field as we only consider elastic deformation). Elements containing quadratic shape functions on the other hand, should produce a linear strain field. The Q4 and Q9 elements are special cases as they have bilinear and biquadratic shape functions which should produce non-constant and bilinear strain fields respectively. When the results are outputted with nodal point averaging techniques it is hard to tell if the elements have a constant or linear variation in their stresses. This is because ParaView produces continuous fields between the nodal values. If the output format is changed to the aforementioned *.stl* file format, however, the behaviour of the elements become more apparent. Figs. 6.7 and 6.8 compares *.stl* stress values from meshes consisting of the four different element types. The shear analysis is used, as it produces a quadratic stress field over the cross-section. We have deliberately chosen coarse meshes in order to observe the differences between the linear, quadratic, bilinear, and biquadratic elements. Considering the coarseness of the mesh, the numerical values should not be given too much consideration.

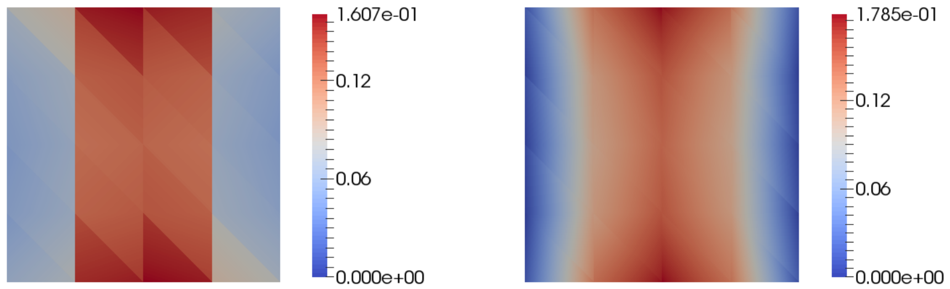


Figure 6.7: Comparing τ_{xz} for $F_{s,x}$ applied in the horizontal direction, with T3 elements (T3Large) to the left and T6 elements (T6Large) to the right. 32 elements were used for the analysis

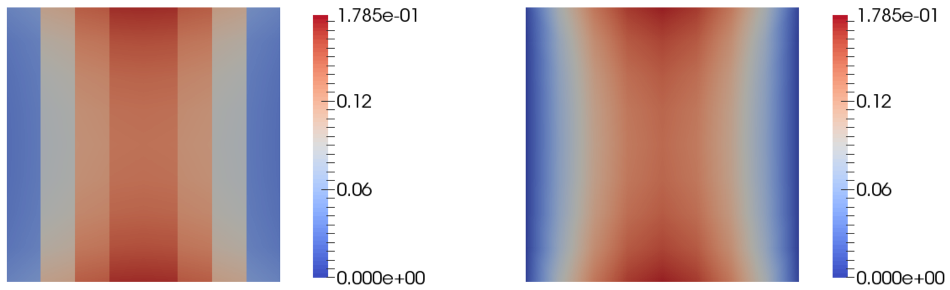


Figure 6.8: Comparing τ_{xz} for $F_{s,x}$ applied in the horizontal direction, with Q4 elements (Q4Large) to the left and Q9 elements (Q9Large) to the right. 64 elements were used for the analysis

Fig. 6.7 shows that the stress field obtained from T3 elements is constant, while the T6 elements produce a non-constant stress field. Fig. 6.8 illustrates a non-constant stress field for both Q4 and Q9 elements. The stress field of the Q4 element however, only varies linearly with y (vertical direction) as described in Section 2.5.2. We observe that T6 and Q9 elements produce significantly more accurate stress distributions, which ultimately means that they will have a faster rate of convergence compared to the T3 and Q4 elements. This will be investigated further in Section 6.3.6. When using nodal point averaging, the stress fields for the four elements will appear the same in ParaView, however the numerical values will be different.

We will now compare the numerical values obtained from meshes with different element types. In FEA the number of nodes generally dominates the workload for very large meshes (see Section 6.5). Therefore we have matched the meshes on the number of nodes. Again we use the massive rectangular mesh with dimensions $200 \times 100\ mm^2$. The torsion constant and the shear deformation factors are compared, as they serve as an indication on the correctness of both the torsion and shear analysis. We used a coarse mesh with 153 nodes, as well as a refined mesh with 2145 nodes. It should be noted that CrossX uses T6 elements, and for the analysis we assume the CrossX results for the fine mesh to be correct. The results are summarized in Tables 6.4 and 6.5.

Table 6.4: Cross-sectional parameters for massive rectangle of $200 \times 100\ mm^2$, using triangular elements. Number of nodes is indicated in parenthesis

Rectangle	**Coarse (153)**			**Refined (2145)**		
Parameter	**T3**	**T6**	**CrossX**	**T3**	**T6**	**CrossX**
$\boldsymbol{I}_t \cdot 10^{-6}\ [mm^4]$	46.4787	45.7928	45.793	45.7841	45.7367	45.737
$\boldsymbol{\kappa}_x$	1.17352	1.17687	1.17629	1.17671	1.17692	1.17692
$\boldsymbol{\kappa}_y$	1.16339	1.17621	1.17688	1.17606	1.17692	1.17692

Table 6.5: Cross-sectional parameters for massive rectangle of $200 \times 100\ mm^2$, using quadrilateral elements. Number of nodes is indicated in parenthesis

Rectangle	**Coarse (153)**			**Refined (2145)**		
Parameter	**Q4**	**Q9**	**CrossX**	**Q4**	**Q9**	**CrossX**
$\boldsymbol{I}_t \cdot 10^{-6}\ [mm^4]$	46.0079	45.7536	45.793	45.7535	45.73764	45.737
$\boldsymbol{\kappa}_x$	1.17348	1.17688	1.17629	1.17671	1.17692	1.17692
$\boldsymbol{\kappa}_y$	1.16327	1.17623	1.17688	1.17606	1.17692	1.17692

First we compare the T6 and Q9 (quadratic) elements to the T3 and Q4 elements (linear). We observe that the quadratic elements significantly outperforms the linear elements in terms of accuracy. This difference is even more apparent for the coarse mesh. The results obtained from the coarse mesh with quadratic elements are in fact as accurate as the results from the fine meshes with linear elements. This is an indication that for coarse meshes the quadratic elements are preferred.

Next we compare the accuracy of the triangular elements to the accuracy of quadrilateral elements. The Q4 elements produce a non-constant stress field, and are expected to outperform T3 elements. The torsion constant derived from the coarse T3 mesh is off by 1.62% compared to the CrossX results from the fine mesh (assumed to be accurate). This result is a lot stiffer than for the coarse Q4 mesh, which is only off by 0.59 %. For the shear factors the differences between T3 elements and Q4 elements are insignificant. When comparing T6 and Q9 elements from the coarse mesh to the accurate CrossX values, we can observe a similar trend. The torsional constant of the Q9 element is only off by 0.04 %, while the T6 elements are off by 0.12 %. Again the difference for the shear factor is insignificant. From this we conclude that the quadrilateral elements are preferred for simple cross-sections where the geometry allows for undistorted elements. For complex meshes however, the triangular elements will create a better approximation of the cross-section geometry.

For further validation of our results we seek an exact solution for the torsion constant. Eq. (6.1) [15] is known to calculate the torsional constant for rectangular cross-sections with a very small margin of error. For the rectangular cross-section we compute: $I_t = 45.776\ mm^4$, which is identical up to the three first digits of CrossX from the fine mesh. This indicates that CrossX (and by extension CCSC) is producing correct results.

$$I_t = ab^3\left(\frac{16}{3} - 3.36\left(\frac{b}{a} - \frac{b^4}{12 \cdot a^4}\right)\right) \tag{6.1}$$

6.3.5 Combining element types

Often times it can be beneficial to combine different element types in a mesh. As shown previously, quadrilateral elements often perform well for simple geometries, but they are not very good at physically approximating complex geometries. Combining element types can sometimes be the solution. An example of this is shown in Fig. 6.9, where six-node triangular elements are used to approximate the half circle, while nine-node quadrilateral elements are used to approximate the square. For CCSC we have implemented compatibility for multiple element types in the same mesh. In order to test the results when combining elements we compared the mesh seen in Fig. 6.9, with a mesh consisting only of six-node triangular elements. Results of the analyses performed on the two meshes is shown in Fig. 6.10. We observe that the distribution is the same for both cases, and the maximum effective stress of the combined mesh is only off by 0.03% compared to the T6 mesh.

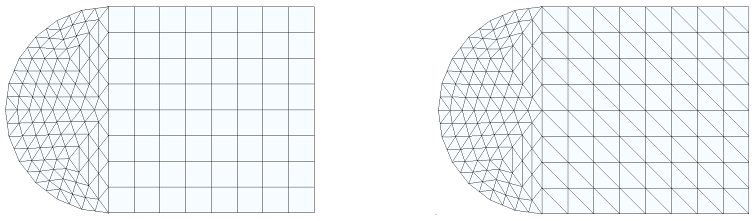


Figure 6.9: A mixed mesh on the left and a regular T6 element mesh on the right

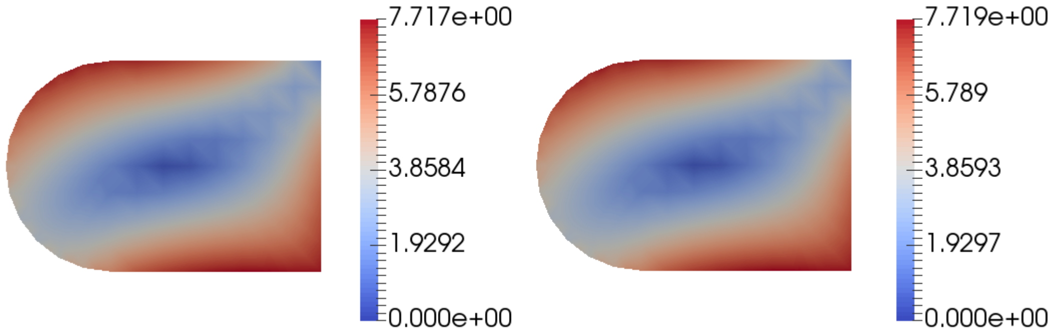


Figure 6.10: Effective stress. Mixed mesh on the left, T6 mesh on the right

6.3.6 Mesh convergence

We have mentioned the importance of convergence numerous times. To test for convergence we used a square cross-section ($100\times100mm^2$) meshed with T3, T6, Q4, and Q9 elements. Several different refinement levels, ranging from minimal working examples to very fine meshes were used. For this analysis we chose to compare meshes with the same number of elements. This will provide an indication of the accuracy achieved with roughly the same computational effort (see Section 6.5). We also compared the convergence of CCSC with that of CrossX. All the tests compare the maximum effective von Mises stress in the cross-section, as this value is based on both torsion and shear analysis. Fig. 6.11 shows how the effective stress converges to a solution as the mesh is refined. We observe that both CCSC and CrossX converge towards 12.10 MPa when using T6 elements. They produce almost identical results, even for coarse meshes. The Q9 element also converges to the same value, although a lot faster. Consequently, the Q9 element is by far the most accurate element type for coarse meshes.

