NTNU
Det skapende universitet

Applikasjon for beregning av tverrsnittsparametre for komplekse bjelketverrsnitt

Kristian Strømstad

Produktutvikling og produksjon
Innlevert: juni 2014
Hovedveileder: Bjørn Haugen, IPM

Norges teknisk-naturvitenskapelige universitet
Institutt for produktutvikling og materialer

NORGES TEKNISK-
NATURVITENSKAPELIGE UNIVERSITET
INSTITUTT FOR PRODUKTUTVIKLING
OG MATERIALER

MASTEROPPGAVE VÅR 2014

FOR

STUD.TECHN. KRISTIAN STRØMSTAD

APPLIKASJON FOR BEREGNING AV TVERRSNITTSPARAMETRE FOR KOMPLEKSE BJELKETVERRSNITT
Software application for computation of parameters of 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 utvikle funksjonalitet mot en applikasjon som skal beregne tverrsnittsparametre for komplekse tverrsnitt ved hjelp av elementmetoden. Det er ønskelig at applikasjonen også kan håndtere kompositttmaterialer.

Applikasjonen bør også kunne beregne spenningstilstanden til tverrsnittet basert på spenningsresultanter som moment, skjærkraft og aksialkraft. Applikasjonen bør være enkel å bruke da en ønsker å benytte den i undervisningssammenheng. Applikasjonen bør bygge på enkle småprogrammer som lar seg benytte effektivt i skript dersom en skal utføre beregningene for mange forskjellige tverrsnitt.

Oppgaven har særlig fokus på den numeriske elementmetode delen av applikasjonen. Grensesnittet mellom den numeriske koden og det grafiske brukergrensesnittet må defineres i samarbeid med annen masteroppgave som har dette som hovedfokus.

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 (http://www.ntnu.no/ipm/masteroppgave). Arket skal også oppdateres en uke før innlevering av masteroppgaven.

Arbeidet i masteroppgaven skal risikovurderes. Hovedaktiviteter som er kjent/planlagt skal risikovurderes ved oppstart og skjema skal leveres innen 3 uker etter utlevering av oppgavetekst. Alle prosjekt skal vurderes, også de som kun er teoretiske og virtuelle. Skjemaet må signeres av veileder. Risikovurdering er en løpende dokumentasjon og skal gjøres før oppstart av enhver aktivitet som KAN være forbundet med risiko. Kopi av signert risikovurdering skal være inkludert i vedlegg ved levering av rapport

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.

Torgeir Welo
Instituttleder

Bjørn Haugen
Faglærer

NTNU
Norges teknisk-
naturvitenskapelige universitet
Institutt for produktutvikling
og materialer

Sammendrag

Rapporten beskriver arbeidet som er gjennomført ved NTNU i vårsemesteret 2014 der det er utviklet en applikasjon for å gjøre elementanalyse av komplekse bjelketverrsnitt. Applikasjonen kan beregne spenningsfordelinger over vilkårlige tverrsnitt som er utsatt for torsjonsmoment eller bøyemomenter. Det er skrevet i C++ og parallelt med denne oppgaven har at annet prosjekt tatt for seg utviklingen av et brukergrensesnitt for applikasjonen som er basert på visual toolkit-platformen VTK.

Tester har blitt gjennomført for å validere resultatene og disse har vist seg å stemme tilfredsstillende godt med virkeligheten. For torsjonsanalysen avhenger presisjonen langs randen av geometrien av et fint mesh. I bøyemomentsanalysen skyldes eventuelle avvik avrundingsfeil.

Abstract

This report describes the work conducted at NTNU in the spring of 2014, where a software application was developed which performs finite element analysis on complex beam cross sections. This application can calculate the stress distributions across any meshed cross section that is subjected to a torsional load or bending moments. It is developed in the C++ programming language and parallel to this project the development of a user interface for the application based on the visual toolkit VTK platform has been under way.

Tests have been conducted to verify the results from the application and they have been found to match the real value to a reasonable degree. For the torsional analysis, a correct result in the boundaries of the mesh is dependent on a fairly fine mesh. For the bending analysis any deviations are due to rounding errors.

Forord

Denne rapporten beskriver Kristian Strømstads avsluttende arbeid ved masterstudiet i Produktutvikling og Produksjon ved NTNU våren 2014.

Jeg vil rette en takk til Bjørn Haugen for god veiledning gjennom oppgaven.

Innhold

Figurer

Kapittel 1

Innledning

1.1 Bakgrunn for prosjektet

Denne oppgaven er utarbeidet på vegne av institutt for produktutvikling og materialer (IPM) ved fakultet for ingeniørvitenskap og teknologi (IVT) ved NTNU. Målet er en programpakke for å beregne tverrsnittsparametre for bjelketverrsnitt som deretter kan benyttes som bjelkemodeller i for eksempel dynamiske analyser hvor det ikke alltid er hensiktsmessig å modellere hele bjelketverrsnittet. Figur 1.1 viser et bjelketverrsnitt med de laster det er aktuelt å beregne.

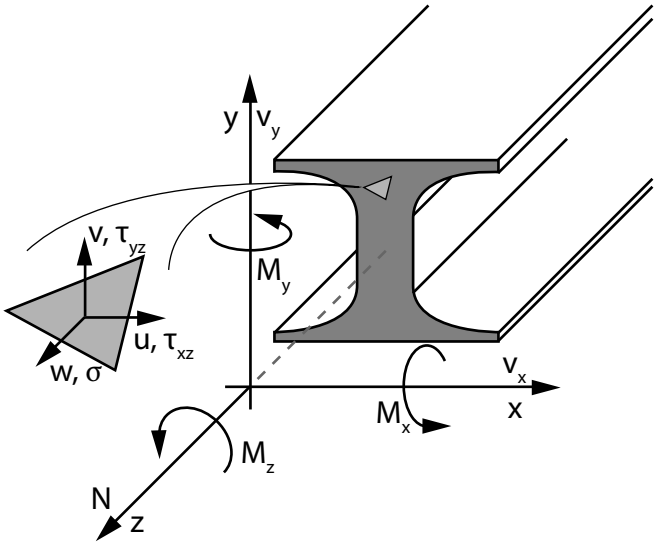
Denne oppgaven har fokus på den numerisk elementmetodedelen av programpakken og utvikling av de metodene som benyttes i analysene. Dette er skilt ut som en separat applikasjon som kan brukes for seg selv, eller sammen med de øvrige delene.

1.2 Programpakkens oppbygging

Programpakken består av tre delprogrammer. «Mesher», «solver» og et brukergrensesnitt.

1.2.1 Brukergrensesnitt

Utviklingen av brukergrensesnittet er beskrevet i masteroppgave av Askevold[2]. Brukergrensesnittet lar brukeren velge om de vil importere et ferdig generert mesh, eller om de vil lage en tegning som deretter behandles av mesheren. Mesh som genereres i brukergrensesnittet kan endres og justeres av brukeren for å oppnå ønsket mesh-kvalitet. Når soveren har kjørt åpnes resultatfilen og brukeren kan velge hvilket resultat de ønsker å plotte.



Figur 1.1: Bjelketverrsnitt med laster

1.2.2 Mesher

Mesheren er en tilpasning av den åpne programvaren Gmsh. Gmsh har funksjonaliteter for meshing med de fleste typer element og har også moduler for geometri, solving og postprosessering, men er her begrenset til meshing av 2D-mesh med trekantelementer.

1.2.3 Solver

Selve analysen blir utført av solveren. Den kjøres enten fra brukergrensesnittet eller fra kommandovindu og tar mesh-filen som argument. Solveren beregner de spenningsfordelinger som følger av fblir orskjellige enhetslaster og beregner meshets hovedakser. Resultatene skrives til en resultatfil som så leses inn av brukergrensesnittet.

1.3 Status ved oppstart

Arbeidet med programmet ble påbegynt i forbindelse med fordypningsprosjekt i samarbeid med to andre studenter høsten 2013. Da denne oppgaven ble påbegynt bestod solveren av en meshparser, en klassestuktur med en Mesh-klasse, Element-klasse og en Node-klasse som sammen beskriver meshet i programmet. Noen funksjoner var på plass, som beregning av tverrsnittsegenskaper som ikke krever elementmetodeløsning. Dog med noen feil i integrasjonsmetodene. Klas-

sen som skriver resultatfilen var også opprettet men skrev da til et filformat som senere har blitt valgt bort.

1.4 Omfang og forutsetninger

Funksjonalitet som ønskes av solveren er beregning av:

- spenningsfordeling ved aksialkraft og rene bøyemomenter
- hovedaksers orientering
- spenningsfordelinger ved torsjonsbelastning
- spenningsfordeling ved skjærbelastning
- skjærsenter
- parametre for kompositttstrukturer

Det er valgt å ikke fokusere på kompositttmaterialer i denne oppgaven.

Ved beregning av tverrsnittsdata har man valget mellom to beregningsmodeller. Tynnvegget tverrsnitt der meshet består av 1D-elementer, og massive tverrsnitt som meshes med 2D-elementer. Oppgaven tar for seg sistnevnte.

I tillegg er det et mål for kandidaten, som ikke har bakgrunn fra IT, å tilegne seg mye kunnskap om programmering, samt fordype seg i elementmetoden.

1.5 Metoder

Programmet utvikles i C++ med Microsoft Visual Studio 2012. Parallelt med arbeidet jobbes det mye med å bygge nødvendige ferdigheter innen programmering. Ellers er oppgaven et studie gjennom praksis av elementmetoden der mye av kunnskapen kommer fra prøving og feiling, og refleksjon over testresultater.

Prosjektet er et samarbeidsprosjekt og utviklingen av solveren koordineres med utviklingen av brukergrensesnittet for å sikre kompatibilitet og god funksjonalitet.

Kapittel 2

Teori

Dette kapittelet gir det teoretiske grunnlaget for analysene som gjennomføres av programmet. Dette kapittelet er i stor grad basert på teori fra BELL[1].

Selv om ikke alle analysene baserer seg på elementmetoden, utføres alle på det samme meshet og de samme elementene. Det er derfor hensiktsmessig å diskutere prinsippene bak elementmetoden før vi går inn på det som gjelder spesielt i disse analysene. De påfølgende seksjonene antar imidlertid en grunnleggende kjennskap til elementmetoden .

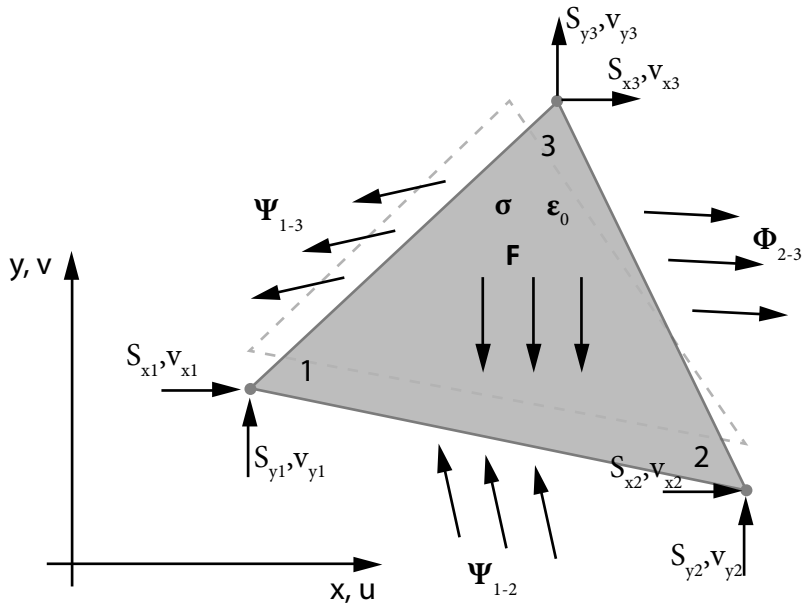
2.1 Elementmetoden

Et element består i prinsippet av et gitt antall noder. Tilstanden internt i elementet styres av tilstanden til nodene. I hver node defineres belastninger og forskyvninger som korresponderer med nodens frihetsgrader. Disse samles henholdsvis i vektorene $\mathbf{S}$ og $\mathbf{v}$, der $\mathbf{S}^T = \begin{bmatrix} S_{1x} & S_{1y} & S_{2x} & \dots & S_{ny} \end{bmatrix}$ og $\mathbf{v}^T = \begin{bmatrix} v_{1x} & v_{1y} & v_{2x} & \dots & v_{ny} \end{bmatrix}$ for et element med n noder og $2n$ frihetsgrader. Forholdet mellom belastning og forskyvning defineres ved $\mathbf{S}$=$\mathbf{kv}$ der $\mathbf{k}$ er stivhetsmatrisen til elementet. Å bestemme stivhetsmatrisen er den mest sentrale oppgaven i elementmetoden og man må gjøre en rekke antagelser for å finne en rimelig tilnærming.

Det defineres et forskyvningsfelt internt i elementet som er en funksjon av forskyvningen i nodene. Dette kan formuleres som

$$\begin{bmatrix} u \\ v \end{bmatrix} = \mathbf{u} = \mathbf{Nv} \tag{2.1}$$

der u og v er forskyvning i henholdsvis x- og y-retning av et vilkårlig punkt i elementet. Utledning av formfunksjonen $\mathbf{N}$ for trekantelementet er beskrevet i seksjon 2.3.



Figur 2.1: Krefter og spenninger i et element (inspirasjon fra Bell[1] s.124)

2.2 Virtuelle forskyvningers prinsipp

Virtuelle forskyvningers prinsipp (VFP) innfører et sett virtuelle forskyvninger ($\tilde{\mathbf{v}}$) og sier at «Det virtuelle arbeidet utført av de virkelige eksterne kreftene over den virtuelle forskyvningen er lik det virtuelle arbeidet utført av de virkelige interne kreftene over den virtuelle tøyningen». Indre virtuelt arbeid er lik ytre virtuelt arbeid ($W_i = W_y$). Figur 2.2 viser kreftene og spennigene som opptrer i et element. Elementet i figuren har to naboelementer, langs sidekantene 1-2 og 1-3, mens 2-3 er en fri sidekant. Ψ representerer påvirkning fra naboelementene, Φ er påvirkning fra omgivelsene på den frie kanten, $\mathbf{F}$ er volumkrefter og σ og ε_0 er den interne spenningen og initiell tøyning. Nodebelatningene $\mathbf{S}$ kan betraktes som resultanter av påkjenningen fra naboelementene. Ligningen for arbeidet blir da

$$\tilde{\mathbf{v}}^T\mathbf{S} + \int_V \tilde{\mathbf{u}}^T\mathbf{F}dV + \int_S \tilde{\mathbf{u}}^T\Phi dS = \int_V \tilde{\varepsilon}^T\sigma dV \tag{2.2}$$

Definerer $\tilde{\mathbf{u}} = \mathbf{N}\tilde{\mathbf{v}}$ og $\tilde{\varepsilon} = \Delta\tilde{\mathbf{u}} = \Delta\mathbf{N}\tilde{\mathbf{u}} = \mathbf{B}\tilde{\mathbf{u}}$. Spenningene i elementet er gitt ved $\sigma = \mathbf{C}(\varepsilon - \varepsilon_0)$. Ved å substituere dette inn i ligning 2.2 får man

$$\tilde{\mathbf{v}}^T\mathbf{S} + \int_V \tilde{\mathbf{v}}^T\mathbf{N}^T\mathbf{F}dV + \int_S \tilde{\mathbf{v}}^T\mathbf{N}^T\Phi dS = \int_V \tilde{\mathbf{v}}^T\mathbf{B}^T\mathbf{C}(\mathbf{B}\mathbf{v} - \varepsilon_0)dV$$

$$\mathbf{S} = \int_V \mathbf{B}^T\mathbf{C}\mathbf{B}\mathbf{v}dV - \int_V \mathbf{B}^T\mathbf{C}\varepsilon_0 dV - \int_V \mathbf{N}^T\mathbf{F}dV - \int_S \mathbf{N}^T\Phi dS$$

Dette kan skrives som

$$\mathbf{S} = \mathbf{kv} + \mathbf{S}^0 \tag{2.3}$$

der

$$\mathbf{k} = \int_V \mathbf{B}^T \mathbf{C} \mathbf{B} dV \tag{2.4}$$

er stivhetsmatrisen og

$$\mathbf{S}^0 = \mathbf{S}^0_{\varepsilon 0} + \mathbf{S}^0_F + \mathbf{S}^0_\Phi$$

er den konsistente lastvektoren som består av initiell tøyning, volumkrefter og overflatekrefter.

$$\begin{aligned} \mathbf{S}^0_{\varepsilon 0} &= -\int_V \mathbf{B}^T \mathbf{C} \varepsilon_0 dV \\ \mathbf{S}^0_F &= -\int_V \mathbf{N}^T \mathbf{F} dV \\ \mathbf{S}^0_\Phi &= -\int_S \mathbf{N}^T \Phi dS \end{aligned} \tag{2.5}$$

2.3 Interpolasjon

I seksjon 2.1 ble $\mathbf{N}$ definert som matrisen som kobler forskyvningen av et vilkårlig punkt i elementet til forskyvningen av nodene. $\mathbf{N}$ må bestemmes slik at $\mathbf{N}(x_i, y_i) = \begin{bmatrix} x_i & y_i \end{bmatrix}$. Arealkoordinater $\begin{bmatrix} \zeta_1 & \zeta_2 & \zeta_3 \end{bmatrix}$ er definert slik at $\zeta_i = 0$ representerer linjen som går fra node j til node k i trekanten (i j k) og $\zeta_i = 1$ er linjen som går gjennom node i. Se figur 2.3. Koordinatene til punktet P i figuren er gitt av

$$\begin{aligned} x &= \zeta_1 x_1 + \zeta_2 x_2 + \zeta_3 x_3 \\ y &= \zeta_1 y_1 + \zeta_2 y_2 + \zeta_3 y_3 \end{aligned} \tag{2.6}$$

Ved å innføre $x_{ij} = x_i - x_j$ og $y_{ij} = y_i - y_j$ kan disse formlene inverseres og skrives på matriseform som

$$\begin{bmatrix} \zeta_1 \\ \zeta_2 \\ \zeta_3 \end{bmatrix} = \frac{1}{2A} \begin{bmatrix} y_{23} & x_{32} & (x_2 y_3 - x_3 y_2) \\ y_{31} & x_{13} & (x_3 y_1 - x_1 y_3) \\ y_{12} & x_{21} & (x_1 y_2 - x_2 y_1) \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \tag{2.7}$$

der A er arealet til elementet. For at dette skal stemme er det forøvrig en forutsetning at elementene er nummerert fra 1 til 3 i rekkefølge mot urviseren. Interpolasjonen for en frihetsgrad kan uttykkes som $u = \mathbf{Nv}$ der $\mathbf{v}^T = \begin{bmatrix} v_1 & v_2 & v_3 \end{bmatrix}$. Formfunksjonen vil i dette tilfelle være gitt av

$$\mathbf{N} = \begin{bmatrix} \zeta_1 & \zeta_2 & \zeta_3 \end{bmatrix} \tag{2.8}$$