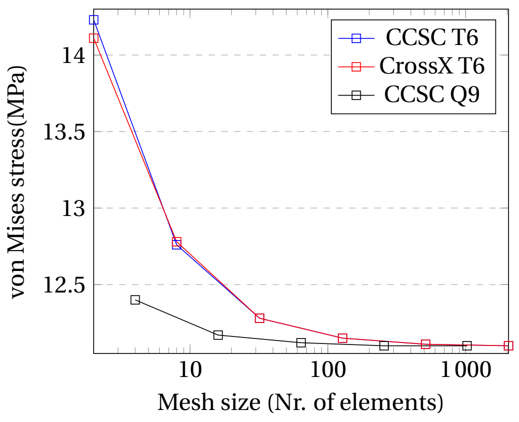


Figure 6.11: Convergence of stress values for CCSC and CrossX when refining mesh using quadratic elements

Fig. 6.12 depicts the same analysis, but now comparing meshes containing triangular elements (on the left), and meshes containing quadrilateral elements (on the right). As expected we observe the T6 mesh to converge much faster to the correct solution compared to the T3 mesh. The quadratic element is again to be preferred over the linear counterpart for coarse meshes. The same tendencies can be seen when comparing stress values for Q4 and Q9 elements. The improved accuracy and rate of convergence for Q9 compared to Q4 elements is even more pronounced than what we saw in the comparison of triangular elements.

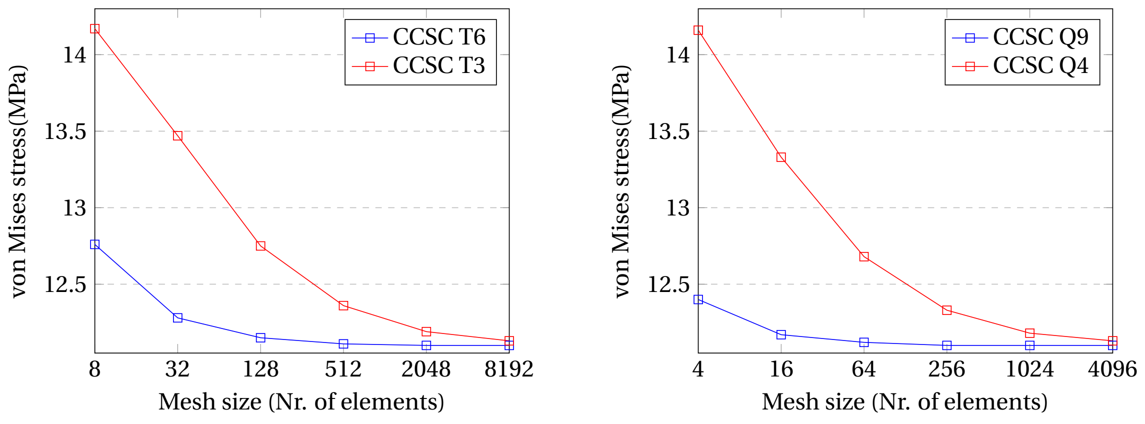


Figure 6.12: Convergence of stress values for CCSC and CrossX when refining mesh. Triangular elements on the left, quadrilateral elements on the right

6.4 Validation of thin-wall results

The implementation of thin-walled cross-section analysis needs further validation than simply considering the parameters from the two beam profiles previously discussed in Section 6.2. For this purpose we designed a simplified thin-wall airfoil cross-section.

6.4.1 Airfoil analysis

The airfoil is illustrated in Fig. 6.13. Aluminum alloys are frequently used in aircraft construction, and alloy 2024-T3 was therefore our choice of material. Its material properties are described in Table F.1 found in the Appendix. We performed the same analyses in both CCSC and CrossX, using the standard set of forces as defined in Section 6.1. CrossX calculates the angle of principal axis to be 89.63°, while CCSC calculates it to be −0.37° (this is discussed in Section 4.1). In order to compare the results more easily, we forced CCSC to rotate 89.63°. Table 6.6 presents a comparison of cross-sectional parameters, as calculated by the two programs.

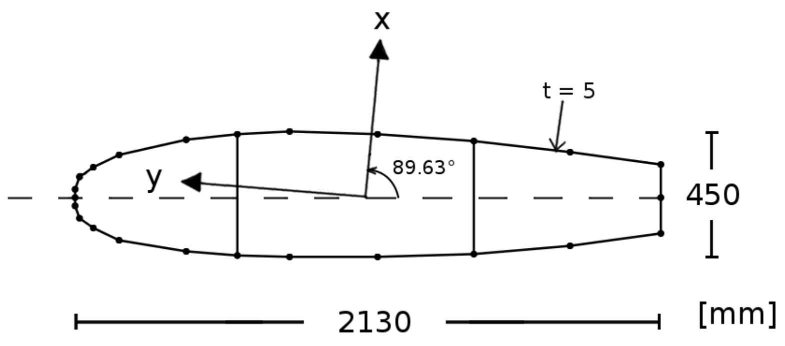


Figure 6.13: Thin-walled cross-section of an airfoil

Table 6.6: Cross-sectional parameters for airfoil, comparing CCSC to CrossX

Airfoil	**Thin-wall analysis**	
Parameters	**CCSC**	**CrossX**
$\boldsymbol{A} \cdot 10^{-3}\quad [mm^2]$	27.7413	27.741
$\boldsymbol{I_x} \cdot 10^{-8}\quad [mm^4]$	112.364	112.36
$\boldsymbol{I_y} \cdot 10^{-8}\quad [mm^4]$	8.74393	8.7444
$\boldsymbol{I_t} \cdot 10^{-8}\quad [mm^4]$	28.5638	28.564
$\boldsymbol{I_\omega} \cdot 10^{-14}\quad [mm^4]$	1.23862	1.2386
$\boldsymbol{x_s}\quad [mm]$	5.77183	5.8
$\boldsymbol{y_s}\quad [mm]$	51.6396	51.6
$\boldsymbol{\kappa_x}$	3.50461	3.54154
$\boldsymbol{\kappa_y}$	1.50189	1.50189

Every parameter listed in the previous table (except for one) is calculated numerically identical up to at least four digits (excluding the shear centers which only has one decimal point for CrossX). The only exception is κ_x, with a difference of about 1.04%. We have analyzed numerous thin-walled cross-sections (listed in Table D.4 in the Appendix), and none of them has had a discrepancy close to this.

We also compared the stress distributions obtained from CCSC and CrossX. In Fig. 6.14 it is easily observed that the shape of the ω - diagram is matching, and that the maximum values are virtually identical. The minimum value is, however, harder to determine from the figure. This is because the result bar used in CrossX is scaled so that it goes from $\pm$ the maximum *absolute* value, instead of ranging from the maximum to the minimum value. For this reason we put together Table 6.7 in order to better illustrate the maximum and minimum values of the analyses. From this table we can see that the minimum values are in fact equal for this diagram.

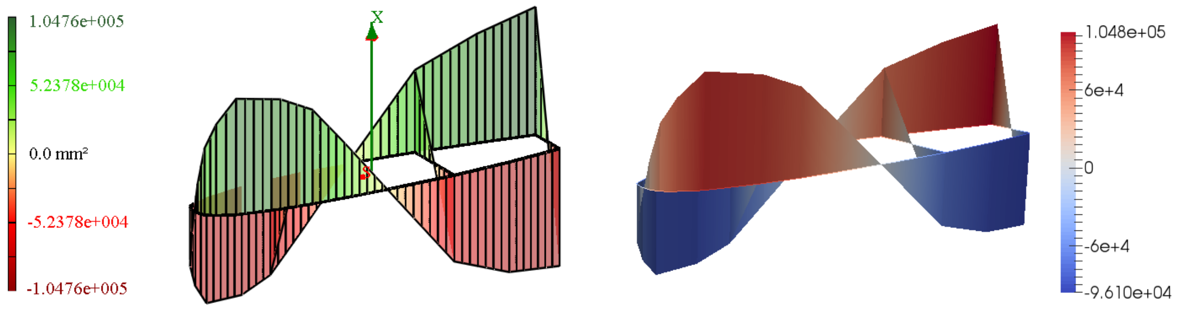


Figure 6.14: ω - diagram for airfoil in CrossX (to the left) and CCSC (to the right)

We compared the stress distributions for all the individual load cases, and all of them matched both in terms of shape and distribution. When analyzing the shear stresses obtained from CCSC it is important to keep the direction of each element in mind. The element direction can be viewed in ParaView by using the python script *DisplayArrows*, found under the *Scripts* folder (described further in Appendix D.1). If the shear stress is positive along an element, that indicates that the shear flow has the same direction as the element. Negative shear stress values indicate that the shear flow moves in the opposite direction of the element. In CrossX the direction of the shear flows are directly indicated by arrows.

As a consequence the shear stress distributions obtained from CCSC might have some elements where the numerical values have the opposite sign of CrossX. If the element direction of CCSC is taken into consideration however, the results are in fact the same (even though they appear to be opposite), as we are only looking for the direction and size of the flow. This can easily be observed in Figs. B.4 and B.5 in the Appendix. This phenomenon only occurs when an element is part of two cells, and is set to match a counter-clockwise direction for *one* of them (see Section 3.5). It should also be noted that this is only related to the shear flow, and therefore only happens for shear stress distributions.

The axial and shear stress resultants are compared in Figs. 6.15 and 6.16 respectively and they are found to be identical. Again we can observe that the numerical values in the webs have opposite values for CCSC and CrossX, and again the flow is found to be the same. Because of the difference in numerical values, the maximum and minimum τ values in Table 6.7 might differ from each other.

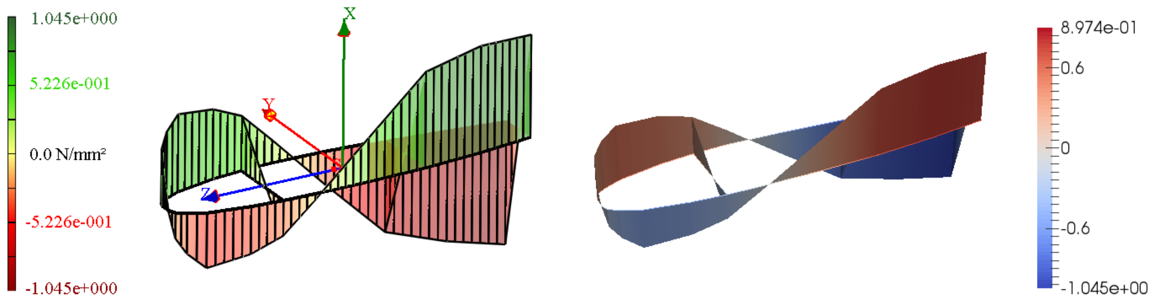


Figure 6.15: Resulting axial stress (σ_z) distribution for airfoil

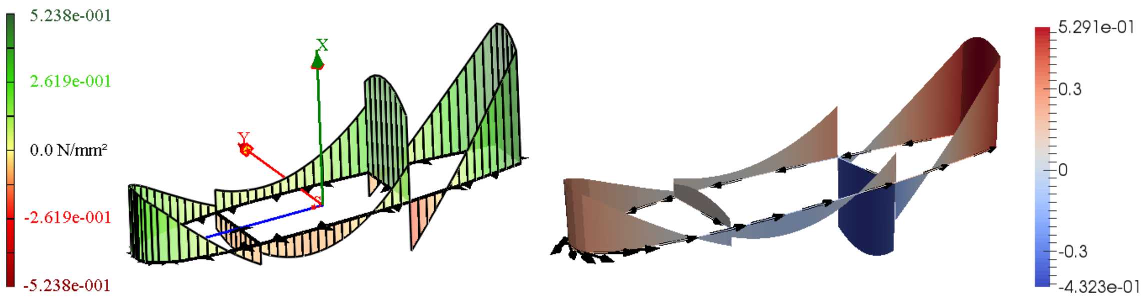


Figure 6.16: Resulting shear stress (τ) distribution for airfoil

Table 6.7: Comparing extreme values for ω - diagram and resulting axial and shear stress distributions. For shear stress we expect the extreme values to differ significantly occasionally, due to differences in element directions

Airfoil	**Max**		**Min**	
Parameters	**CrossX**	**CCSC**	**CrossX**	**CCSC**
$\boldsymbol{\omega} \cdot 10^4 \quad [mm^2]$	10.48	10.476	-9.610	-9.6102
$\boldsymbol{\sigma_z}$ (resulting) $\quad [N/mm^2]$	0.8974	0.8974	-1.045	-1.045
$\boldsymbol{\tau}$ (resulting) $\cdot 10^{-1} \quad [N/mm^2]$	5.291	5.238	-4.323	-1.991

6.4.2 In-depth analysis of ω - diagram

During our extensive validation of thin-walled analysis, we have discovered that for some cross-sections the computed ω - diagram differs from what CrossX suggests. An example of such a cross-section is a simple L-beam ($100 \times 100mm^2$). CCSC computes the entire diagram to be equal to zero, while CrossX computes significant extreme values of $\pm 0.0625mm^2$. Table 6.8 demonstrates how all parameters are computed virtually identical except for the warping stiffness I_ω. We also observe that CrossX calculates slightly different values for the two shear deformation factors.

Table 6.8: Cross-sectional parameter for L-beam ($100 \times 100 mm^2$ and thickness equal to 1). Angle of principal axis is −45° for CCSC and 45° for CrossX

L-beam	**Program**	
Parameter	**CCSC**	**CrossX**
$\boldsymbol{I}_t \quad [mm^4]$	66.6667	66.6667
$\boldsymbol{I}_\omega \quad [mm^6]$	0	1041.6
$\boldsymbol{x}_s \quad [mm]$	0	0
$\boldsymbol{y}_s \quad [mm]$	-35.3553	-35.4
$\boldsymbol{\kappa}_x$	1.2	1.19976
$\boldsymbol{\kappa}_y$	1.2	1.19994

The consequences of the ω - values may seem insignificant, but in practice they are not. Using the set of default forces (defined in Section 6.1) for the cross-section, the stresses originating from bimoment and warping torsion actually dominates the axial stress field. Certainly a correct ω - diagram is of importance. Table 6.9 presents the maximum axial and shear stresses for the L-beam.

Table 6.9: Maximum axial and shear stresses in L-beam subjected to different loading cases

Axial stress $[N/mm^2]$	**CCSC**	**CrossX**
$\boldsymbol{\sigma}_z$ due to **N**	5	5
$\boldsymbol{\sigma}_z$ due to $\boldsymbol{M}_x$	424.3	212.1
$\boldsymbol{\sigma}_z$ due to $\boldsymbol{M}_y$	212.1	424.2
$\boldsymbol{\sigma}_z$ due to $\boldsymbol{B}$	0	6000

Shear stress $[N/mm^2]$	**CCSC**	**CrossX**
$\boldsymbol{\tau}$ due to $\boldsymbol{T}_{stv}$	15000	15000
$\boldsymbol{\tau}$ due to $\boldsymbol{V}_x$	10.61	10.47
$\boldsymbol{\tau}$ due to $\boldsymbol{V}_y$	10.61	10.61
$\boldsymbol{\tau}$ due to $\boldsymbol{T}_{wrp}$	0	300

ω - values are calculated according to Eq. (3.15). For most cross-sections the extreme ω - values are significantly large. A selection of such cross-sections can be seen in Table 6.10 along with their ω_{max} value.

Table 6.10: Table showing ω_{max} value from CCSC for a selection of thin-walled cross-sections

Mesh	IPE300	RHS	airfoil
$\boldsymbol{\omega}_{max}\ [mm^2]$	1085.0	387.0	10480

For a few cases CrossX computes significantly lower ω_{max} values (approximately by an order of 10^5), with the L-beam being one of them. It is only in these cases where CCSC computes zero ω over its entirety. When CrossX outputs the shear center, it is rounded to one significant decimal point. We believe that this might be the problem, however we have no way of knowing if these parameters are rounded of during the computations of CrossX, or if it is just rounded of for the output. In any case we believe the output of CCSC to be correct, as simple hand calculations support our theory.

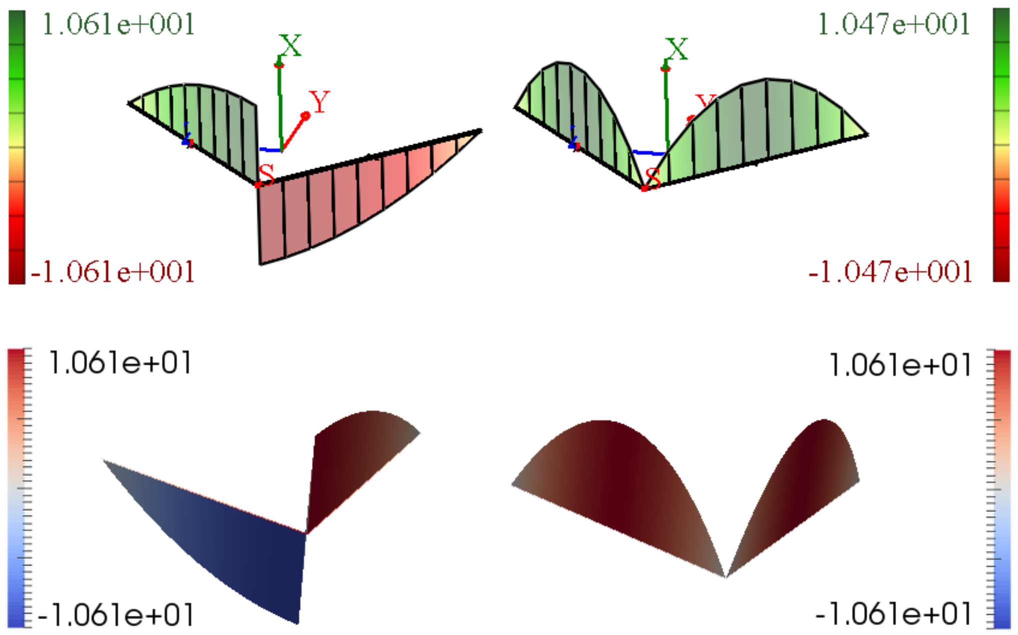


Figure 6.17: τ due to V_x and V_y for the simple L-beam. CrossX on top, and CCSC on the bottom

Fig. 6.17 shows a similar case. In this symmetric case we would expect that the same extreme values between the two cases, however CrossX shows a slight difference between the two. This in turn causes the slight difference in shear deformation factors seen in Table 6.8. This further confirms our theory of some numerical inaccuracy in CrossX.

6.5 Running time of CCSC

To compare the running time of the massive implementation of CCSC, we used the same meshes as in the convergence test in Section 6.3.6. They are matched on the number of elements, as this will dominate the running time of CCSC for most use cases. For very large meshes the number of nodes is the dominating factor, as the Cholesky decomposition algorithm has a complexity of $O(n^3)$. We observe from Figs. 6.18 that analysis of meshes containing T6 and Q9 elements is somewhat slower than T3 and Q4 for the same number of elements. This is to be expected as the computations done for the quadratic elements are more complex, also these elements have more nodes leading to a larger system of equations. For smaller meshes the differences in running time is negligible.

As the number of elements in the mesh increases the differences become more pronounced. We mentioned in Section 6.3.6 that the quadratic elements are more accurate for coarse meshes, when compared to linear elements. The difference in accuracy decreases as the number of elements increases. If the mesh is large enough to give the same accuracy between linear elements and quadratic elements, the linear elements are preferred, as they require less computational effort. All tests were done on a laptop with an Intel Core i7-4700MQ CPU with 2.40 GHz clock speed, and are averaged from five separate runs. The results can be seen in Fig. 6.18.

Further analysis has shown that nearly half of the running time is spent on writing results to file. We have not made made any attempts to improve this, and we leave it to new developers to investigate improved file writing options.

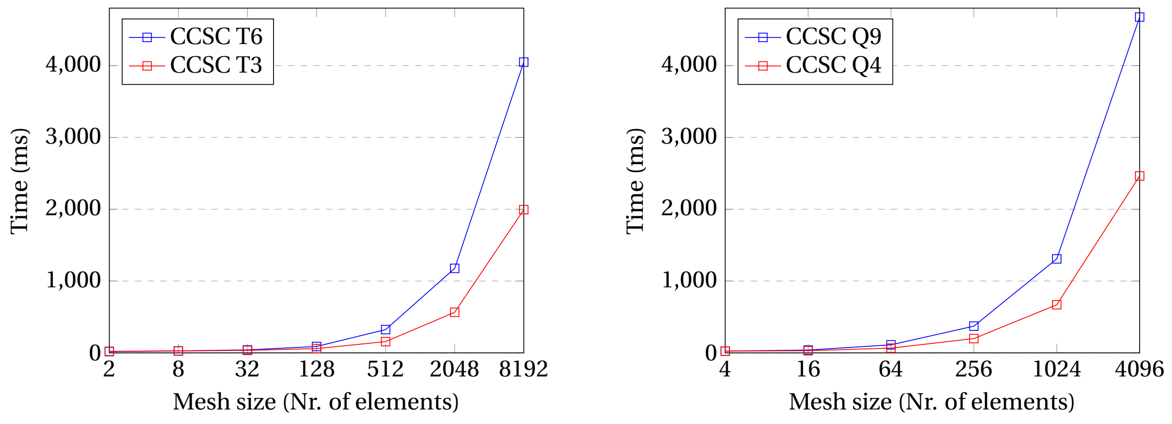


Figure 6.18: Running time of CCSC. Triangular elements on the left, quadrilateral elements on the right

The thin-walled models are very simplistic, and as a consequence the running times of the analysis is significantly faster than what is seen for massive cross-sections. For all practical purposes the time spent on analyses is a non-issue for the user. For the airfoil cross-section analyzed in Section 6.4.1, we measured the overall running time to be 1288 ms. However only 152 ms was spent on the analysis, 88% of the time was spent writing the results to file.

Chapter 7

Summary and Recommendations for Further Work

7.1 Summary and conclusions

For our master thesis we were tasked with further developing an analysis software for calculation of parameters and stress distributions for complex beam cross-sections. Our work consisted of implementing support for thin-walled cross-sections, adding four- and nine-node quadrilateral elements, starting the development of composite materials, as well as evaluating and restructuring the existing code. Additionally, a lot of effort was put into correcting both the torsion and shear analyses and implementing computations of additional parameters such as the torsion constant, shear center, and shear deformation factors.