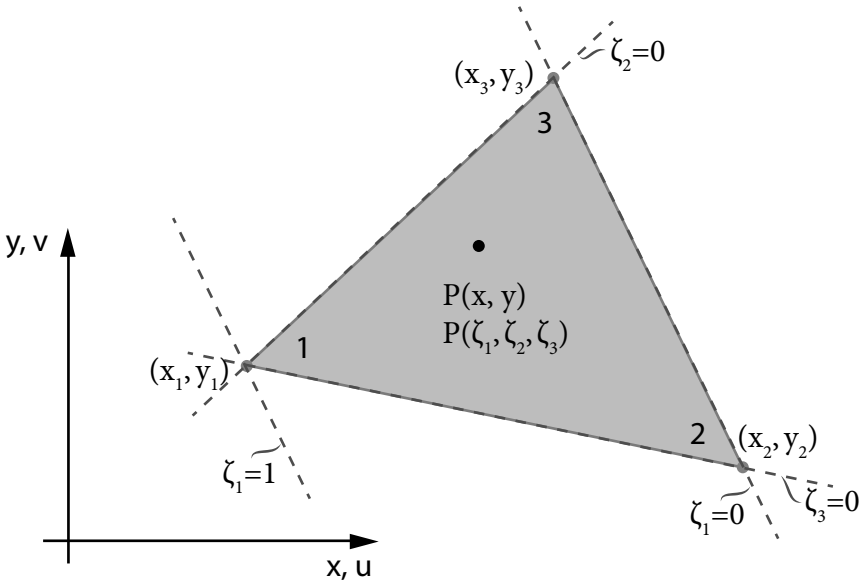
Det er også nødvendig å definere $\mathbf{B}$-matrisen fra seksjon 2.2. Denne inneholder de partiellderiverte av formfunksjonen $\mathbf{N}_{,x}$ og $\mathbf{N}_{,y}$. Formel 2.6 og 2.7 gir

$$\frac{\partial x}{\partial \zeta_i} = x_i \qquad \frac{\partial y}{\partial \zeta_i} = y_i \tag{2.9}$$

og

$$\frac{\partial \zeta_i}{\partial x} = \frac{y_{jk}}{2A} \qquad \frac{\partial \zeta_i}{\partial y} = \frac{x_{kj}}{2A} \tag{2.10}$$

$$\begin{aligned} \mathbf{N}_{,x} &= \tfrac{1}{2A} \begin{bmatrix} y_{23} & y_{31} & y_{12} \end{bmatrix} \\ \mathbf{N}_{,y} &= \tfrac{1}{2A} \begin{bmatrix} x_{32} & x_{13} & x_{21} \end{bmatrix} \end{aligned} \tag{2.11}$$



Figur 2.2: Arealkoordinater

2.4 Numerisk integrasjon

Numerisk integrasjon er metoder for å, ved hjelp av diskrete størrelser, finne tilnærminsverdier av integraler som ellers er vanskelig å evaluere eksakt.

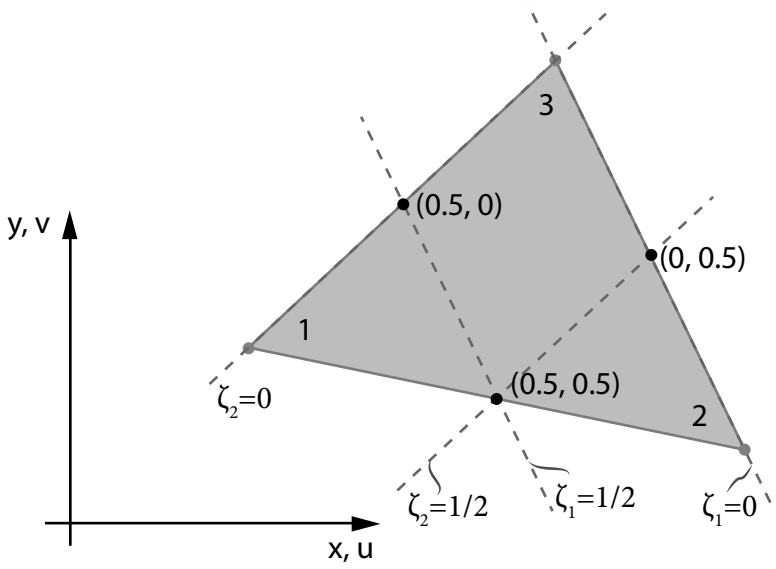
I forbindelse med elementmetoden er gauss-integrasjon mye anvendt. Ved Gauss-integrasjon beregnes integralet som en sum av vektede funksjonsverdier.

$$I = \int f(x)dx = \Sigma_{i=1}^{n} W_i f(x_i)$$

der $f(x_i)$ er funksjonsverdien i integrasjonspunkt i, W_i er et korresponderende vekttall og n er antall integrasjonspunkter. For trekantelementer med arealkoordinater blir dette

$$I = \int_A f(x,y)dA = \Sigma_{i=1}^{n} W_i f(x_i, y_i) = A\Sigma_{i=1}^{n} W_i f(\zeta_{1i}, \zeta_{2i})$$

Generelt gjelder at n antall integrasjonspunkter evaluerer integraler av orden $2n-1$ eksakt. Med 3 integrasjonspunkter får man altså eksakt løsning for 5.-ordens funksjoner. I dette arbeidet blir det nødvendig å integrere 2.-ordens funksjoner som altså ikke lar seg integrere eksakt med ett punkt. Figur 2.4 viser integrasjonspunktene for 3-punkts integrasjon over et trekantelement. Punktene er plassert midt på hver sidekant i punktene $(\zeta_1, \zeta_2) = (0,0.5)$, $(0.5,0)$ og $(0.5,0.5)$ og vekttallet $W_i = \frac{1}{3}$ for alle punktene[4].



Figur 2.3: Gausspunkter for trekantelement

2.5 Spenninger

Med stegene beskrevet i de foregående seksjonene kan man sette opp systemligningen

$$\mathbf{R} = \mathbf{Kr}$$

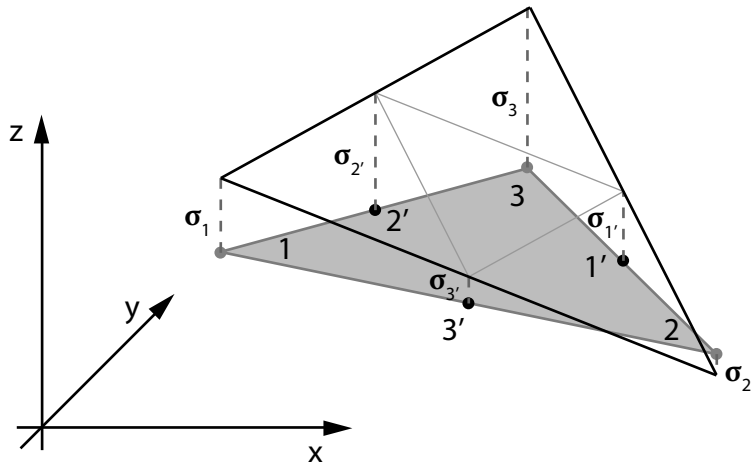
der

$$\mathbf{R} = \Sigma_{i=1}^{e} \mathbf{S}_i^0$$

og

$$\mathbf{K} = \Sigma_{i=1}^{e} \mathbf{k}_i$$

og få en løsning for forskyvningsvektoren $\mathbf{r}$ for hele systemet og dermed elementforskyvningene $\mathbf{v}_e$. Spenningen i et element er gitt av $\sigma = \mathbf{C}(\mathbf{Bv} - \varepsilon_0)$. Forskyvningene vil ha kontinuitet mellom elementene. Dette gjelder imidlertid ikke spenningene hvor verdien vil variere avhengig av hvilket element man beregner spenningen fra. Dette er et resultat av at spenningene er funksjon av de deriverte av formfunksjonen og derfor vil være av en lavere orden enn tøyningene. For en node som er koblet til flere elementer må man derfor beregne spenningen fra alle elementene, og finne gjennomsnittet, for å få en verdi som nærmer seg den virkelige verdien. Presisjonen til beregningene er også avhengig av hvor i elementet spenningene beregnes. For best mulig presisjon bør beregningene generelt utføres i integrasjonspunktene av en grad lavere enn full integrasjon og ekstrapoleres til nodene. For trekantelementet med 3 noder blir dette ett punkt



Figur 2.4: Ekstrapolering til nodeverdier

og dermed antas en konstant spenning over elementet som ikke vil gi en realistisk spenningsfordeling. I dette tilfellet benyttes derfor tre integrasjonspunkter til beregning av spenningene.