Through extensive testing we have demonstrated that our implementation of massive analysis computes sufficiently accurate cross-sectional parameters and stress distributions for well-known cross-sections. We have shown that the implementation of quadrilateral elements is working, and that it produces more accurate results for simple cross-sections when compared to triangular elements. The thin-walled implementation has also been analyzed thoroughly for numerous test cases, and was shown to produce satisfying parameters and stress distributions for both simple and complex cross-sections.

We spent more time than anticipated on implementing the thin-walled analysis, and as a consequence we did not get as far with the composite functionality as we would have liked. We have, however, developed a unified analysis for torsion and shear, which is an important first step towards composite analysis. This unified analysis has been shown to provide identical results to the individual torsion and shear analyses for the case of isotropic materials. Throughout the development the software has been extensively restructured with composite functionality in mind. Along with a comprehensive documentation, this should provide a solid foundation for future implementation of composite materials.

7.2 Discussion

A simple square cross-section with numerous refinement levels was used to test the performance of the four element types available for massive analysis. Through testing we found that the quadratic elements outperformed the linear elements in terms of accuracy. This is expected as they have a linear strain field, which in turn gives a linear stress variation over the element. For very large meshes it might be advisable to use linear elements instead. This is because the difference in accuracy goes to zero as the mesh is refined. The difference in computational effort on the other hand, keeps increasing because the number of nodes in the mesh becomes the dominating factor. We have also demonstrated that the quadrilateral elements have a faster rate of convergence compared to triangular elements. However, they are not as good at approximating complex geometry without causing severe element deformations. Because of this, a good solution is sometimes to combine different element types.

One of the properties of FEM is that it produces a positive definite stiffness matrix. However, for some rare cases, CCSC computes it to be non-positive definite. We have not found the reason for this, but it seems to only happen when circular features are present in the cross-section. Whenever this happens the results should be considered with great caution, although from our experience it does not seem to affect the results too much. CCSC is highly dependent on quality meshes in order to provide accurate results, as is the case for most FEA programs. In order to help the mesh generation for more complex cross-sections the user should partition the cross-section into smaller sub-regions with regular shapes. When creating mesh files for CCSC it is important to note that the nodes are arranged in a counter-clockwise fashion for each element. Failing to do so may result in parameters having opposite signs of the correct solution. It is the users responsibility to be able to discern between good and bad meshes. In the same vein, the user is expected to critically analyze results and make sure they are not polluted by singularities.

The analysis of thin-walled cross-sections is solved for element shear flows as the unknowns. This differs from the massive analysis approach, where FEM is used to solve for nodal displacements as unknowns. The main advantage of our thin-wall approach is that we achieve linear stress variation with two-node elements. Because of this we did not need to implement the three-node line elements in order to compute accurate stress distributions. The most apparent downside to this approach is that there is currently no way to use line elements along side the triangular and quadrilateral elements for massive cross-section analysis.

During initial testing of the thin-walled analysis we discovered that our shear stress distributions due to axial stresses were inverted when compared to CrossX. To remedy this problem we ended up forcing the load vector to be negative in order to obtain results that make physical sense. Further validation revealed that CCSC and CrossX computes different ω - diagrams. In cases where CrossX produces small ω - values CCSC computes zero across the entire diagram. This difference was shown to be of great importance as the axial stresses calculated from warping proved to be the dominating factor. We believe that this has its roots in some kind of small numerical round-off error in CrossX. Hand calculations have further supported this theory.

It should be noted that we have not performed testing of cross-sections consisting of elements with different material properties.

7.3 Recommendations for further work

Being such a big and comprehensive task, we had to prioritize our time to what we felt to be most important for CCSC. Consequently there are still several aspects of CCSC that could be interesting for future projects or master theses.

7.3.1 Enhancing/adding new functionality

Using line elements for massive analysis - An interesting task would be to implement support for massive analysis using line elements in combination with regular elements. Line elements obtain stress variation in only one direction, which could potentially reduce the computational effort significantly. The massive analysis solves for displacements as unknowns, and consequently line elements consisting of three nodes are needed in order to obtain linear stress variation.

Support for composite materials - In today's industry composite materials have become very popular because of their high strength and light weight when compared to traditional materials. Because of this, an implementation of composite analysis would be a huge addition to this software. Through our proposed method of unifying torsion and shear analysis, we have made a solid foundation for future development of composite functionality.

Individual assignment of material properties - An interesting task for future development of CCSC, would be to find an efficient way to individually assign material properties to groupings of elements, ideally directly through the user interface. This would enhance the user-friendliness of the application, and is vital if composite functionality is added.

Supporting additional mesh formats - Adding support for additional mesh formats is something that could be done to increase the versatility of the application. This would be easy to do, either by creating an abstract MSHParser class, or having a class that translates other mesh formats to the *.msh* file format.

Shear center from shear analysis - Currently CCSC only computes the shear center based on torsional analysis. Finding the shear center from shear analysis will give slightly different results (unless Poisson's ratio is zero), and could therefore provide some additional analytical value.

7.3.2 Increase modifiability

Creating abstractions of classes - There are some parts of the code that could benefit from restructuring classes into a hierarchy of abstract classes and sub-classes. Perhaps the most obvious cases would be the parser and result writer classes. The massive and thin-walled implementations currently have separate parsers and writers which operate in pretty much the same manner. A lot of code could be reused by having a general abstract class and leaving specific implementations to the sub-classes as needed. Another place where this could be done is with the Mesh and MeshSTL class.

Renaming axes - The common naming convention in most literature and standards is to use (y,z) as the coordinate system for the cross-sectional plane. When we started working on CCSC, (x,y) was used instead. As this is of no real importance to the results we have not prioritized changing this. It would, however, be advantageous to adopt the naming convention, as this would make implementation of new theory less confusing for the programmer.

7.3.3 Optimization

Writing results to file - For thin-walled analysis it is not uncommon that 80-90% of the running time is spent writing results to file. A solution to this could be to find a better way to write the result files in batches, but we leave the implementation to future developers. This would also benefit the running time of massive analysis significantly.

Reduced integration - As previously mentioned in Section 2.6.3, there is currently no support for reduced integration in CCSC. By trading off some accuracy, reduced integration could significantly lower the computational effort needed for cross-section analysis, especially for larger problems. For some cases reduced integration is even recommended, as the loss of accuracy is counteracted by a closer approximation to the real world.

Considering symmetry - The workload of the program could be significantly lowered by taking symmetry into consideration. By applying different boundary conditions for the different symmetry cases, the number of nodes and elements to consider would be significantly reduced, which again would lower the computational effort needed. Although the global stiffness matrix needs to be recomputed for every load case, applying symmetry could potentially save a lot of time for large meshes.

Use of smart pointers - In its current state CCSC maintains vectors with pointers. Because C++ lacks garbage collection, having a lot of unhandled pointers could cause substantial memory leaks in the program. This is especially true when analyzing well refined meshes, or if the program is continuously running and not just executing the calculations once (as is the case in its current state).

A better approach to avoiding these memory leaks would be to let the objects handle their own lifespan. There are several ways to do this, the easiest would probably be to make use of smart pointers[21].

Eigen - Although the running times of CCSC are, for the most part, more than adequate, there is still room to make optimizations in the code. According to Eigen's documentation, all small matrices (a matrix with less than roughly 32 values) should have fixed sizes wherever possible[22]. This will improve the performance of matrix operations greatly, as Eigen no longer has to worry about dynamic memory allocation and loop unrolling.

Bibliography

[1] Kristian Strømstad. *Applikasjon for beregning av tverrsnittsparametre for komplekse bjelketverrsnitt*. Norwegian University of Science and Technology, 2014. Master's thesis.

[2] Kolbein Bell. *An Engineering Approach to Finite Element Analysis of Linear Structural Mechanics Problems*. Akademika publishing, 2013.

[3] Kolbein Bell, Oddmund V. Bleie, and Vollebæk. *CrossX documentation*, 2000.

[4] Gaël Guennebaud, Benoît Jacob, et al. Eigen version 3. `http://eigen.tuxfamily.org`, 2010.

[5] GitHub Inc. GitHub - Where software is built. `https://github.com/`, 2008.

[6] Christophe Geuzaine and Jean-François Remacle. Gmsh. `http://geuz.org/gmsh/`, 1997.

[7] Kitware. ParaView - Parallel Visualization Application. `http://www.paraview.org/`, 2015.

[8] Kitware. VTK - The Visualization Toolkit. `http://www.vtk.org/`, 1993.

[9] Dimitri van Heesch. Doxygen - Generate documentation from source code. `http://www.doxygen.org/`, 1997.

[10] The Welding Institute. What is reduced integration in the context of finite element analysis. `http://www.twi-global.com/technical-knowledge/faqs/structural-integrity-faqs/faq-what-is-reduced-integration-in-the-context-of-finite-element-analysis/`, 2016.

[11] S. Timoshenko and J. N. Goodier. *Theory of Elasticity.* McGraw-Hill Book Company, Inc., 1951.

[12] J. L. Meek. *Computer Methods in Structural Analysis of General Prismatic Beams.* E & FN SPON (Chapman & Hall), 1991.

[13] W. E. Mason and L.R. Herrmann. Elastic Shear Analysis of General Prismatic Beams. *Journal of the Eng. Mech. Div., ASCE*, pages 965–983, August 1968.

[14] Carlos A. Felippa. Introduction to Finite Element Methods, Chapter 28. Department of Aerospace Engineering Sciences and Center for Aerospace Structures, University of Colorado. `http://www.colorado.edu/engineering/CAS/courses.d/IFEM.d/IFEM.Ch28.d/IFEM.Ch28.pdf`, 2004.

[15] Warren C. Young and Richard G. Budynas. *Roark's Formulas for Stress and Strain.* The McGraw-Hill Companies, 7th edition, 2001. Chapter 10, Chapter 17.

[16] NAFEMS. International Association for the Engineering Modelling, Analysis and Simulation Community - The Importance of Mesh Convergence - 1. `https://www.nafems.org/join/resources/knowledgebase/001/`, 2016.

[17] Kjell H. Holthe. *FRAME - A computer program for linear static analysis of three dimensional frame structures including warping. Theoretical Manual SINTEF Report STF71 A 79005.* Trondheim, 1979.

[18] Christopher Gorse, David Johnston, and Martin Pritchard. Dictionary of Construction, Surveying and Civil Engineering. `http://www.oxfordreference.com/view/10.1093/acref/9780199534463.001.0001/acref-9780199534463-e-4530?rskey=YuX0bI&result=4526`, 2013.

[19] Axel Kemper. Finding all cycles in undirected graphs. `http://stackoverflow.com/questions/12367801/finding-all-cycles-in-undirected-graphs`, 2013.

[20] Per Kr. Larsen, Arild H. Clausen, and Arne Aalberg. *Stålkonstruksjoner - profiler og formler 3. utgave.* Tapir Akademiske Forlag, 2003.

[21] Microsoft MSDN. Smart pointers. `https://msdn.microsoft.com/en-us/library/hh279674.aspx`, 2015.

[22] Gaël Guennebaud, Benoît Jacob, et al. Eigen version 3 - fixed vs dynamic size. `http://eigen.tuxfamily.org/dox/group__TutorialMatrixClass.html#title`, 2010.

[23] ASM Aerospace Specification Metals Inc. ASM Material Data Sheet. `http://asm.matweb.com/search/SpecificMaterial.asp?bassnum=MA2024T3`, 2016.

Appendix A

Additional Theory

A.1 Bending stiffness

The bending stiffnesses are defined as

$$EI_x = \int_A E\, y^2\, dA \tag{A.1a}$$

$$EI_y = \int_A E\, x^2\, dA \tag{A.1b}$$

$$EI_{xy} = \int_A E\, xy\, dA \tag{A.1c}$$

The element has a different center of area than the cross-section, and the *Parallel axis theorem* adjust for this. The new bending stiffness is adjusted with the stiffness multiplied with the distance *d* between the axes.

$$EI_x^e = \int_{A_e} y^2 E_e\, dA_e + b^2 E_e A_e \tag{A.2a}$$

$$EI_y^e = \int_{A_e} x^2 E_e\, dA_e + a^2 E_e A_e \tag{A.2b}$$

$$EI^e_{xy} = \int_{A_e} xyE_e \, dA_e + abE_e A_e \tag{A.2c}$$

For the cross-sections the total bending stiffness is simply the sum of the elementary bending stiffness. With n elements this becomes

$$EI_x = \sum_{i=1}^{n} EI_{x_i} \tag{A.3a}$$

$$EI_y = \sum_{i=1}^{n} EI_{y_i} \tag{A.3b}$$

$$EI_{xy} = \sum_{i=1}^{n} EI_{x_i y_i} \tag{A.3c}$$

A.2 Second moment of area

The second moments of area with respect to x and y is given in Eq. (A.4), together with the product moment of area. They are defined as the bending stiffness divided by the Young's modulus of the base material (E_b).

$$I_x = \frac{EI_x}{E_b} \tag{A.4a}$$

$$I_y = \frac{EI_y}{E_b} \tag{A.4b}$$

$$I_{xy} = \frac{EI_{xy}}{E_b} \tag{A.4c}$$

The maximum second moment of area is given by

$$I_{max} = \frac{I_x + I_y}{2} + \sqrt{\left(\frac{I_x - I_y}{I_{xy}}\right)^2 + I_{xy}^2} \tag{A.5}$$

and the angle between the principal and global axis is given by

$$\beta = tan^{-1}\left(\frac{I_x - I_{max}}{I_{xy}}\right) \tag{A.6}$$

A.3 Shape functions

For all four element types implemented in our program it is a necessity to establish the shape functions. They are most easily established from the simple construction of *"0-lines"*. From this method any shape function can be expressed as

$$N_i = c_i L_1 L_2 \cdots L_n \tag{A.7}$$

$$L_j = 0 \quad \text{and} j = 1, 2, \cdots, n \tag{A.8}$$

L_j is then the equation of a straight line, that has a expression of a linear function dependent on the natural coordinates. c_i is a constant that assures unity at node i, i.e. $N_i = 1$.

The shape function N_i are then established by following three steps:

- choose the lowest number of the equations L_j needed to cover all nodes, except for node i, forcing the shape function to be zero at these nodes.
- coefficient c_i must be determined such that $N_i = 1$ at node i
- make sure that for every edge adjacent to node i, the number of nodes along the edge is of size one larger than the polynomial order of the shape function covering the edge. This assures continuity between two elements sharing this edge.

This method is used for establishing the shape functions for the four element types.

Appendix B

Additional Results

B.1 Massive results

B.1.1 Convergence

Table B.1: Convergence analysis for von Mises stress using triangular elements (results in MPa)

Number of elements	**2**	**8**	**32**	**128**	**512**	**2048**
Stress (CCSC T3)	14.16	14.17	13.47	12.75	12.36	12.19
Stress (CCSC T6)	14.23	12.76	12.28	12.15	12.11	12.10
Stress(CrossX T6)	14.11	12.78	12.28	12.15	12.11	12.10

Table B.2: Convergence analysis for von Mises stress using quadrilateral elements (results in MPa)

Number of elements	**4**	**16**	**64**	**256**	**1024**
Stress (CCSC Q4)	14.16	13.33	12.68	12.33	12.18
Stress (CCSC Q9)	12.40	12.17	12.12	12.10	12.10

B.2 Optimization

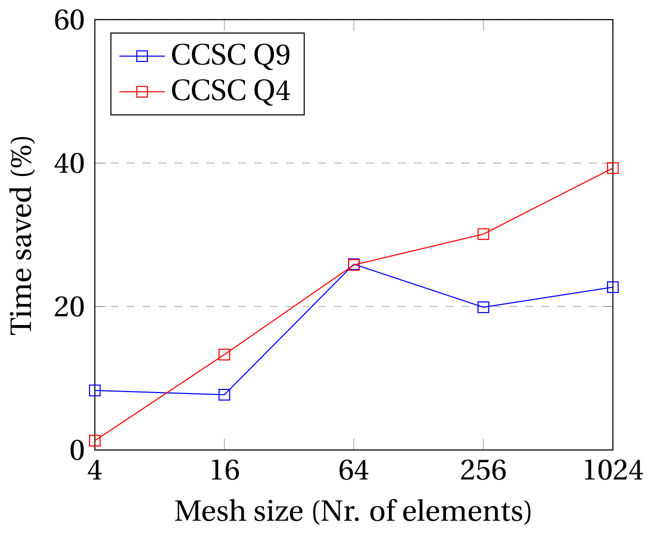


Figure B.1: Improvements from storing and reusing **N**, **B**, and **Jacobi** matrices

B.3 Thin-wall results

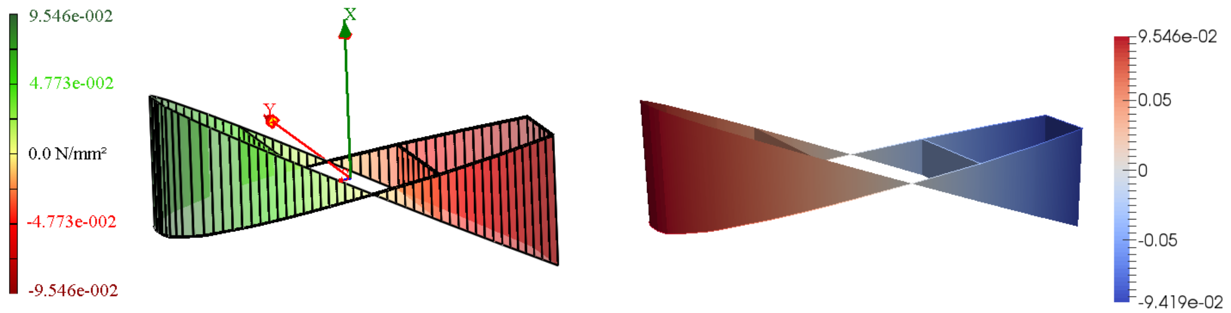


Figure B.2: Axial stress (σ_z) distribution due to shear load M_x

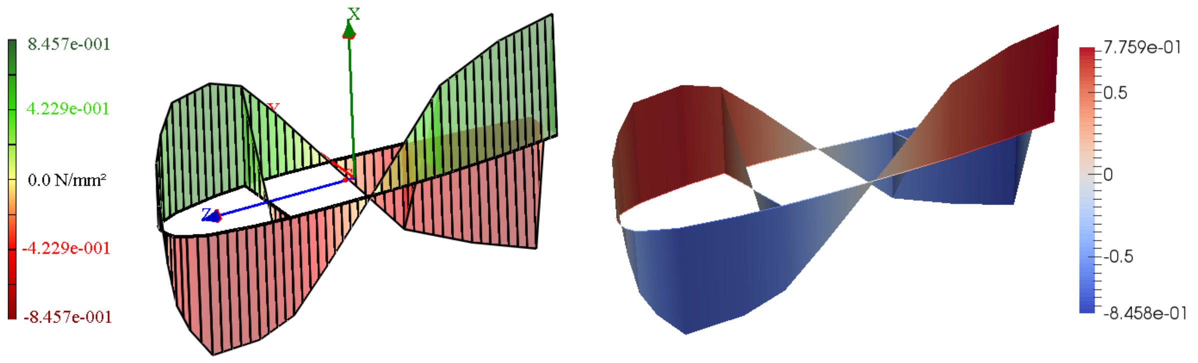


Figure B.3: Axial stress (σ_z) distribution due to bimoment B

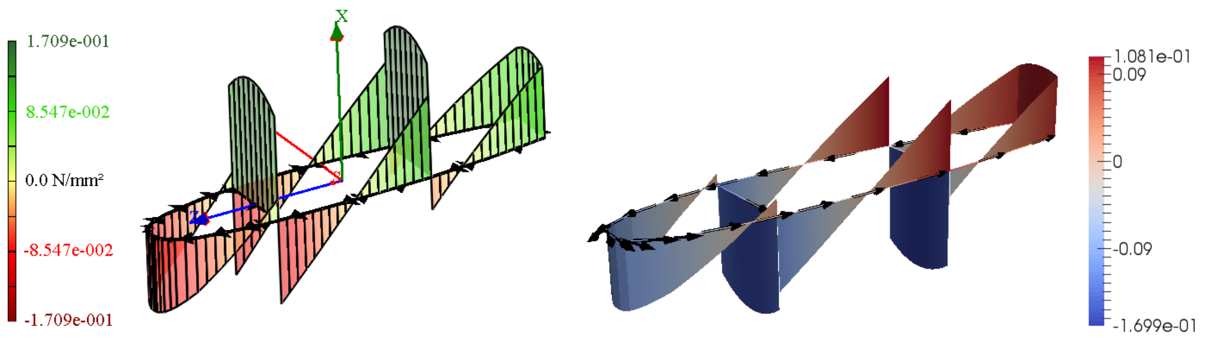


Figure B.4: Shear stress (τ) distribution due to shear load V_x

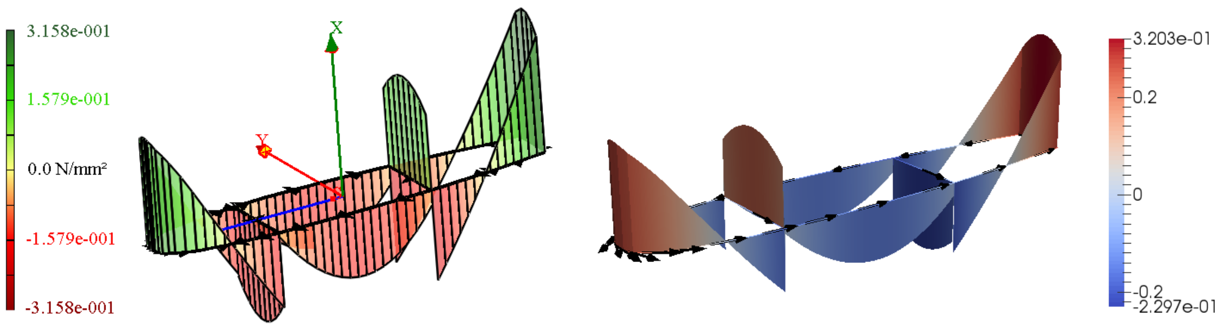


Figure B.5: Shear stress (τ) distribution due to T_{wrp}

Table B.3: Comparing extreme values outputted by CCSC and CrossX

Airfoil	**CCSC**		**CrossX**	
Parameters	**Min**	**Max**	**Min**	**Max**
$\boldsymbol{\sigma_z}$ due to shear load $M_x \cdot 10^{-2} [N/mm^2]$	9.546	-9.419	9.546	-9.419
$\boldsymbol{\sigma_z}$ due to bimoment B $\cdot 10^{-1} [N/mm^2]$	7.759	-8.458	7.759	-8.457
$\boldsymbol{\tau}$ due to shear load $V_x \cdot 10^{-1} [N/mm^2]$	1.081	-1.699	1.709	-1.121
$\boldsymbol{\tau}$ due to $T_{wrp} \cdot 10^{-1} [N/mm^2]$	3.200	-2.291	3.158	-2.319