Ekstrapoleringen fra punktene $1'$, $2'$ og $3'$ som vist i figur2.5 er triviell og er gitt av

$$\sigma_i = \sigma_j' + \sigma_k' - \sigma_i' \tag{2.12}$$

2.6 Bøyemomenter

Spenningstilstanden ved bøyemomenter beregnes som vanlig ved hjelp av annet arealmomentene og sentrifugalmomentet med hensyn til x- og y-aksen

$$I_x = \int_A y^2 dA$$

$$I_y = \int_A x^2 dA$$

$$I_{xy} = \int_A xy dA$$

For et element som har arealsenter i avstand a langs x-aksen og b langs y-aksen fra tverrsnittets arealsenter blir bidraget til annet arealmomentene

$$I_x^e = \int_{A_e} y^2 dA_e + b^2 A_e$$

$$I_y^e = \int_{A_e} x^2 dA_e + a^2 A_e$$

$$I_{xy}^e = \int_{A_e} xy dA_e + abA_e$$

og tverrsnittets annet arealmomenter og sentrifugalmoment er gitt av

$$I_x = \Sigma_{i=1}^e I_{xi}$$
$$I_y = \Sigma_{i=1}^e I_{yi}$$
$$I_{xy} = \Sigma_{i=1}^e I_{xyi}$$

Tverrsnittets maksimale annet arealmoment

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

blir deretter kalkulert og hovedaksenes orientering med de globale aksene er gitt av

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

2.7 Torsjonsmoment

Ved tosjonsmoment er forskyvningene i 3 dimensjoner gitt av

$$\begin{aligned} u &= -\theta yz \\ v &= \theta xz \\ w &= \theta\Psi(x,y) \end{aligned} \tag{2.13}$$

der $\theta = \frac{d\phi}{dz}$ og ϕ er rotasjonen av tverrsnittet som funksjon av z. $\phi = \theta z$. θ er konstant og antas for enkelthets skyld å være lik 1. Tverrsnittet antas videre å rotere som et stivt legeme uten deformasjon i xy-planet. Forskyvningene $w = \Psi(x,y)$ er derfor den eneste forskyvningen som bidrar til spenningene. Ved ren tosjon oppstår ingen normalspenninger og skjærspenningene $\tau_{xy} = \tau_{yx}$ i tverrsnittsplanet er lik 0. Dermed er også skjærtøyningen $\gamma_{xy} = 0$.

$$\begin{aligned} \gamma_{xz} &= w_{,x} + u_{,z} = \Psi_{,x} - y = \tfrac{1}{G}\tau_{xz} \\ \gamma_{yz} &= w_{,y} + v_{,z} = \Psi_{,y} + x = \tfrac{1}{G}\tau_{yz} \end{aligned} \tag{2.14}$$

På matriseform blir dette

$$\varepsilon = \begin{bmatrix} \gamma_{xz} \\ \gamma_{yz} \end{bmatrix} = \theta\left(\begin{bmatrix} \Psi_{,x} \\ \Psi_{,y} \end{bmatrix} + \begin{bmatrix} -y \\ x \end{bmatrix}\right) \tag{2.15}$$

For et element med forskyvningsvektoren $\mathbf{v}^T = \begin{bmatrix}\Psi_1 & \Psi_2 & \Psi_3\end{bmatrix}$ kan dette skrives som

$$\varepsilon = \mathbf{B}\mathbf{v} + \varepsilon_0 \tag{2.16}$$

der

$$\mathbf{B} = \begin{bmatrix} \mathbf{N}_{,x} \\ \mathbf{N}_{,y} \end{bmatrix}$$

og

$$\varepsilon_0 = \begin{bmatrix} \mathbf{0} & -\mathbf{N} \\ \mathbf{N} & \mathbf{0} \end{bmatrix} \begin{bmatrix} \mathbf{x} \\ \mathbf{y} \end{bmatrix}$$

Spenningsvektoren

$$\sigma = \begin{bmatrix} \tau_{xz} \\ \tau_{yz} \end{bmatrix} = \begin{bmatrix} G & 0 \\ 0 & G \end{bmatrix} \begin{bmatrix} \gamma_{xz} \\ \gamma_{yz} \end{bmatrix} = \mathbf{G}\varepsilon = \mathbf{G}(\mathbf{B}\mathbf{v} + \varepsilon_\mathbf{0})$$

Dermed kan systemlignigen settes opp med utgangspunkt i formel 2.3, 2.4 og 2.5.

$$\begin{aligned} \mathbf{k} &= \int_A \mathbf{B}^T \mathbf{G} \mathbf{B} dA \\ \mathbf{S}^0 &= -\int_A \mathbf{B}^T \mathbf{G} \varepsilon_0 dA \\ \mathbf{K} &= \Sigma_\mathbf{e} \mathbf{k}_i \\ \mathbf{S} &= \Sigma_e \mathbf{S}^0 \end{aligned} \tag{2.17}$$

Tilstrekkelige grensebetingelser oppnås enkelt ved å fastholde en tilfeldig node i. Dette gjøres ved å sette rad i og kolonne i lik null i $\mathbf{K}$ og $\mathbf{K}$(i, i)=1. Samtidig settes $\mathbf{S}$(i)=0. Når en løsning for forskyvningen i elementene er funnet beregnes spenningene på samme måte som beskrevet i seksjon 2.5.

$$\begin{bmatrix} \tau_{xz} \\ \tau_{yz} \end{bmatrix} = G\theta \left(\begin{bmatrix} \mathbf{N}_{,x} \\ \mathbf{N}_{,y} \end{bmatrix} \mathbf{v} + \begin{bmatrix} \mathbf{0} & -\mathbf{N} \\ \mathbf{N} & \mathbf{0} \end{bmatrix} \begin{bmatrix} \mathbf{x} \\ \mathbf{y} \end{bmatrix} \right) \tag{2.18}$$

Kapittel 3

Kode

Alle beregningene som utføres i programmet tar utgangspunkt i mesh-filen som er gitt som argument.

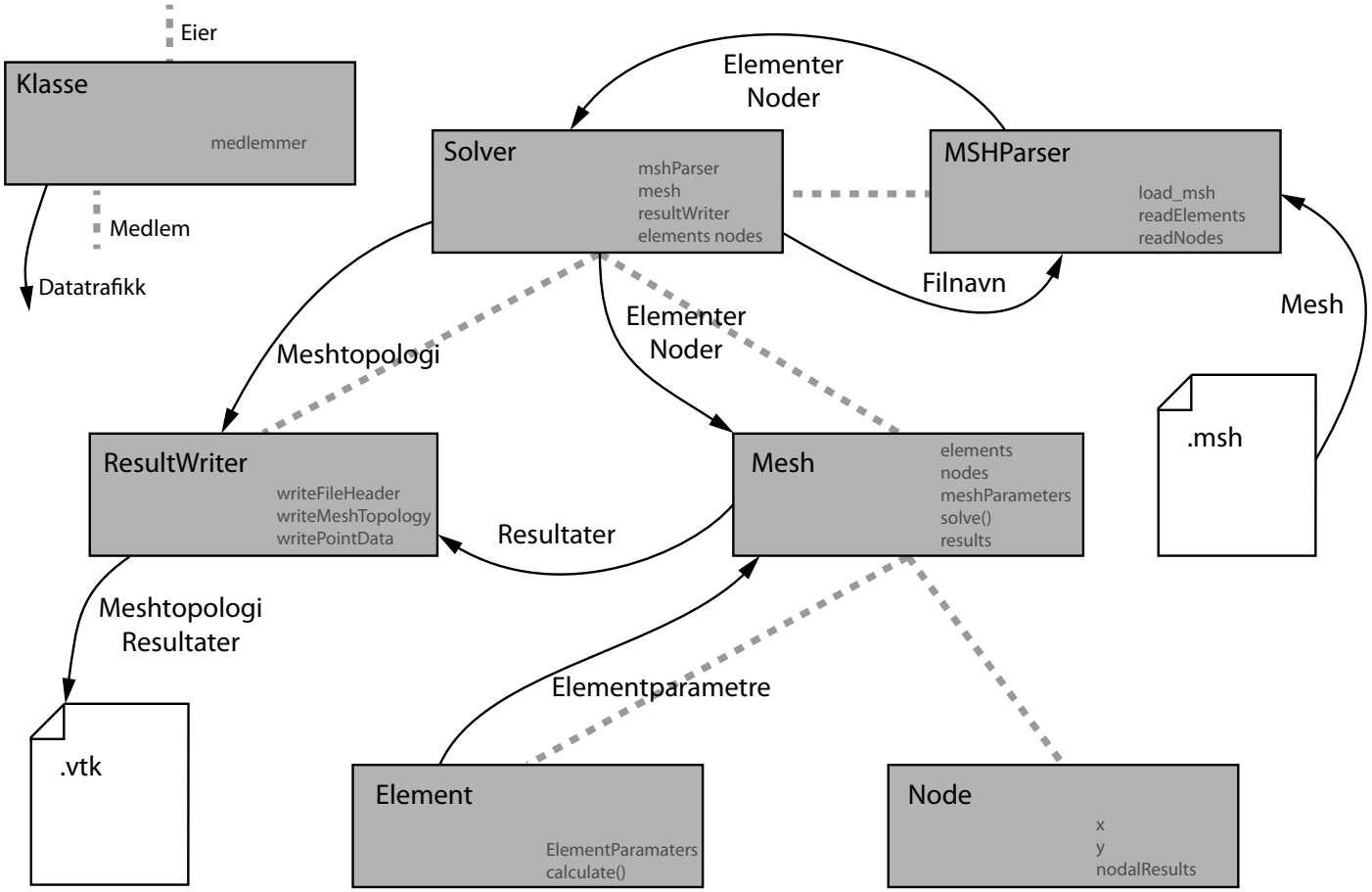
3.1 Eigen

Eigen[5] er et bibliotek for lineær algebra i C++ og er benyttet i denne oppgaven, hovedsakelig til utførelse av matrisealgebra og også til ligningsløsning som er diskutert i seksjon 3.5.

SparseMatrix-klassen SparseMatrix-klassen er en effektivisert implementsajon av «compressed column/row storage»-metoden (CCS/CRS) for sparse matriser. Ved CCS/CRS lagres alle verdier som ikke er null i én vektor hvor de adderes inn kolonnevis eller radvis. Sammen med denne vektoren lagres tre andre vektorer med pekere som indikerer hvor i matrisen de forskjellige verdiene hører hjemme. Spesielt for metoden som benyttes i Eigen er at det etterlates rom i strategiske posisjoner mellom verdiene for å minimere mengden omdisponering av minne under innaddering i matrisen. Hvor mye minne som skal «settes av» kan og bør bestemmes på forhånd der man vet anslagsvis hvor mange verdier ulik null matrisen kommer til å ende opp med.