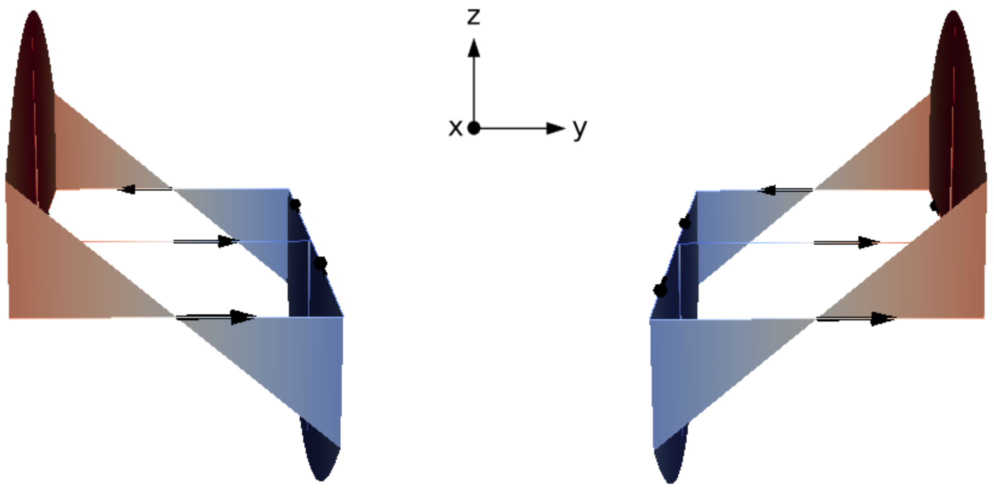
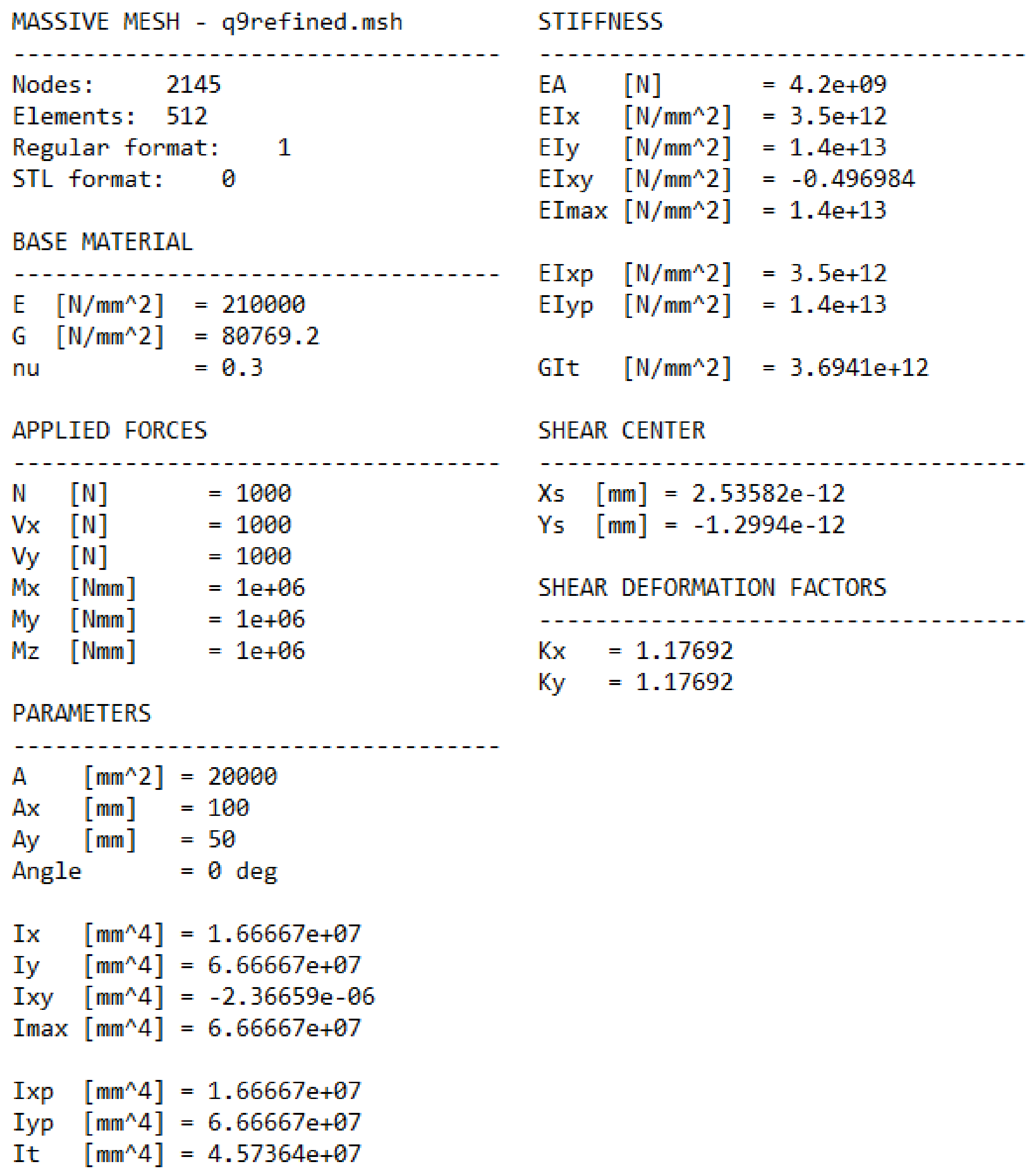


Figure B.6: Shear stress (τ) distribution due to V_x (into-plane). Arrow indicates positive direction for line segment. If negative numerical value, flow is in the opposite direction. With this in mind we see an example of incorrect shear flow to the left (shear flow meets at attacking point) and an example of correct shear flow to the right (distributes away from attacking point)

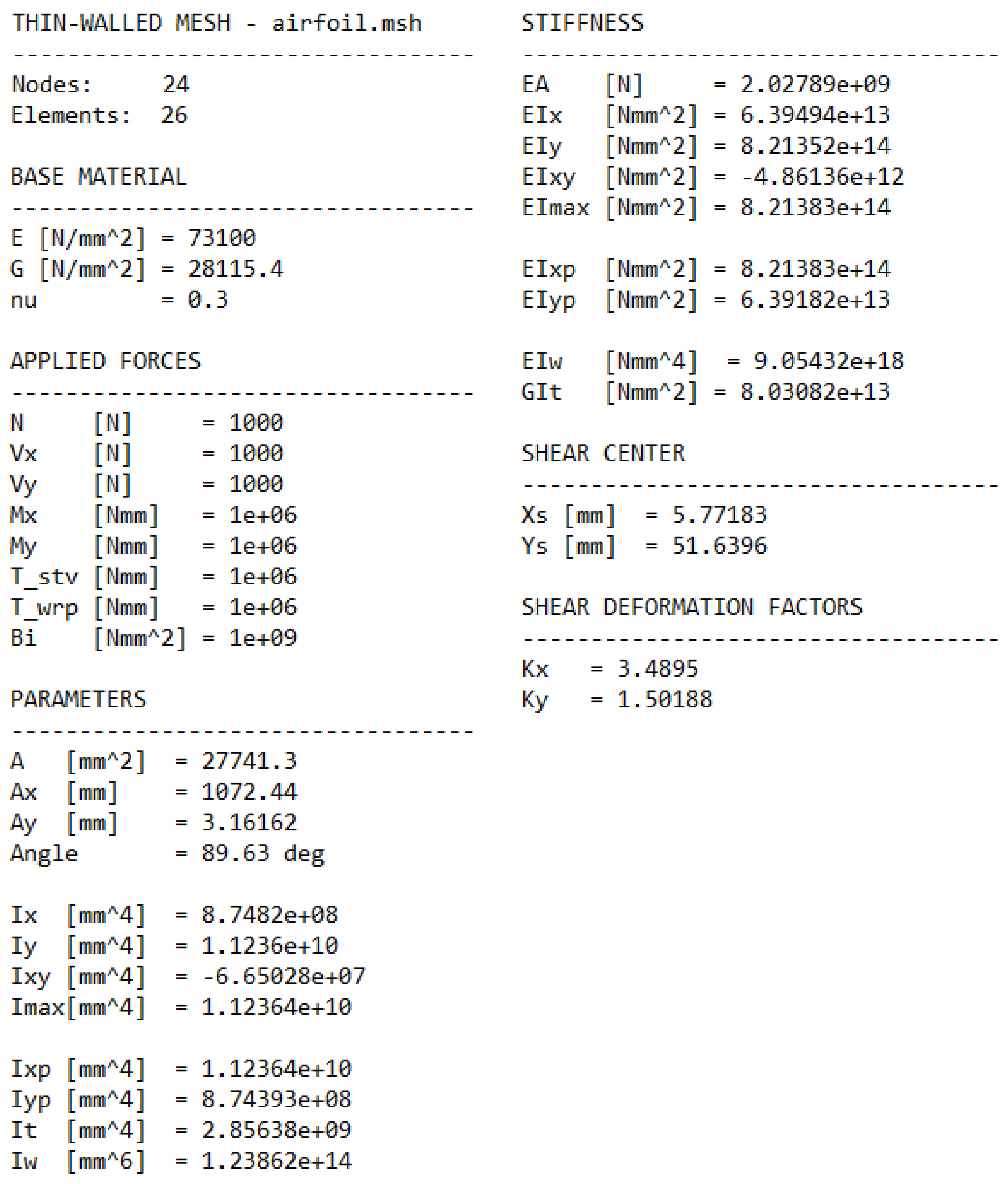
Figure B.7: Output file (*.txt*) for massive analysis of *Q9Refined.msh*. In addition to base material and applied forces, bending stiffnesses, torsion constant, shear center, and shear deformation factors are outputted



```
MASSIVE MESH - q9refined.msh
-----------------------------------
Nodes:     2145
Elements:  512
Regular format:    1
STL format:    0

BASE MATERIAL
-----------------------------------
E  [N/mm^2]  = 210000
G  [N/mm^2]  = 80769.2
nu           = 0.3

APPLIED FORCES
-----------------------------------
N   [N]       = 1000
Vx  [N]       = 1000
Vy  [N]       = 1000
Mx  [Nmm]     = 1e+06
My  [Nmm]     = 1e+06
Mz  [Nmm]     = 1e+06

PARAMETERS
-----------------------------------
A    [mm^2] = 20000
Ax   [mm]   = 100
Ay   [mm]   = 50
Angle       = 0 deg

Ix   [mm^4] = 1.66667e+07
Iy   [mm^4] = 6.66667e+07
Ixy  [mm^4] = -2.36659e-06
Imax [mm^4] = 6.66667e+07

Ixp  [mm^4] = 1.66667e+07
Iyp  [mm^4] = 6.66667e+07
It   [mm^4] = 4.57364e+07
```

```
STIFFNESS
-----------------------------------
EA    [N]        = 4.2e+09
EIx   [N/mm^2]   = 3.5e+12
EIy   [N/mm^2]   = 1.4e+13
EIxy  [N/mm^2]   = -0.496984
EImax [N/mm^2]   = 1.4e+13

EIxp  [N/mm^2]   = 3.5e+12
EIyp  [N/mm^2]   = 1.4e+13


GIt   [N/mm^2]   = 3.6941e+12

SHEAR CENTER
-----------------------------------
Xs   [mm] = 2.53582e-12
Ys   [mm] = -1.2994e-12

SHEAR DEFORMATION FACTORS
-----------------------------------
Kx    = 1.17692
Ky    = 1.17692
```