3.2 Klassestruktur

Programmet har seks egendefinerte datatyper. Solver, MshParser, Mesh, Element, ResultWriter og Node. Et Solver-objekt deklareres i main-funksjonen og parsing av meshet og instansiering av Mesh-klassen skjer i konstruktøren til Solver. MshParser leser mesh-filen og lagrer noder og elementer i to std containere. Disse sendes som argument til Mesh-objektet som lagrer disse i egne containere. Alle beregningene gjøres i Mesh-konstuktøren med unntak av noen elementparametre som regnes ut i Element-konstruktøren når meshet parses. Ellers har Element-klassen funksjoner som kalles fra Mesh-konstruktøren for



Figur 3.1: Programmets struktur og inforsmasjonsflyt

oppsett av systemligninger. Figur 3.1 viser en forenklet oversikt over programmets struktur og informasjonsflyt.

3.3 Arbeidsflyt

Parsing av mesh Meshet parses av MSHParser-klassen som først kontrollerer at filen som har blitt gitt som argument er tilgjengelig før den kjører funksjonen load_msh() som leser nodene og elementene fra filen og lagrer dem i vektorer. load_msh() kjører også noen av funksjonene som regner ut egenskapene til elementet. calculateArea(), calculateAreaCenter() og calculateConstants(). Den siste regner blant annet ut **B**- og **G**-matrisen til elemnetet.

Solveren lagrer en kopi av disse og sender de videre som argument til Mesh-konstruktøren.

Elementer til noder I mesh-konstruktøren settes det opp to vektorer som angir hvilke elementer som er avhengige av hver node. Dette gjøres ved å kjøre en løkke over alle elementene og legge elementnummeret til hvert element inn i vektoren til hver node den er tilkoblet.

Arealegenskaper calculateMeshProperties() regner ut det totale arealet og arealsenteret til meshet, og beregner deretter annet arealmomentene. Funksjonene for annet arealmomenterne er medlemmer i Element-klassen og kjøres fra Mesh i en løkke over alle elementene. Deretter beregnes orienteringen til hovedaksene.

Orientering etter hovedaksene transformCoordinates() transformerer x- og y-koordinatene så origo sammenfaller med arealsenteret og hovedaksene med x- og y-aksen. Dette innebærer å endre parametre både i alle nodene og alle elementene ettersom begge brukes i analyser senere i programmet.

Spenninger fra bøyemoment Spenninger fra enhets bøyemoment beregnes i hver node fra forholdet mellom nodens avstand fra en akse og annet arealmomentet om samme akse. $\sigma = \frac{y}{I_x}$. Spenningene lagres i en matrise i Mesh-objektet.

3.3.1 Torsjonsanalyse

Konsistent lastvektor Lastvektoren regnes ut for hvert element. Lastvektoren for torsjonsanalysen er gitt i formel 2.17 og

$$\varepsilon_0 = \begin{bmatrix} \mathbf{0} & -\mathbf{N} \\ \mathbf{N} & \mathbf{0} \end{bmatrix} \begin{bmatrix} \mathbf{x} \\ \mathbf{y} \end{bmatrix}$$

funksjonen som finner initiellt̊oyningen bruker getNMatrix(ζ_1,ζ_2,ζ_3) som igjen kaller getShapeFunction(ζ_1,ζ_2,ζ_3).

Systemligningene I systemEquations() beregnes stivhetsmatrisen for hvert element og elementlastvektorene og elementstivhetsmatrisene adderes inn i systemvektoren og systemmatrisen. constrainNode() innfører deretter grensebetingelser.

Løsning Å løse ligningene er enkelt med Eigen.
SolverClassName<SparseMatrix<double>> solver(A) utfører faktoriseringen av **A**-matrisen og b = solver.solve(C) løser ligningen **Ab** = **C** med hensyn på **b**.

b-matrisen gir forskyvningene til systemet som deretter brukes til å finne spenningene.

Spenninger For å beregne spenningene tilegnes elementene sine lokale forskyvninger. Deretter regnes spenningene ut i hver node i computeTau() fra formelen $\tau = G(\mathbf{Bv} + \varepsilon_0)$. Spenningene for hver node lagres i en vektor.

Noder og gjennomsnitt Nå som spenningene er beregnet i hvert element kan disse adderes inn i spenningsvektorene til nodene de er koblet til. Deretter blir disse dividert på antall elementer de har knyttet til seg og ender da opp med gjennomsnittsverdien av alle elementene. Disse verdiene lagres så i matriser i Mesh-klassen for å skrives til resultatfilen.

Resultater Tilbake i Solver blir resultatene skrevet ut til skjerm og til fil. Funksjonen for å skrive resultatene til fil har en funksjon som skriver headeren i filen, en som skriver topologien til meshet og en som skriver punktdata. Topologien som skrives til filen hentes fra medlemmene i Solver-klassen, som ikke har blitt transformert. Filen lukkes og lagres.

3.4 Integrasjon

Oppretter tabeller med koordinater for integrasjonspunkter og vekttall. For integral over trekantelement i arealkoordinater blir nå

$$I = \int_A f(x,y)dA = A\Sigma_{i=1}^{n} W_i f(\zeta_{1i}, \zeta_{2i})$$

i programmet løses dette med en enkel løkke som repeteres n antall ganger.

```
double zeta[2][n] = {{zeta11, zeta12, ..., zeta1n},
                     {zeta21, zeta22, ..., zeta2n}};
double weight[n] = {W1, W2, ..., Wn};

for(int i=0; i<n; i++){
        I += weight[i] * f(zeta[i][i]);
}
I *= area
```

Ved beregning av annet arealmomenter er funksjonen som integreres en skalar, ved beregning av stivhetsmatriser og lastvektorer er funksjonene produkter av matriser. For at disse integralene skal la seg beregne defineres funksjoner som tar arealkoordinater som argument og returnerer variabelen. Disse funksjonene kan kalles fra løkken med integrasjonspunktene som argument. Konstante funksjoner krever ingen løkke eller integrasjonspunkter og ganges bare med arealet for å finne integralet.

3.5 Ligningsløsning

Ligningssystemene som løses er på formen $\mathbf{Ax} = \mathbf{b}$ der $\mathbf{A}$ er en sparse matrise men systemparametre. Eigen har en rekke ligningsløsere å velge mellom og det er egenskapene til $\mathbf{A}$ og størrelsen på systemet som er avgjørende for hvilken metode som er best å bruke. Det som hovedsakelig skiller ligningsløserene fra hverandre er metoden for å faktorisere $\mathbf{A}$-matrisen og disse kan deles inn i to kategorier. Direkte faktorisering og iterativ faktorisering. De aktuelle direkte metodene baserer seg på LU-faktorisering der $\mathbf{A}$ faktoriseres til to triangulære matriser $\mathbf{L}$ og $\mathbf{U}$ der $\mathbf{L}$ er en nedre triangulær matrise og $\mathbf{U}$ er en øvre triangulær matrise.

$$\mathbf{A} = \mathbf{LU} = \begin{bmatrix} l_{11} & 0 & .. & 0 \\ l_{21} & l_{22} & .. & 0 \\ .. & .. & .. & .. \\ l_{n1} & l_{n2} & .. & l_{nn} \end{bmatrix} \begin{bmatrix} u_{11} & u_{12} & .. & u_{1n} \\ 0 & u_{22} & .. & u_{2n} \\ .. & .. & .. & .. \\ 0 & 0 & .. & u_{nn} \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} & .. & a_{1n} \\ a_{21} & a_{22} & .. & a_{2n} \\ .. & .. & .. & .. \\ a_{n1} & a_{n2} & .. & a_{nn} \end{bmatrix}$$

Dette systemet er underbestemt og man er avhengig av å innføre en betingelse for å bestemme $\mathbf{L}$ og $\mathbf{U}$.

Ved Cholesky-faktorisering (LLT-faktorisering) gjøres dette ved å bestemme $\mathbf{U} = \mathbf{L}^T$. Systenet blir da $\mathbf{Ax} = \mathbf{LL}^T\mathbf{x} = \mathbf{b}$ og løses ved å sette $\mathbf{Ly} = \mathbf{b}$ og løse denne med hensyn på $\mathbf{y}$ og deretter løse $\mathbf{L}^T\mathbf{x} = \mathbf{y}$ med hensyn på $\mathbf{x}$.

En utvidelse av Cholesky er LDLT-metoden der det defineres en matrise $\mathbf{D}$ slik at $\mathbf{A} = \mathbf{LDL}^T$ der $\mathbf{L}$ er en enhets triangulær matrise (en matrise der alle koeffisienter langs diagonalen er 1). Ligningssystemet løses ved å sette

$$\begin{gathered} \mathbf{L}^T\mathbf{x} = \mathbf{y} \\ \mathbf{Dy} = \mathbf{z} \\ \mathbf{Lz} = \mathbf{b} \end{gathered}$$

og løse for $\mathbf{z}$, $\mathbf{y}$ og deretter $\mathbf{x}$.

LDLT er en mer fleksibel metode enn LLT som egner seg for faktorisering av matriser som ikke er SPD (symmetric positive definite) der LLT ikke lar seg anvende. Stivhetsmatrisen vil imidlertid alltid være SPD og det er derfor grunn til å tro at LLT-metoden vil være den raskeste av de to ettersom LDLT innfører et ekstra steg.

Iterative metoder tar utgangspunkt i en «gjettet» verdi $\mathbf{x}_0$ og definerer en restvektor $\mathbf{r}_0 = \mathbf{b} - \mathbf{Ax}_0$. Denne brukes for å finne $\mathbf{x}_1$ og denne prosessen

Navn	Type	CPU-tid (ms)
LDLT	Direkte faktorisering	32
LLT	Direkte faktorisering	33
Conjugate Gradient	Iterativ	517

Tabell 3.1: Test av ligningsløsere på mesh med 26450 elementer

fortsetter til et konvergenskriterie er nådd. Dette er metoder som egner seg til store systemer der direkte metoder ikke kan anvendes.

Gjennom tester av de forskjellige metodene viste det seg at LDLT er marginalt raskere enn LLT, mens den iterative metoden (conjugate gradient) brukte mer enn 15 ganger så lang tid. Tabell 3.1 viser resultater av en test på et mesh med 26450 elementer. Tiden er et gjennomsnitt over 5 tester. På bakgrunn av dette velges LDLT-metoden til å løse systemligningene.

3.6 Filformater

Brukergrensesnittet til programpakken er basert på VTK (Visualization toolkit)[8] som er et open-source software-system for 3D datagrafikk og visualisering. Det var derfor naturlig å bruke vtk-formatet til resultatfilen. Dette legger også til rette for å undersøke resultatene fra solveren uavhengig av brukergrensesnittet ettersom mange open-source visualiseringsverktøy støtter dette formatet.

Resultatene fra analysen skrives ut som skalarer, men formatet har også støtte for en rekke andre datatyper som kan benyttes ved behov.

Syntaks i vtk-formatet er som følger.

```
# vtk DataFile Version 4.0
filnavn.vtk
ASCII
DATASET POLYDATA                    //Spesifiserer datasettype
POINTS n datatype                   //n=Antall noder.
x1 y1 z1                            //nodevis x-, y- og z-koordinater.
x2 y2 z2
...
xi yi zi
POLYGONS e q                        //e=Antall elementer. q=Antall data.
p i1 j1 k1                          //kobling til node i, j og k. p=Antall noder
p i2 j2 k2
...
p ie je ke
POINT_DATA n                        //Data assosiert med noder
SCALARS dataSetNavn datatype        //Gjentas for alle data.
LOOKUP_TABLE default                //Bestemmer mapping av verdier mot farge.
data_1
data_2
```

```
...
data_n
```

Koordinatene til hver node skrives under «POINTS», og koblingen mellom elementer og noder angis under «POLYGONS». Her angir p hvor mange noder det er i hvert element, og q angir lengden på hele datablokken. Skalar data er angitt node for node i listen under «SCALARS». «DataSetNavn» er navnet som vises i listen over tilgjengelig data i brukergrensesnittet. Et eksempel på resultatfil er gitt i tillegg B.

3.7 Bruk

Solveren kan brukes også uten brukergrensesnittet, ved å kjøre det fra kommandovindu. Dette gjøres ved å finne lenken til mesh-filen og gi denne, og eventuelt E-modul, som argument på følgende måte.

```
C:\Path\Solver -f filePath\filnavn -e <E-modul>
```

Resultatfilen blir opprettet i samme mappe og kan åpnes i alle visualiseringsverktøy som støtter VTK. Ved bruk sammen med brukergrensesnittet blir dette gjort via en enkelt knapp, og resultatfilen kan fortsatt hentes og brukes uavhengig av brukergrensesnittet.

3.8 Dokumentasjon

For å legge forholdene til rette for at prosjektet skal videreføres er det viktig at det kommer godt frem hva de forskjellige funksjonene i koden gjør og hvordan de forholder seg til hverandre. To forutsetninger for å oppnå dette er god dokumentasjon og annotasjoner i kildekoden.

Denne prosessen kan til en viss grad automatiseres ved hjelp av Doxygen[6]. Doxygen er et program som genererer dokumentasjon fra annoterte kildekoder og setter opp klasselister og diagrammer etter ønske. Annotasjonene kan formateres som LaTeX-kode og Doxygen samler disse i et dokument som senere kan eksporteres til pdf. Det er også mulighet til å eksportere dokumentasjonen til HTML. Dokumentasjonen som ligger vedlagt er en pdf basert på LaTeX.

Kapittel 4

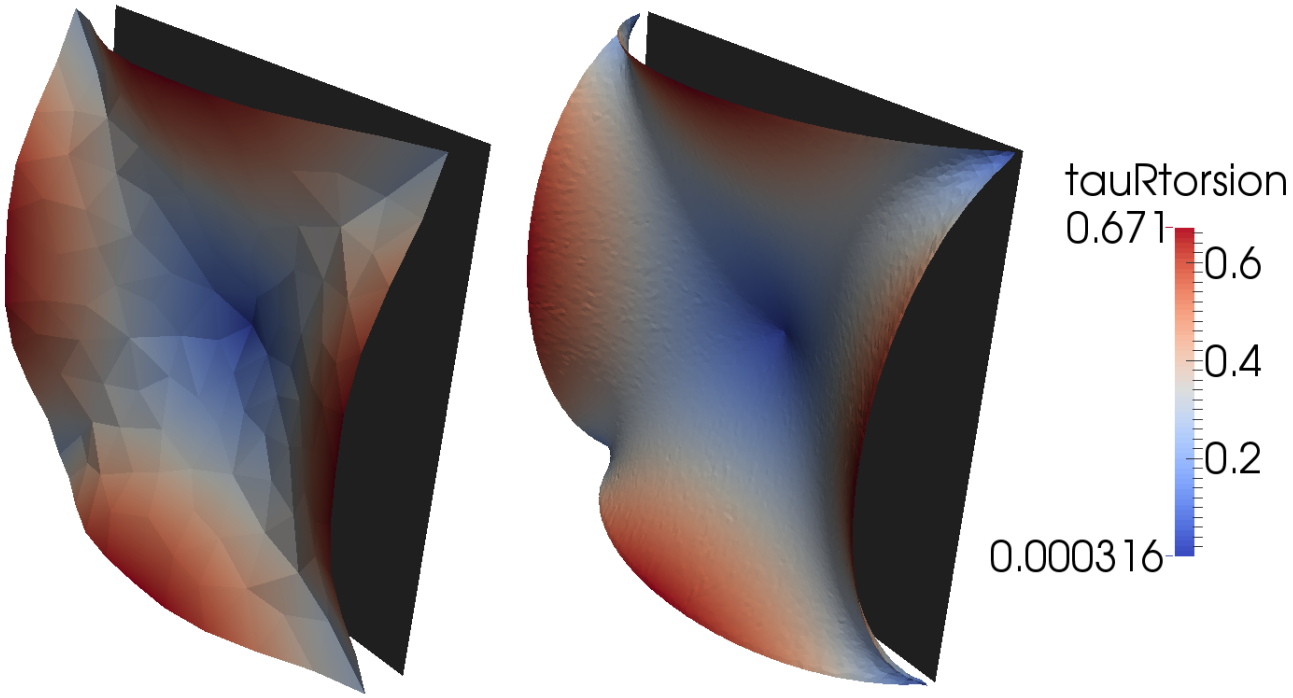
Verifikasjon

Under utviklingen av solveren har det vært nødvendig å evaluere utskrift fra programmet fortløpende for å vurdere gyldigheten av funksjonene. Noen av parametrene som må evalueres kan printes ut direkte til skjerm. Dette gjelder verdier for eksempelvis annet arealmomenter og hovedaksenes orientering. Andre verdier, som varierer over tverrsnittet, blir vanskeligere å evaluere manuelt ettersom antall noder i meshet, og dermed datamengden, øker. For å visualisere resultatene er Paraview tatt i bruk. Paraview bygger på VTK-platformen og har muligheter for å visualisere spenninger og forskyvninger som en deformert flate.

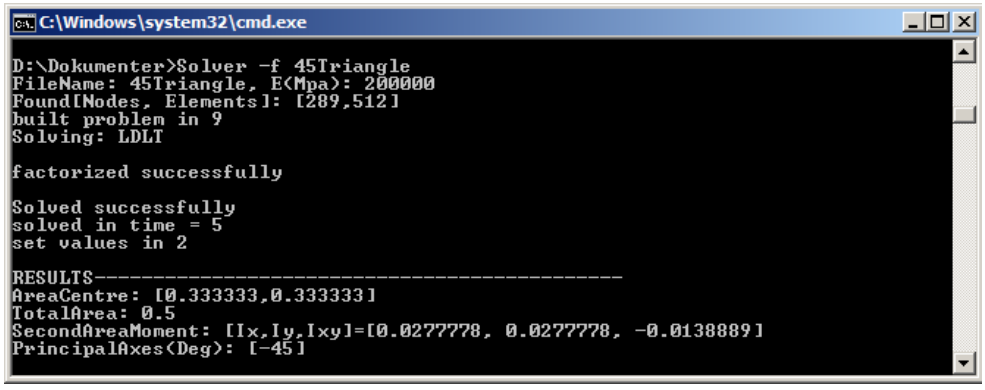
For å evaluere validiteten i resultatene er programmet blant annet testet på geometrier med kjente løsninger. Rektangler har en enkel løsning for annet arealmomenter og det er forholdsvis lett å forutse hvordan spenningsfordelingene blir under torsjonslast. Figur 4.1 viser resultater fra to analyser der den ene er gjennomført med et grovt mesh og den andre med et fint mesh. Det viser seg at disse sammenfaller innenfor en viss avstand fra ytterkanten, mens helt i randen gir det grove meshet mindre nøyaktige resultater. Helt i hjørnene skal skjærspenningen være null og mens resultatet fra det fine meshet nærmer seg rett verdi ligger det grove meshet mye høyere. Dette kan skyldes at spenningene hentes ut i nodene i stedet for i integrasjonspunktene. Flere plott av resultater er vist i tillegg A.