Figure B.8: Output file (*.txt*) for thin-wall analysis of *airfoil.msh*. In addition to base material and applied forves, bending stiffnesses, torsion and warping constant, shear center, and shear deformation factors are outputted



```
THIN-WALLED MESH - airfoil.msh
----------------------------------
Nodes:      24
Elements:   26

BASE MATERIAL
----------------------------------
E [N/mm^2] = 73100
G [N/mm^2] = 28115.4
nu         = 0.3

APPLIED FORCES
----------------------------------
N     [N]      = 1000
Vx    [N]      = 1000
Vy    [N]      = 1000
Mx    [Nmm]    = 1e+06
My    [Nmm]    = 1e+06
T_stv [Nmm]    = 1e+06
T_wrp [Nmm]    = 1e+06
Bi    [Nmm^2]  = 1e+09

PARAMETERS
----------------------------------
A    [mm^2]  = 27741.3
Ax   [mm]    = 1072.44
Ay   [mm]    = 3.16162
Angle        = 89.63 deg

Ix   [mm^4]  = 8.7482e+08
Iy   [mm^4]  = 1.1236e+10
Ixy  [mm^4]  = -6.65028e+07
Imax[mm^4]   = 1.12364e+10

Ixp  [mm^4]  = 1.12364e+10
Iyp  [mm^4]  = 8.74393e+08
It   [mm^4]  = 2.85638e+09
Iw   [mm^6]  = 1.23862e+14
```

```
STIFFNESS
----------------------------------
EA    [N]      = 2.02789e+09
EIx   [Nmm^2]  = 6.39494e+13
EIy   [Nmm^2]  = 8.21352e+14
EIxy  [Nmm^2]  = -4.86136e+12
EImax [Nmm^2]  = 8.21383e+14

EIxp  [Nmm^2]  = 8.21383e+14
EIyp  [Nmm^2]  = 6.39182e+13

EIw   [Nmm^4]   = 9.05432e+18
GIt   [Nmm^2]  = 8.03082e+13

SHEAR CENTER
----------------------------------
Xs [mm]   = 5.77183
Ys [mm]   = 51.6396

SHEAR DEFORMATION FACTORS
----------------------------------
Kx    = 3.4895
Ky    = 1.50188
```

Appendix C

File formats

C.1 Massive *.msh* file format

```
$MeshFormat
version-number file-type data-size
$EndMeshFormat
$Nodes
number-of-nodes
node-number x-coord y-coord z-coord
...
$EndNodes
$Elements
number-of-elements
elm-number elm-type number-of-tags <tag> node-number-list
...
$EndElements
```

C.2 Thin-wall *.msh* file format

For regular two-node line elements the format is as follows:

```
$MeshFormat
version-number file-type data-size
$EndMeshFormat
$Nodes
number-of-nodes
node-number x-coord y-coord z-coord
...
$EndNodes
$Elements
number-of-elements
elm-number elm-type number-of-tags <tag> node-number-list
...
$EndElements
```

In the case of defining a material specific two-node line element, the element record changes to the following. Both element types may be used interchangeably, as long as the element type matches the number of line entries.

```
elm-number elm-type number-of-tags <tag> node-number-list elm-E elm-nu element-thickness
```

C.3 *.vtk* file format

```
DATASET POLYDATA
POINTS n dataType
p0x p0y p0z
p1x p1y p1z
...
p(n-1)x p(n-1)y p(n-1)z
VERTICES n size
numPoints 0, i0, j0, k0, ...
numPoints 1, i1, j1, k1, ...
...
numPoints n-1, in-1, jn-1, kn-1, ...
LINES n size
numPoints 0, i0, j0, k0, ...
numPoints 1, i1, j1, k1, ...
...
numPoints n-1, in-1, jn-1, kn-1, ...
POLYGONS n size
numPoints 0, i0, j0, k0, ...
numPoints 1, i1, j1, k1, ...
...
numPoints n-1, in-1, jn-1, kn-1, ...
TRIANGLE_STRIPS n size
numPoints 0, i0, j0, k0, ...
numPoints 1, i1, j1, k1, ...
...
numPoints n-1, in-1, jn-1, kn-1, ...
```

C.4 Drawing of polygons

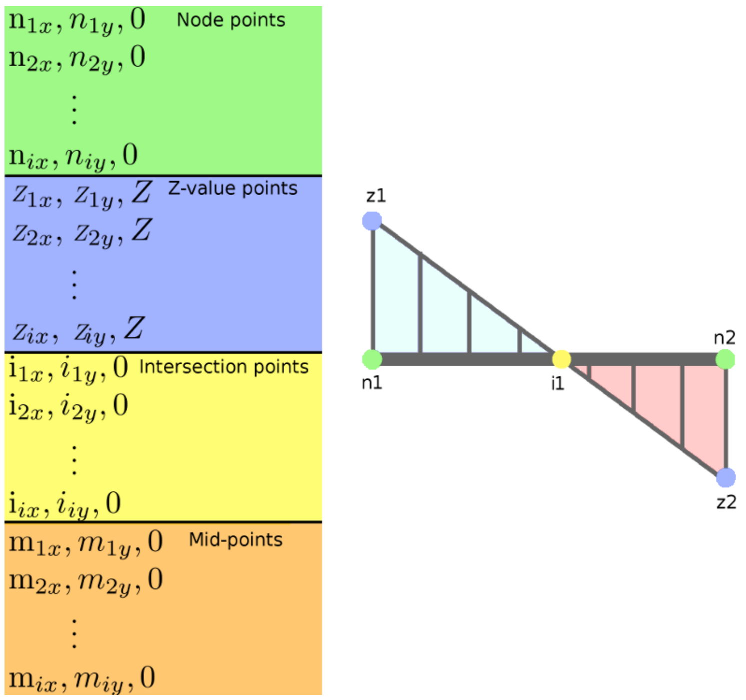


Figure C.1: Illustrates how polygons are made up using nodes, for linear stress distributions

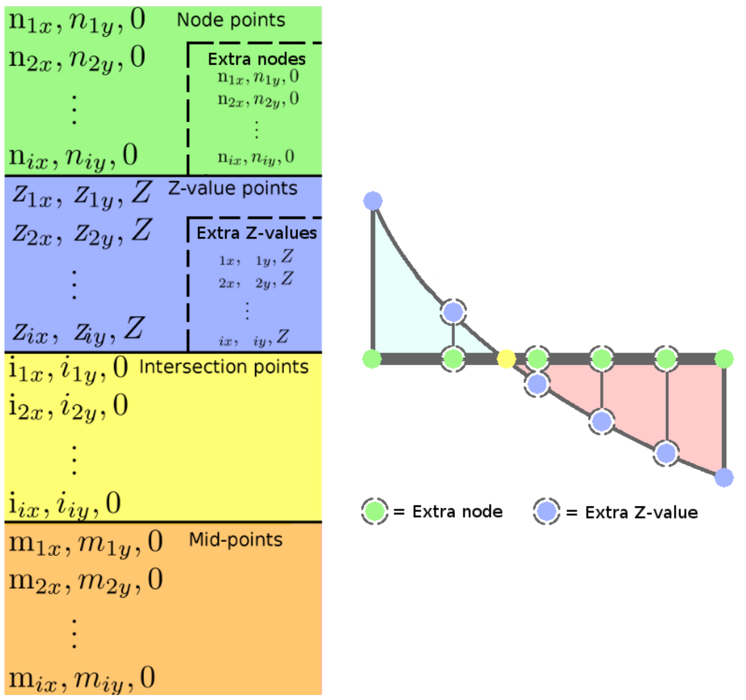


Figure C.2: Illustrates how polygons are made up using nodes, for quadratic stress distributions

C.5 .stl format

Outputting results in the "*.stl*" file format simply involves outputting results element wise. This is obtained by going through all elements, and outputting the nodal stress values for each element. However, we are not actually outputting the results in an *.stl* file format, we are still using the *.vtk* file format in order to visualize results using ParaView. Hence it is important to output the nodes element wise at the beginning of the file. This is done with a for-loop over the elements where the nodes of each element is outputted. This way, the nodes will be matched with their respective stress values. Algorithm 10 shows how stresses are stored to be outputted element wise, while Algorithm 11 shows how stresses are stored when outputted node wise.

Algorithm 10 Algorithm to store stress values element wise ("*.stl* file format")

```
Maintain a Vector of stress values and call it stress
int numberOfCornerNodes = 0
if (triangular element) then
    numberOfCornerNodes = 3
else if (quadrilateral element) then
    numberOfCornerNodes = 4
end if
for (each element e) do
    for (each node n) do
        stress(numberOfCornerNodes*e + n, 0) = stressValue
    end for
end for
```

Algorithm 11 Algorithm to store stress values node wise (regular *.vtk* file format)

```
Maintain a Vector of stress values and call it stress
for (each node n) do
    stress(n, 0) = stressValue
end for
```

Appendix D

Data Files

D.1 Scripts

The following two python scripts have been written for use in CCSC. They are both found under the *Script* folder.

1. **Display Arrows** - Used for thin-wall analysis to visualize the direction of every line segment. This is useful when interpreting results. Wherever numerical values are negative, the shear flow is in the opposite direction of what is indicated (i.e. positive element direction). Change color to "Solid Coloring - black" for best practice.

2. **DisplayCoordinateSystem** - Used for massive analysis to visualize original axis (indicated by arrows) and shear center (indicated by square). If scaled improperly, use "Scale Factor" manually to match size of cross-sections (set by default to 0.0005).