For å teste funksjonene som gir arealsenter og orienteringen til hovedaksene benyttes geometrier som er rotert en gitt vinkel om z-aksen og forskøvet en gitt avstand fra senter for å se om funksjonene returnerer samme verdie. Figur 4.2 til 4.4 viser resultater fra et utvalg av slike tester. Disse testene viser gode resultater på beregning av hovedakser der mesh-geometrien er bestemt eksakt. Avviket i figur 4.3 skyldes formodentlig avrunding av koordinatene når geometrien ble bestemt.

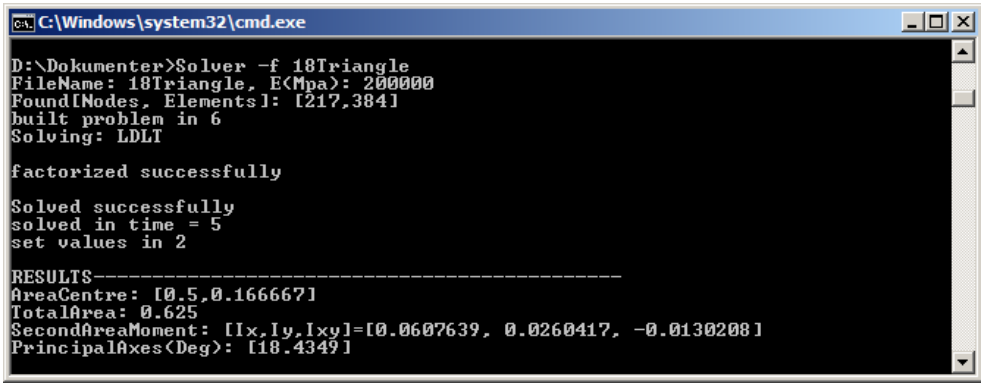
Et firepunkts integrasjonsmønster gitt av Bell[1] Appendix C ble testet som et alternativ til integrasjonspunktene gitt i seksjon 2.4, og de to ble funnet til å gi identisk løsning. Dette blir tolket som at begge metodene er gode og trepunktsintegrasjonen er valgt for å holde arbeidsmengden så lav som mulig.



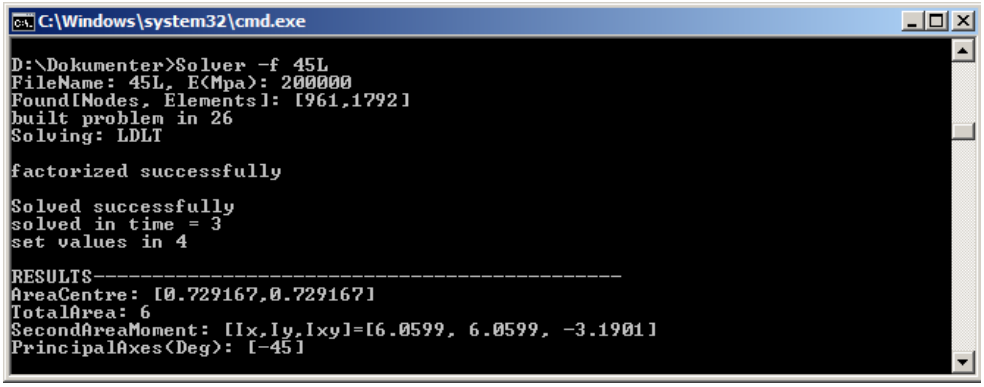
Figur 4.1: Grovt mesh og fint mesh. Spenningsresultant ved torsjon



Figur 4.2: Analyse av trekant rotert 45 grader



Figur 4.3: Analyse av trekant rotert 18 grader



Figur 4.4: Analyse av L-bjelke

Kapittel 5

Konklusjon

Ved prosjektperiodens slutt er kan solveren beregne

- spenningsfordeling ved aksialkraft og rene bøyemomenter
- hovedaksers orientering
- spenningsfordelinger ved torsjonsbelastning

Det er også lagt til rette for lett implementering av skjærkraftberegninger.

Beregningene ser ut til å være korrekte selv om grove mesh gir noe avvik i randene på torsjonsanalysen. Avviket kan muligens utbedres ved å gå over til å hente ut spenninger i integrasjonspunktene. Programmet regner spenningsfordeling ved bøyemomenter og hovedaksenes orientering korrekt.

Kapittel 6

Videreføring

Som det står nevnt i seksjon3.5 bruker programmet med LDLT-metoden i gjennomsnitt 32 ms på å løse et ligningssystem med 26450 elementer. Til sammenligning går det 6718 ms til å sette opp systemligningene på samme mesh. Før nye funksjoner implementeres bør det undersøkes om dette kan effektiviseres. De eksisterende funksjonene er tilpasset gjenbruk slik at implementering av funksjoner for skjærkraftanalyse skal være enkelt.

Kompatibilitet med flere filformater vil øke fleksibiliteten og anvendbarheten til programpakken så for fremtiden vil det være naturlig å satse på støtte for flere mesh-formater.

Dersom funksjonalitet for håndtering av kompositttmaterialer skal implementeres anbefales det at disse baseres på en tynnvegget beregningsmodell da dette vil tilby en større fleksibitet i forbindelse med brukergrensesnitt og angivelse av materialegenskaper.

Bibliografi

[1] Bell, K. (2013). An engineering approach to finite element analysis of linear structural mechanics problems. Akademika Publishing. ISBN 978-82.321-0268-6

[2] Askevold, B. (2014). Applikasjon for beregning av tverrsnittsparametre for komplekse bjelketverrsnitt. Masteroppgave NTNU.

[3] Lafore, R. (2002). Object-Oriented Programming in C++. Sams Publishing. ISBN 978-0-672-32308-9

[4] Hughes, T. (2000). The Finite Element Method: Linear Static and Dynamic Finite Element Analysis. dover civil and mechanical engineering. ISBN 978-0486411811

[5] Eigen. C++ template library for linear algebra. <www.eigen.tuxfamily.org>

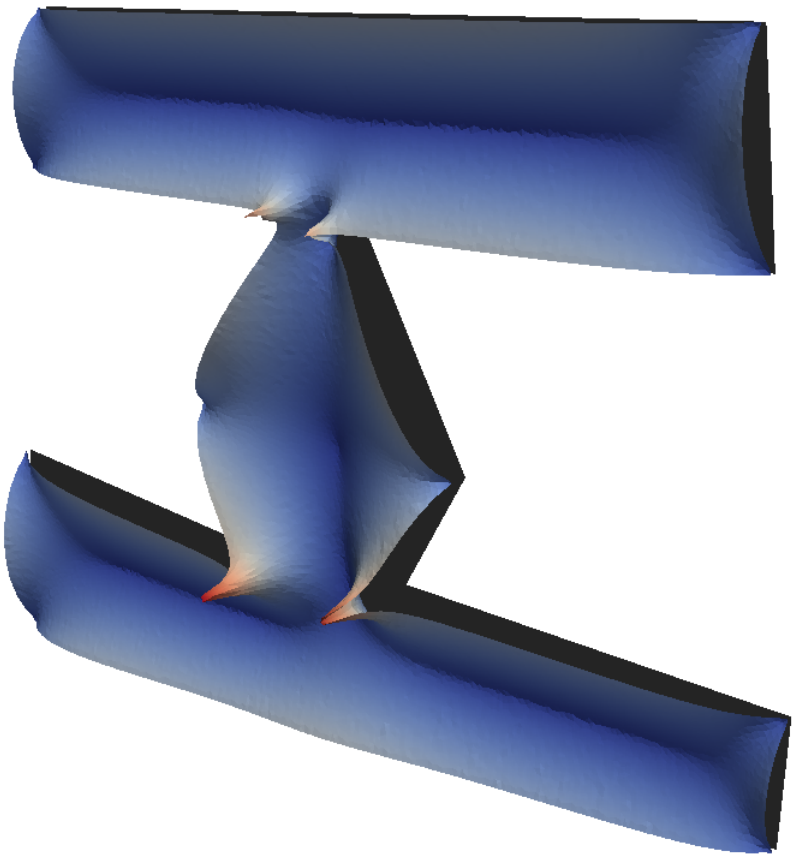
[6] Doxygen. Generate documentation from annotated source code. <www.doxygen.org>

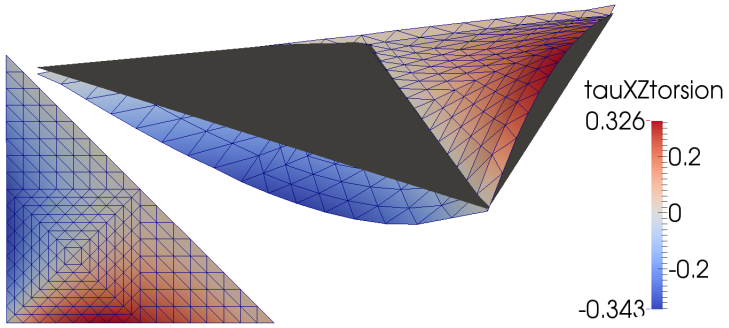
[7] Paraview. Visualization of large data sets. <www.paraview.org>

[8] VTK. Kitware, Visualization toolkit. <www.vtk.org>

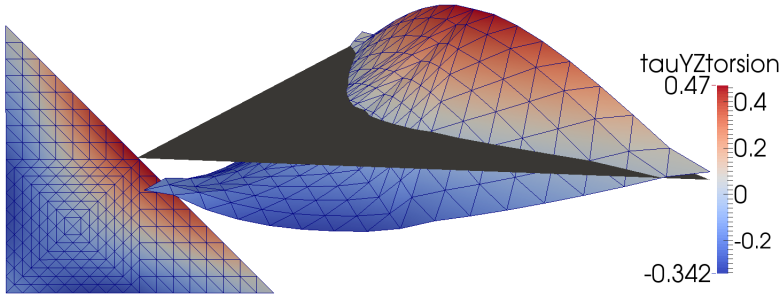
Tillegg A

Testresultater

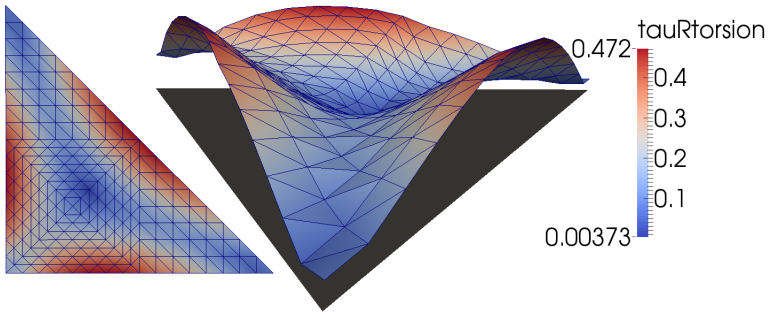




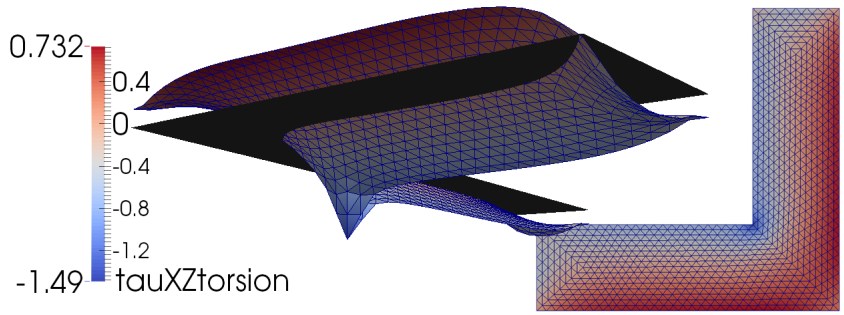
Figur A.1: Skjærspenning x-komponent ved torsjon av trekantet tverrsnitt



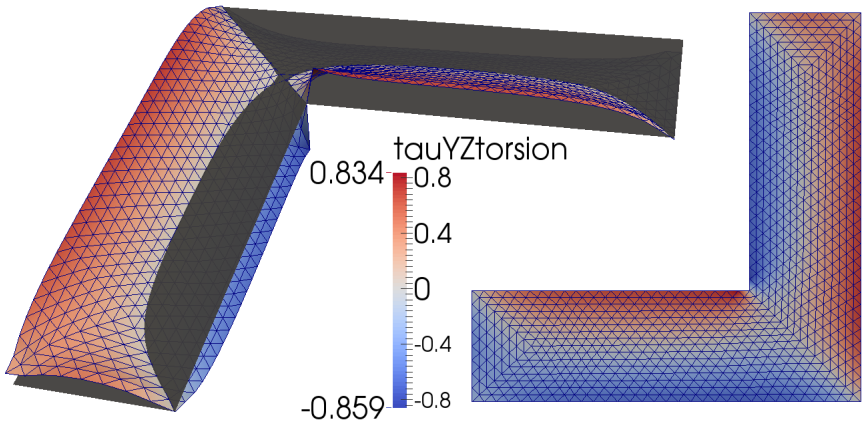
Figur A.2: Skjærspenning y-komponent ved torsjon av trekantet tverrsnitt



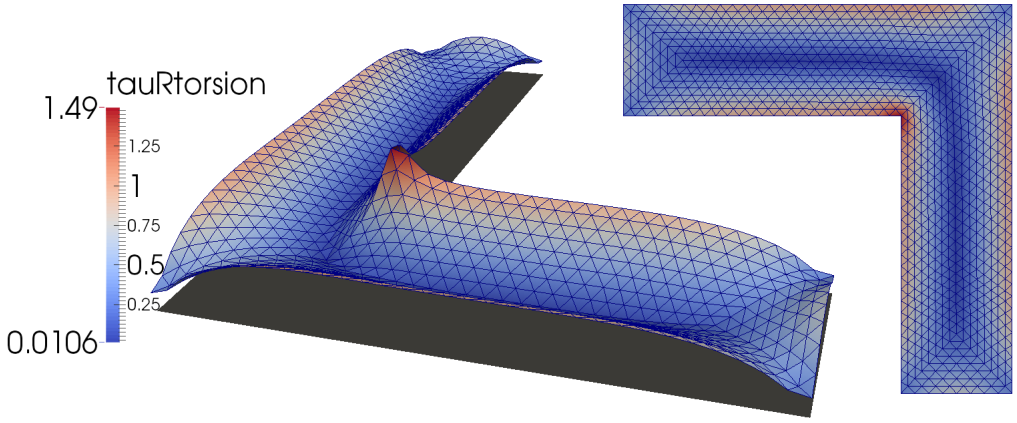
Figur A.3: Skjerspenningsresultant ved torsjon av trekantet tverrsnitt



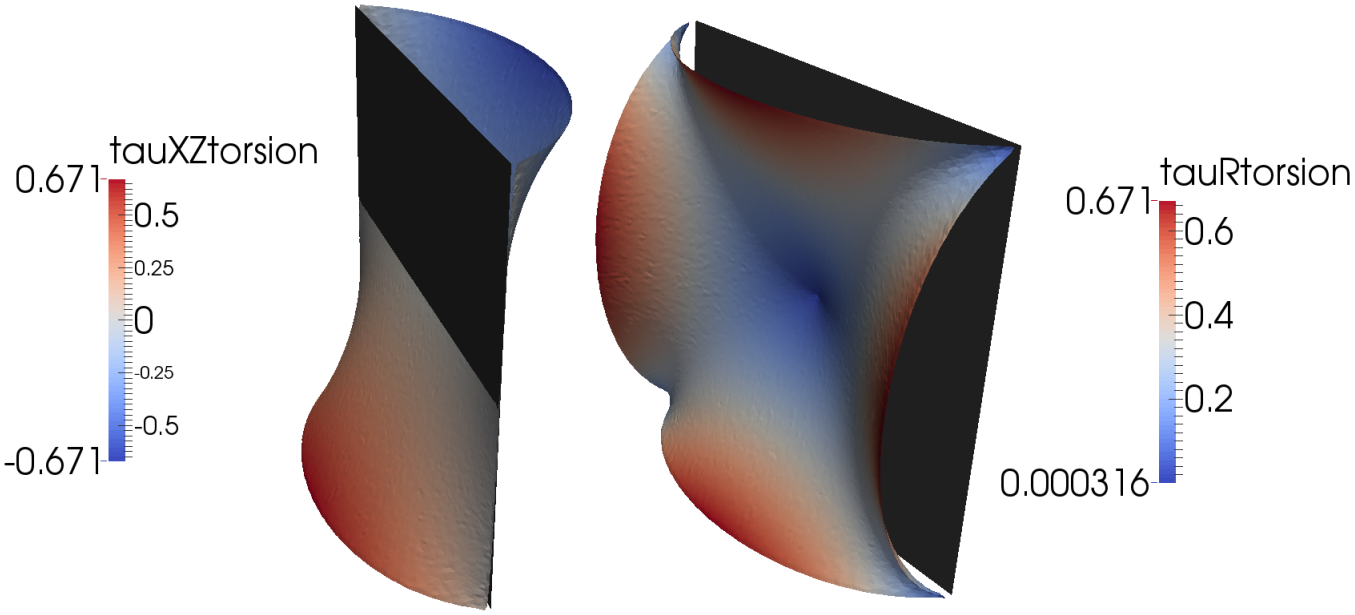
Figur A.4: Skjærspenning x-komponent ved torsjon av L-bjelke



Figur A.5: Skjærspenning y-komponent ved torsjon av L-bjelke



Figur A.6: Skjærspenningsresultant ved torsjon av L-bjelke



Figur A.7: Skjærspenningsfordelinger ved torsjon av finmeshet rektangel

Tillegg B

Resultatfil eksempel

```
# vtk DataFile Version 4.0
ExampleMesh/testbeam.vtk
ASCII
DATASET POLYDATA
POINTS 6 double
1 -1 0
0 1 0
1 1 0
0 -1 0
1 0 0
0 0 0
POLYGONS 4 16
3 4 1 2
3 1 4 5
3 4 3 5
3 3 4 0
POINT_DATA 6
SCALARS tauR double 1
LOOKUP_TABLE default
1.88562
1.2693
1.88562
1.2693
1.5
0.666667
```

Tillegg C

Mesh-fil eksempel

```
$MeshFormat
2.2 0 8
$EndMeshFormat
$Nodes
6
1 1 -1 0
2 0 1 0
3 1 1 0
4 0 -1 0
5 1 0 0
6 0 0 0
$EndNodes
$Elements
14
1 15 2 0 2 1
2 15 2 0 5 2
3 15 2 0 6 3
4 15 2 0 7 4
5 1 2 0 3 4 1
6 1 2 0 4 3 5
7 1 2 0 4 5 1
8 1 2 0 6 2 3
9 1 2 0 7 2 6
10 1 2 0 7 6 4
11 2 2 0 9 5 3 2
12 2 2 0 9 2 6 5
13 2 2 0 9 5 6 4
14 2 2 0 9 4 1 5
$EndElements
```