D.2 Mesh files

Table D.1: Mesh files used for convergence and running time analysis of massive cross-sections. Meshes listed are found under *ExampleMesh/Massive/Convergence* folder. Cross-section is a 100 × 100 mm^2 square

Filename	# of nodes	# of elements
T3Small	4	2
T3Medium	9	8
T3Large	25	32
T3XLarge	81	128
T3Huge	289	512
T3XHuge	1089	2048
T3XXHuge	4225	8192

Filename	# of nodes	# of elements
T6Small	9	2
T6Medium	25	8
T6Large	81	32
T6XLarge	289	128
T6Huge	1089	512
T6XHuge	4225	2048
T6XXHuge	16641	8192

Filename	# of nodes	# of elements
Q4Small	9	4
Q4Medium	25	16
Q4Large	81	64
Q4XLarge	289	256
Q4Huge	1089	1024
Q4XHuge	4225	4096

Filename	# of nodes	# of elements
Q9Small	25	4
Q9Medium	81	16
Q9Large	289	64
Q9XLarge	1089	256
Q9Huge	4225	1024
Q9XHuge	16641	4096

Table D.2: Mesh files used for comparison of linear and quadratic elements. Meshes listed are found under *ExampleMesh/Massive/Rectangle* folder. Cross-section is a 200 × 100 mm^2 rectangle

Filename	# of nodes	# of elements
T3Coarse	153	256
T3Refined	2145	4096

Filename	# of nodes	# of elements
T6Coarse	153	64
T6Refined	2145	1024

Filename	# of nodes	# of elements
Q4Coarse	153	128
Q4Coarse	2145	2048

Filename	# of nodes	# of elements
Q9Coarse	153	32
Q9Refined	2145	512

Table D.3: Additional massive meshes used for analysis. Meshes listed are found under *ExampleMesh/Massive* folder

Filename	**# of nodes**	**# of elements**
HalfCircleMixed	689	256
HalfCircleTriangular	1138	544
Hole	1104	512
Hook	697	320
IPE300	749	302
RHS	303	390
USP	4769	2304

Table D.4: Mesh files used for thin-wall analysis. Meshes listed are found under *ExampleMesh/Thinwall*

Filename	**# of line segments**
LBeam	2
Rectangle	4
IPE300	5
TwoRectangles	7
ComplexCrossSection	14
RHS	20
Airfoil	26

Appendix E

How to run CCSC

Run the Windows installer file *CCSC.msi* provided alongside this thesis. Once the installation is complete run the command line as an administrator, and navigate to the location of the installed files. Inside this folder the executable *Solver.exe* can be run as follows

```
Solver -f nameOfFile
```

All additional mesh files have to be placed inside the *ExampleMesh* folder. If the file is located in additional sub-folders below ExampleMesh, the entire path to the file needs to be given as the argument

```
Solver -f subFolder1/subFolder2/nameOfFile
```

For an entire list of commands the help command can be run

```
Solver -help
```

The results are written to the *Results* folder, if not ensure that the command line is running as an administrator. It should be noted that running CCSC requires the programming environment of Visual Studio to be installed on the computer.

Appendix F

Additional Information

F.1 Materials

Table F.1: Materials used in analysis

	Young's modulus [MPa]	**Poisson's ratio**	**Shear Modulus** [MPa]
Steel	210 000	0.3	80769.2
Aluminium 2024-T3[23]	73 100	0.3	28115.4

F.2 Original class diagram

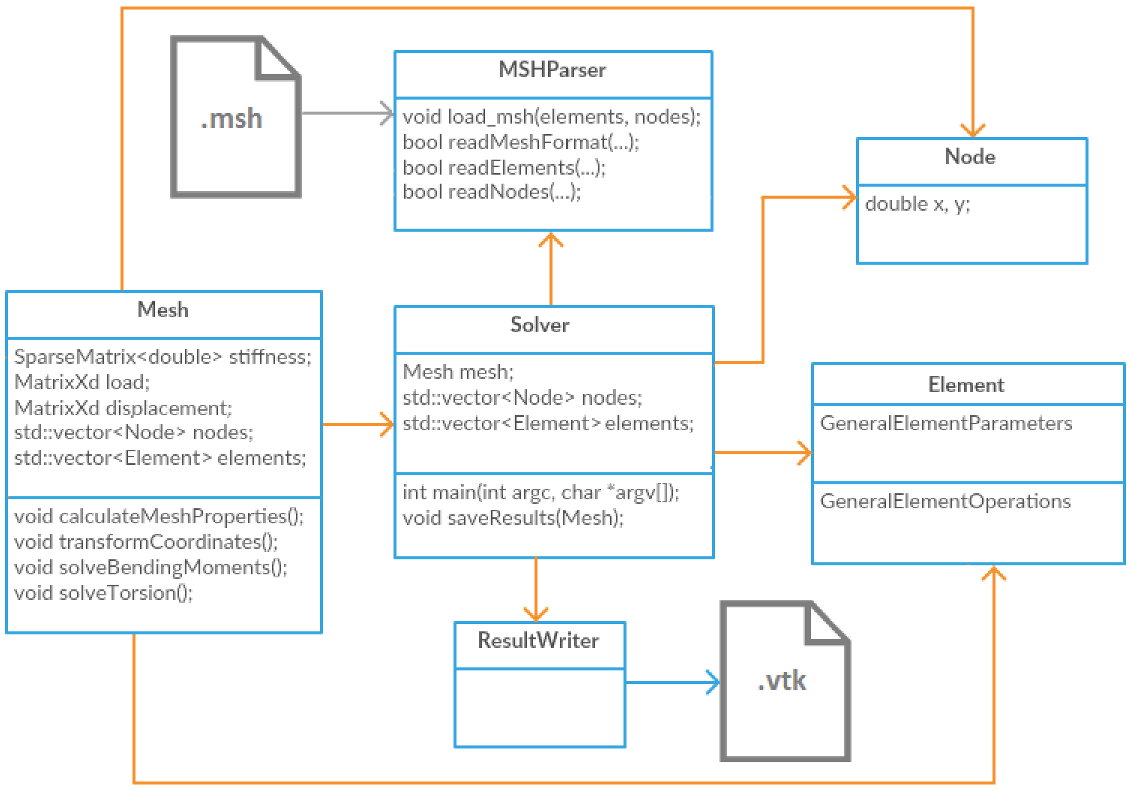


Figure F.1: UML class diagram showing the relationship between classes at the end of Strømstad's thesis

NTNU - NORGES TEKNISK-
NATURVITENSKAPELIGE UNIVERSITET
INSTITUTT FOR PRODUKTUTVIKLING
OG MATERIALER

MASTEROPPGAVE VÅR 2016
FOR
STUD.TECHN. SIMEN AAKHUS
OG
STUD.TECHN. SAMSON SVENDSEN

Analyseprogram for beregning av tverrsnittsparametre for komplekse tverrsnitt.

Analysis software for calculation of cross-section parameters for complex beam cross-sections.

Tverrsnittsdata for komplekse tverrsnitt som f.eks. et vindturbin-blad er vanskelig å beregne analytisk. Dynamiske beregninger av vindturbiner benytter ofte bjelkemodeller for turbin-bladene, og resultatene er avhengig av korrekte tverrsnittsdata.

Oppgaven tar sikte på å videreutvikle funksjonalitet til en applikasjon som beregner tverrsnittsparametre for komplekse tverrsnitt ved hjelp av elementmetoden. Følgende punkter ønskes å jobbe videre med:

- Teori og implementering av nye elementtyper:
  - 4 og 9-noders kvadrilaterale elementer for tykkvegget tverrsnitt.
  - 2-noders element for tynnvegget tverrsnitt.
  - 3-noders element for tynnvegget tverrsnitt.
- Støtte for kompositt materiale for de eksisterende elementene
- Evaluering av eksisterende klassestruktur og implementere eventuelle re-organiseringer.

Formelle krav:

Senest 3 uker etter oppgavestart skal et A3 ark som illustrerer arbeidet leveres inn. En mal for dette arket finnes på instituttets hjemmeside under menyen masteroppgave (https://www.ntnu.no/web/ipm/masteroppgave-ved-ipm). Arket skal også oppdateres en uke før innlevering av masteroppgaven.

Risikovurdering av forsøksvirksomhet skal alltid gjennomføres. Eksperimentelt arbeid definert i problemstilling skal planlegges og risikovurderes innen 3 uker etter utlevering av oppgavetekst. Konkrete forsøksvirksomhet som ikke omfattes av generell risikovurdering skal spesielt vurderes før eksperimentelt arbeid utføres. Risikovurderinger skal signeres av veileder og kopier skal inngå som vedlegg til oppgaven.

Besvarelsen skal ha med signert oppgavetekst, og redigeres mest mulig som en forskningsrapport med et sammendrag på norsk og engelsk, konklusjon, litteraturliste, innholdsfortegnelse, etc. Ved utarbeidelse av teksten skal kandidaten legge vekt på å gjøre teksten oversiktlig og velskrevet. Med henblikk på lesning av besvarelsen er det viktig at de nødvendige henvisninger for korresponderende steder i tekst, tabeller og figurer anføres på begge steder. Ved bedømmelse legges det stor vekt på at resultater er grundig bearbeidet, at de oppstilles tabellarisk og/eller grafisk på en oversiktlig måte og diskuteres utførlig.

Besvarelsen skal leveres i elektronisk format via DAIM, NTNUs system for Digital arkivering og innlevering av masteroppgaver.

[illegible]
Torgeir Welo
Instituttleder

Bjørn Haugen
Bjørn Haugen
Faglærer

NTNU
Norges teknisk-naturvitenskapelige universitet
Institutt for produktutvikling og materialer

NTNU HMS	Kartlegging av risikofylt aktivitet	Utarbeidet av	Nummer	Dato
		HMS-avd.	HMSRV2601	22.03.2011
		Godkjent av		Erstatter
		Rektor		01.12.2006

Enhet: IPM **Dato:** 3. Feb. 2016

Linjeleder: Torgeir Welo

Deltakere ved kartleggingen (m/ funksjon):
(Ansv. veileder, student, evt. medveiledere, evt. andre m. kompetanse)

Kort beskrivelse av hovedaktivitet/hovedprosess: Masteroppgave student xx. Tittel på oppgaven.

Er oppgaven rent teoretisk? (JA/NEI): JA *«JA» betyr at veileder innestår for at oppgaven ikke inneholder noen aktiviteter som krever risikovurdering. Dersom «JA»: Beskriv kort aktiviteten i kartleggingskjemaet under. Risikovurdering trenger ikke å fylles ut.*

Signaturer: *Ansvarlig veileder:* Bjørn Haugen *Student:* Simen Aakhus

ID nr.	Aktivitet/prosess	Ansvarlig	Eksisterende dokumentasjon	Eksisterende sikringstiltak	Lov, forskrift o.l.	Kommentar

NTNU HMS	Kartlegging av risikofylt aktivitet	Utarbeidet av	Nummer	Dato
		HMS-avd.	HMSRV2601	22.03.2011
		Godkjent av		Erstatter
		Rektor		01.12.2006

Enhet: IPM **Dato:** 03/02/16

Linjeleder: Torgeir Welo

Deltakere ved kartleggingen (m/ funksjon):
(Ansv. veileder, student, evt. medveiledere, evt. andre m. kompetanse)

Kort beskrivelse av hovedaktivitet/hovedprosess: Masteroppgave student xx. Tittel på oppgaven.

Er oppgaven rent teoretisk? (JA/NEI): Ja *«JA» betyr at veileder innestår for at oppgaven ikke inneholder noen aktiviteter som krever risikovurdering. Dersom «JA»: Beskriv kort aktiviteteten i kartleggingskjemaet under. Risikovurdering trenger ikke å fylles ut.*

Signaturer: *Ansvarlig veileder:* Bjørn Haugen *Student:* [illegible]

ID nr.	Aktivitet/prosess	Ansvarlig	Eksisterende dokumentasjon	Eksisterende sikringstiltak	Lov, forskrift o.l.	Kommentar